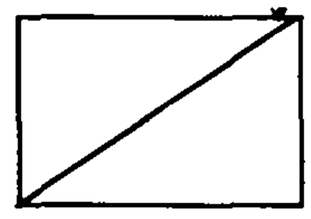
GOVP1199711667

제 1 단 계
최종보고서



RTS 기술개발

RTS Technology Development

연 구 기 관
포 항 공 과 대 학 교
시 스 템 공 학 연 구 소
UCI

과 학 기 술 처

제 출 문

과 학 기 술 처 장 관 귀 하

본 보고서를 "RTS 기술 개발" 과제의 최종 보고서로 제출합니다.

1997. 9. 3.

주관연구기관명 : 포항공과대학교

총괄연구책임자 : 박 찬 모

연 구 원 : 박 찬 모

" : 강 교 철

" : 박 찬 익

협동연구기관명 :시스템공학 연구소

협동연구책임자 :오 영 배

위탁연구기관명 : Univ. of California, Irvine

위탁연구책임자 :김 광 회

여 백

요 약 문

I. 제목

RTS 기술 개발

II. 연구개발의 목적 및 필요성

컴퓨터 하드웨어 성능의 급격한 향상으로 인해 컴퓨터 응용 분야가 급속히 넓어지고 있다. 이러한 결과로부터 거대하고 복잡한 시간적 특성을 지니는 컴퓨터 응용 시스템에 대한 요구가 증가되고 있다. 이러한 복잡한 컴퓨터 응용 시스템은 시스템의 기능적인 측면뿐만 아니라 시간적인 측면도 분석 시에 고려되어야 한다.

대규모 실시간 시스템은 핵 발전소, 항공 제어 시스템 그리고 교통 통제 시스템과 같이 안전성과 신뢰도가 중요시 되는 분야에 적용된다. 이러한 시스템의 오류적 행동은 커다란 사회 문제를 야기시킬 수 있기 때문에 실시간 시스템은 충분한 검증 단계를 거쳐서 개발되어야 한다.

시뮬레이션은 시스템의 개발 단계에서 실시간 시스템의 여러 성질을 효과적으로 검증하는데 사용된다. 특히 시뮬레이션은 단말 사용자에게 최종 시스템의 모습을 보여줄 수 있다.

이러한 기술은 컴퓨터 게임, 교육 등의 타 산업에 효과적으로 적용되어 왔다. 미국, 유럽, 일본과 같은 선진국은 시뮬레이터 도구 기술을 개발하여 개발도상국에 수출하고 있다. 하지만 선진국에서도 기본 기술을 완전히 정립하지 못하였으므로 필요한 응용 분야만 개발하며 자체 보유 기술을 개발도상국에 이전하는 것을 매우 꺼리고 있는 실정이다.

현재 한국에서는 핵발전소, 항공 시스템과 같은 실시간 시스템에 대한 요구가 나날이 증대되고 있다. 하지만 국내 기술은 아직까지 초기 단계에 머물고 있으며 대부분 수입된 기술에 의존하는 실정이다. 지금까지 한국에서는 수입된 시뮬레이터의

운영 기술의 확보와 기존 시스템의 확장을 통한 개발 기술의 확보에 치중하였다. 하지만 이러한 상황에서는 기본 기술조차 얻기에 힘든 실정이다.

본 연구에서는 특정 분야의 시뮬레이터를 개발하는데 치중하는 것을 지양하고 실시간 시뮬레이션 기반 기술을 정립하고 개발 환경을 구축하여 선진국과의 경쟁력을 확보하는데 중점을 두고 있다.

III. 연구개발의 내용 및 범위

본 연구에서 수행한 개발 내용 및 범위는 다음과 같이 요약할 수 있다.

- 실시간 시뮬레이션 사례 연구
  - 실시간 시뮬레이션의 분류
  - 대규모 실시간 시뮬레이션 구축 운영 사례 연구
  - 국내의 시뮬레이션 활용현황 조사
  - 국내의 시뮬레이션 상용도구 조사
- 실시간 시뮬레이션 모델 개발
  - 거시적 및 미시적 시뮬레이션 모델 개발
  - 실시간 시뮬레이션 모델링 방법론 개발
  - 실시간 시뮬레이션 언어 개발
  - 시뮬레이션 개발 도구 개발
- 실시간 시뮬레이션 엔진 개발
  - 거시적 및 미시적 시뮬레이션 수행기 개발
  - 분산 시뮬레이션 수행기 개발
  - 실시간 분산 프로토콜 개발
  - 실시간 마이크로 커널 개발
- 실시간 시뮬레이터 구축

- 압연공정 시뮬레이터 개발
- 실시간 그래픽 사용자 인터페이스 개발
- 거시적 및 미시적 시뮬레이터의 통합

IV. 연구개발 결과

본 연구의 3 개년간의 연구 수행을 통해 다음과 같은 연구 결과를 도출하였다.

1. 실시간 시뮬레이션 사례연구를 통해 실시간 시뮬레이션이 발전소, 공장 등 현재 산업의 많은 부분에서 핵심적인 역할을 수행하고 있음을 파악할 수 있었고 국내의 시뮬레이션 활용 현황 및 시뮬레이션 상용도구를 조사하였다.
2. 실시간 시뮬레이션 모델 개발의 연구 결과는 다음과 같다.
   - 사용자 요구사항으로부터 시스템 모델을 만드는 방법 개발
   - 복잡한 실시간 시스템의 기능적, 시간적 행동을 추계적으로 모델링하고 시뮬레이션 할 수 있는 방법 개발
   - 모델을 작성할 수 있고 작성된 모델을 대화형으로 수행시켜볼 수 있는 도구를 개발
   - 객체지향 실시간 모델링 방법 개발
   - 실시간 시뮬레이션 개발 도구 개발
3. 실시간 시뮬레이션 엔진 개발에 대한 연구 결과는 다음과 같다.
   - 다중 우선 순위 스케쥴러 개발
   - 병렬 로봇 시뮬레이터의 개발
   - 거시적 및 미시적 시뮬레이션 수행기 개발
   - 분산 시뮬레이션 수행기 개발
4. 실시간 시뮬레이터 구축에 대한 연구 결과는 다음과 같다.
   - 압연공정 실시간 시뮬레이터 개발

- VR 기술을 이용한 실시간 삼차원 그래픽 사용자 인터페이스 개발
- 거시적 시뮬레이터와 미시적 시뮬레이터의 통합

V. 연구개발 결과의 활용계획

본 연구의 결과는 실시간 시뮬레이션 응용의 기반 기술로 이용할 수 있다. 국내 산업계에 대규모 실시간 시스템이 활발하게 보급되고 있는 현실을 감안하면 이 기술은 국내 기업에 널리 보급 활용할 필요가 있다고 본다.

본 연구를 통해서 확보한 기술은 학술 세미나나 학술회의 또는 워크숍 개최 등을 통해서 여러 차례 발표하였고 향후에도 이들 활동을 지속할 예정이며, 기업에 대한 기술이전도 적극적으로 추진할 예정이다.

본 연구의 2 단계 사업이 진행되면 1 단계에서 연구 개발한 기술을 고정밀화, 실시간 검증기능의 강화 등 고정도의 기술로 발전 시킬 계획이며 개발 기술의 기업에의 적용에 중점을 둘 예정이다.

Summary

I. Title

Development of RTS technology

II. Objectives and Necessity

Areas of computer application are being broadened rapidly due to the rapid improvement of the performance of computer hardware. This results in increased demands for computer applications that are large and have complex temporal characteristics. For those complex computer applications, the temporal aspect as well as the functional aspect of the systems must be taken into considerations in the analysis.

Large-scale real-time systems are being developed in the fields, such as nuclear power plants, flight control systems and traffic control systems, where safety and reliability are a primary concern. System in these areas must be developed with sufficient verification, as any erroneous behavior of such a system could lead to serious problems.

Simulation is one effective technologies verifying various characteristics of real-time systems during the development. Especially, simulation can provide a look and feel of a target system to the end users.

This technology has been applied to various fields such as computer game, education, etc effectively. Developed countries, such as USA, Europe, and Japan have been developing simulation tools technology for exportation. However, because even they have not established the basic technology completely, they tend to develop application oriented tools and also are not willing to transfer their technologies to other countries.

In Korea, needs for real-time systems such as nuclear power plants and flight systems are

gradually increasing recently. Howerver out technology is at an early stage of development, and we depend mostly the imported technologies. Until now most of development of effort have been on adapting imported technologies and this make it difficult to develop our own technologies.

In this research, we are concentrating on developing general and basic technologies for real-time simulation and building the development environment rather than developing application-oriented simulator.

III. Content and Scope of the study

The content and the scope of the study are discussed below.

- A case study of real-time simulations
  - Classification of real-time simulations
  - A case study of the implementation and maintenance of large-scale real-time simulations
  - Survey of simulation practice in Korea
  - Survey of off-the-shelf tools for simulation in Korea
- Development of a real-time simulation modeling
  - Development of macroscopic and microscopic simulation models
  - Development of a real-time simulation modeling methodology
  - Development of a real-time simulation language
  - Development of simulation development tools
- Development of a real-time simulation engine
  - Development of macroscopic and microscopic simulation engines
  - Development of a distributed simulation engine
  - Development of a real-time distributed protocol
  - Development of a real-time micro kernel

- Implementation of a real-time simulator
  - Development of a rolling mill simulator
  - Development of a real-time graphic user interface
  - Integration of macroscopic and microscopic simulators

IV. Research Results

The followings are the outcome from the three years of research:

1. From the case study of real-time simulation, we can identify requirements of the simulator in many fields of industry such as, a power plant, a factory etc. and we have surveyed simulation practice and simulation off-the-shelf tools.
2. Development of a real-time simulation model
   - Development of a method constructing a system model from user's requirements
   - Development of a method to be able to model and simulate the temporal and functional behavior of a complex real-time system
   - Development of a tool to be able to construct a model and execute it interactively
   - Development of a object oriented real-time modeling method
   - Development of a real-time simulation development tools
3. Development of a real-time simulation engine
   - Development of a multi-priority scheduler
   - Development of a parallel robot simulator
   - Development of macroscopic and microscopic simulation engine
   - Development of a distributed simulation engine
4. Development of real-time simulator
   - Development of a rolling mill real-time simulator
   - Development of a real-time three dimensional graphic interface using VR technology
   - Integration of macroscopic and microscopic simulators

V. Future Works

A result of this R&D work can be used as a basic technology for real-time simulation application. In view of vitally proliferation of large-scale real-time systems to the Korean industries, we think this technology should be proliferated and used widely in the Korean industries.

We have published this technology, acquired from this study, through academic seminars and conferences or holding workshops several times and we will also continue these works in the future. In particular, we will try to transfer technology to industries eagerly.

With the progress of the second phase of this study, we will make this technology stepping up by making it have better high fidelity and expand verifying function and we will focus to apply this technology to industries.

Contents

여 백

총 목 차

여 백

제 1 편

총 론

여 백

목차

여 백

1. 연구의 목표

연차별 연구 개발 목표 및 내용은 다음과 같다.

연차별	연 구 개 발 목 표	연 구 개 발 내 용 및 범 위
1차년도	- 대규모 실시간 시뮬레이션 조사 - 실시간 시뮬레이션 모델 정립	- 현존 대규모 실시간 시스템 조사 및 시뮬레이터 사용 현황 조사 - 실시간 시뮬레이터의 특성 연구 - 특정 실시간 응용시스템의 사례연구 - 실시간 객체지향 모델링 기법 개발 - 실시간 객체지향 모델링 언어 정의 - 모델의 검증 및 조정 - 거시적 실시간 시뮬레이터 모델 방법 - 시뮬레이터 프로그램 변환 기술 개발
2차년도	- 시뮬레이션 모델 처리기 개발 - LAN용 실시간 지원 기능 개발 - 병렬 처리 지원 기능 개발 - 안전성, 생존성 등의 중요 실시간 성질의 정형적 검증 방법 개발 - 사용자 지향의 시뮬레이션 제어 언어 개발 및 통계적 시뮬레이션 방법 개발	- 문법검증기(syntax analyzer)개발 - C++T 전처리기(preprocessor) 개발 - 시뮬레이터 프로그램 수행에 필요한 LAN용 실행지원 기능 개발 - 실시간 성질의 정확한 검증이 요구되는 시스템을 위한 정형적 검증방법의 개발과 검증결과를 시뮬레이션으로 시각화 - 사용자 지향의 시뮬레이션 제어 언어 개발 - 통계적 시뮬레이션 개발 - 병렬처리컴퓨터 기반의 run-time support 기능 개발 - 지능형 생산공장 시뮬레이션 모델 구축 - 거시적 시뮬레이터의 그래픽 인터페이스 개발
3차년도	- 병렬처리 컴퓨터용 지원기능 추가 - 그래픽 사용자 인터페이스 개발 - 자료의 시각(visualization) 기능 및 지능형 생산공장의 시뮬레이터 구축	- 미시적 시뮬레이션 모델에 병렬처리 지원 기능 추가 - 병렬처리에 대한 시뮬레이터 실행지원(run-time support) 프로그램 개발 - 병렬처리 지원기능의 최적화를 위한 기술 개발(정적/동적 분석 방법 개발) - 그래픽 시뮬레이션 모델링 도구 개발 - 그래픽 시뮬레이션 결과분석 도구 개발 - 지능형 생산공장의 시뮬레이터 프로토타입 구축 - 시뮬레이션 시범 적용 - 실행결과 분석 및 평가

2. 연구 수행 방법

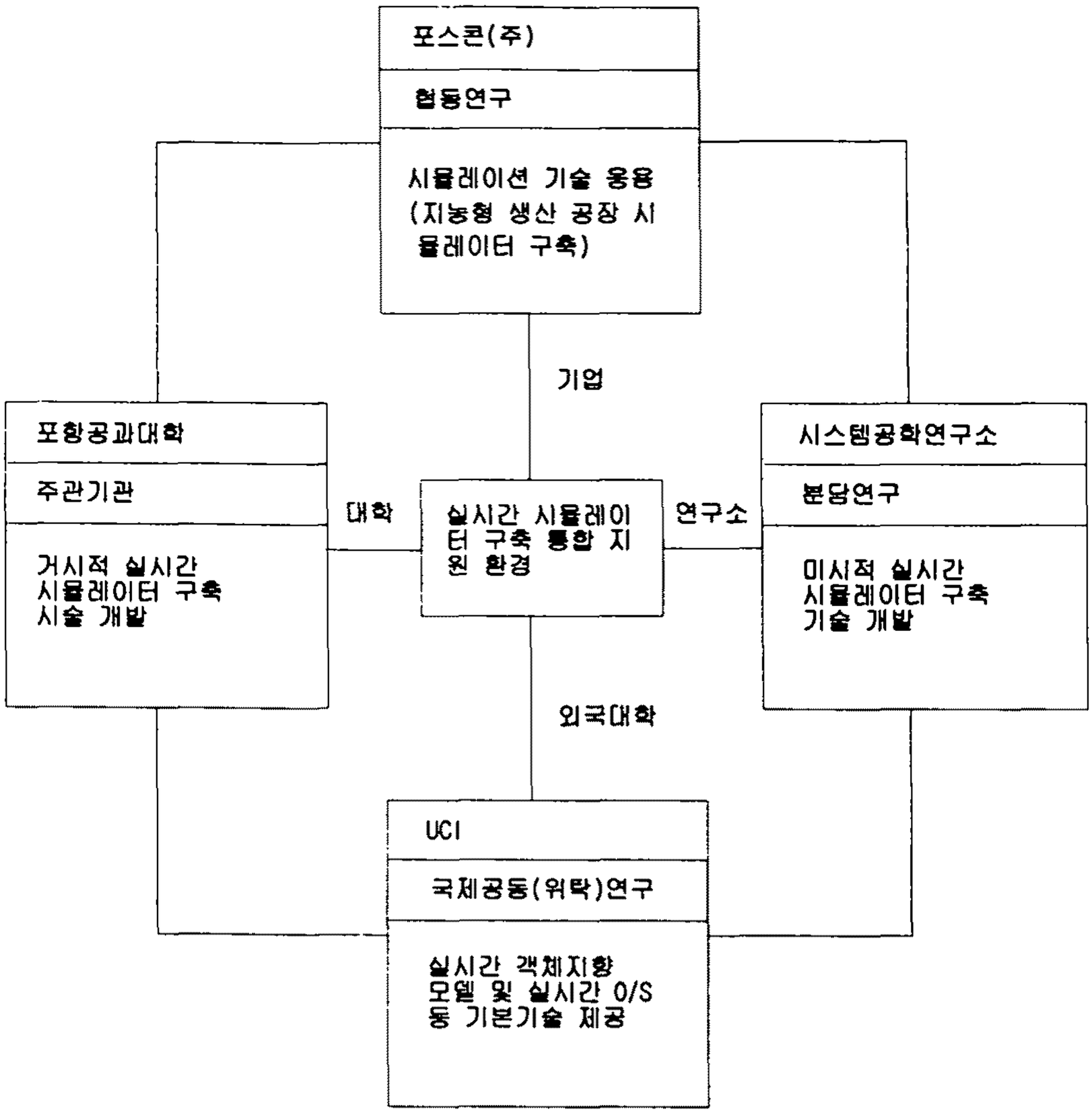
가. 연구의 방향

- 특정 분야의 시뮬레이션 구축에 편향되지 않는 기초 기술 확립
- 실시간 모델링 및 개발환경 구축에 주안점을 두는 포괄적, 체계적인 접근 방향 지향
- 실시간 모델 코딩의 용이성을 확보하기 위한 실시간 객체 지향 언어 개발
- 실시간 객체의 적시성 보증을 지원하는 시뮬레이션 엔진 개발
- 국제 공동 연구 체제 구축

  세계적 수준의 연구업적을 쌓은 미국 UCI 실시간 연구팀에 연구원 파견 공동연구
- 미국과 한국간의 Internet 자원을 통한 온라인 연구망 구축

  산학연 공동 연구 체제 구축
- 기초기술은 학계에서, 개발기술은 연구소에서 문제 영역도출은 산업계에서 맡는 역할 분담 체제 구축
- 참여기업의 적극적인 참여 유도

나. 연구개발 체계도



3. 연구 내용 및 결과

가. 1차년도 연구 내용

- 실시간 시뮬레이션 이용사례
  - 실시간 시뮬레이션의 정의 및 분류
  - 실시간 시뮬레이션의 적용 사례
- 거시적 실시간 시뮬레이터 구축 기술 개발

- ASADAL 방법론
- 명세 방법
- 시뮬레이션 방법
- ASADAL CASE 도구

● 다중 우선순위 스케쥴러 및 병렬 로봇 시뮬레이터 연구
- 병렬 시스템에서의 다중 로봇 시뮬레이션
- 트랜스퓨터 상에서 실시간 커널을 위한 다중 우선순위 스케쥴러

● 미시적 실시간 시뮬레이터 구축 기술 개발
- 실시간 시뮬레이션의 연구 개발 현황
- 실시간 시뮬레이션 모델
- 미시적 실시간 시뮬레이션의 적용
- 실시간 객체 지향 언어
- 실시간 시뮬레이션 엔진의 설계

나. 2차년도 연구 내용

● 거시적 실시간 시뮬레이터 구축 기술 개발
- 사용자 대화형 시뮬레이터
- 추계적인 배치형 시뮬레이터
- ASADAL CASE 도구 기능의 확장
- 정형적인 실시간 성질 검증 방법
- 지능형 생산공장 시뮬레이션 모델

● 실시간 그래픽 사용자 인터페이스 개발
- 실시간 3차원 가시화

- 가상 메뉴
- 가상도구
- 사용자와의 상호작용

● 실시간 마이크로 커널 연구

- 마이크로 커널
- 다중 우선순위 스케쥴러

● 미시적 실시간 시뮬레이터 구축 기술 개발

- 압연공정 AGC 모델링
- 시뮬레이션 모델 설계
- 압연공정 AGC 시뮬레이터의 구현
- LAN용 실행지원 기능
- 실시간 객체지향 언어

다. 3차년도 연구 내용

● 거시적 실시간 시뮬레이터 구축 기술

- 분산 시뮬레이션 기술 및 그 도구
- 거대, 복잡한 명세의 시뮬레이션
- 하향식 시뮬레이션
- 분산 명세 방법
- 데이터 분석기
- 테스트 분석 방법
- 자동 두께 조절 시스템 및 장력 시스템의 개발

● 미시적 실시간 시뮬레이터 구축 기술

- 그래픽 시뮬레이션 접속 기능 개발
- 그래픽 데이터 처리기 개발
- 브리지 노드의 개발
- 압연공정 다스탠드 시뮬레이션 모델 설계
- 압연공정 다스탠드 시뮬레이터 개발

제 2 편

제1차년도 연차보고서

여 백

목 차

여 백

제 1 차년도
연차 보고서

RTS 기술개발

RTS Technology Development

거시적 시뮬레이터 구축 기술 개발

연구기관
포항 공과 대학교

과 학 기 술 처

여 백

제 1 장 실시간 시뮬레이션 이용 사례

1 절 서론

시뮬레이션은 과학, 기술 분야에서뿐 아니라 경제, 사회등 많은 분야에서 사용되어 왔다[1].그런데, 이 시뮬레이션의 대상중에서 실시간 시스템 (real-time system)을 대상으로 하는 실시간 시뮬레이션은 그 결과를 정해진 시간내에 얻어야 하는 이유 때문에 성능이 좋은 컴퓨터를 요구한다. 그런데 성능이 좋은 컴퓨터를 이용할 수 없었던 과거에는 실시간 시스템의 모델링을 실제에 가깝게 할 수가 없고 많은 가정과 이상화 및 단순화를 하여 모델링을 하고 시뮬레이션하였다. 따라서 이러한 시뮬레이션의 결과는 정확도가 떨어질 뿐아니라 복잡한 시스템에 대해서는 시뮬레이션을 수행할 엄두를 못내었다. 그러나 근래에는 PC 및 워크스테이션의 성능이 매우 높아져서 이들을 이용한 시뮬레이션이 활발하게 진행되고 있으며 특히 고성능 병렬처리 시스템 (massively parallel processors)의 출현으로 싼 가격에 방대한 양의 계산을 신속하게 처리할 수 있게 됨으로써 실시간 시뮬레이션을 많은 분야에 적용할 수 있게 되었다. 본 장에서는 실시간 시뮬레이션이 적용되고 있는 분야를 조사하고 그 자세한 적용 예를 밝힌다. 본 보고서의 2 절에서는 실시간 시스템과 그 시뮬레이션에 필요한 기본 사항을 제시하고 3 절에서는 실시간 시뮬레이션이 적용되는 예를 소개하며 4 절에서는 본 장의 결론을 제시한다.

2 절 실시간 시스템의 시뮬레이션

1 실시간 시스템의 필수 조건

실시간 시스템이란 컴퓨터의 계산 결과의 정확성 뿐만 아니라 그 계산의 결과가 나오는 시점이 매우 중요한 시스템을 말한다. 그 예로는 공장 제어 시스템, 항공기 제어 시스템, 고속 전철 시스템 등 산업 현장이나 우리 생활과 밀접한 관계가 있는 많은 시스템들이 있다.

이러한 시스템은 특히 시간 제약성 (timeliness), 신뢰성 (reliability), 그리고 가용성 (availability) 등의 측면이 매우 강조된다. 이같이 사회의 고도화에 중요한 실시간 시스템들은 그 규모의 거대함과 복잡성 때문에 사용좌의 요구 사항을 정확하게 규명하기가 어렵고 개발 단계에서 당면한 여러가지 의사 결정 사항들에 대해서도 미리 예측하기가 어렵다. 특히 시간적 행위에 대한 예측과 증명이 매우 중요한데 이를 해결하는 방법으로 시뮬레이션 기법이 채택되어 왔다.

실시간 시스템의 시뮬레이션을 수행함에 있어서는 다음의 두가지 필수 조건을 만족시켜야 한다.

- 실제 응용 환경에서 발생하는 여러 사건들의 상대적인 시간성이 정확하게 시뮬레이터에 의하여 모방되어야 한다. 따라서 사건들의 순서가 실제 환경에서나 시뮬레이터 상에서 같아야 함은 물론이고 실제 환경에서 일어난 두 사건 간의 시간 간격과 상응하는 시뮬레이터 상에서의 시간 간격과의 비율이 어는 두 사건을 고려하더라도 항상 일정하거나 미세한 변동폭을 보여야 한다.
- 시뮬레이션 속도는 실제 응용 시스템이나 환경이 진행하는 속도만큼 빨라야 한다.

따라서 이러한 실시간 시스템의 시뮬레이션은 응용 환경에 존재하는 여러 구성 요소간의 상호작용의 시간적인 면을 정확하게 보여줄 수 있는 것이다.

2. 거시적 및 미시적 시뮬레이션

복잡한 시스템을 모델링하고 시뮬레이션함에 있어 두가지 접근 방식을 취할 수 있다. 하나는 대규모 응용 시스템 전체를 적당한 수의 추계적 변수 (stochastic variables)로 구성된 거시적 (macroscopic) 모델로 나타내어 시뮬레이션 하는 것이다. 다른 하나는 일단 사용자의 요구 사항과 전체적인 시스템의 주요 기능 및 구조가 결정된 후 시스템의 상세 설계 과정에 있어 발생하는 여러 가지 결정 사항을 미시적(microscopic)으로 모델링하고 시뮬레이션하는 것이다. 즉 시스템의 분할, 분할되어 생성된서브 시스템들의 네트워크를 통한 연결 방법, 타스크의 분할 및 스케쥴링 등 설계 과정에서 결정되어야 할 많은 사항들이 실시간 성능에 직접적인 영향을 미치는데, 미시적 고정밀 시뮬레이션이 이와 같은 결정을 체계적으로 내리며, 결정되는 사항이 시스템 성능에 미치는 영향을 측정할 수 있도록 도와준다. 거시적 시뮬레이션과 미시적 시뮬레이션의 특징을 비교하면 다음과 같다.

구분	거시적 시뮬레이션	미시적 시뮬레이션
장점	시뮬레이터를 신속, 저렴하게 설치할 수 있다.	응용 시스템의 계획, 개발 및 운영 단계 기간중의 상세한 결정 사항에 대하여 정밀한 정보를 제공한다.
단점	시스템 개발 및 운영 단계 기간의 결정 사항에 대한 지원이 약하다.	시뮬레이터의 설치 비용이 크다.

거시적 시뮬레이션은 모델을 설정하고 시뮬레이터를 구현하는 것이 비교적 용이하고

비용이 적게들어 시스템 요구사항을 설정하기 위하여 급히 사용할 수 있는 시제품 도구 (rapid prototyping tool)로 적합하다고 볼 수 있다. 이러한 거시적 시뮬레이션은 시스템 개발 주기에 있어 초기 단계에 활용되며 교통 제어 시스템, 금융 관리 시스템, 에너지 유통 시스템, 정보 통신 시스템, 재해 관리 시스템의 설계 단계에서 커다란 역할을 할 수 있다.

미시적 시뮬레이션은 목표로 하는 응용 시스템의 세부적인 구성 및 동작들을 분명히 표현하는 것으로 다음의 성질을 갖는다. 즉 방대한 양의 각종 객체로 구성되며 각 객체의 동작 특히 시간적 요소를 상세히 표현하여야 한다. 또한 목표로 하는 응용 시스템의 계속적인 확장 및 개조에 따른 모델의 확장 및 개조가 용이하여야 한다. 따라서 미시적 시뮬레이터를 구축하는데는 다음과 같은 난점을 극복하여야 할 필요가 있다.

정확하고 이해하기 쉽고 확장 및 개조가 용이한 미시적 모델을 구축하고 대규모의 고정밀 모델을 병렬 처리 등의 대용량 고성능 컴퓨터 시스템에서 빠른 시간내에 수행되는 프로그램으로의 전환이 쉽게 되어야 한다. 이와 같이 구축된 고정밀 미시적 시뮬레이터는 새로 개발하는 대규모 시스템의 최적 설계 선정 또는 기존 실시간 시스템의 최적 운영 방식 선정에 공헌하고 불규칙적인 동작을 하도록 설계된 시스템의 시간별 성능의 정확한 표시에 사용될 수 있다. 최근들어 병렬처리 시스템 기술 및 객체 지향형 스프트웨어 기술 등이 실용화 단계에 접어들면서 고성능 고정밀 시뮬레이터 구축 활동이 활발하게 되었다.

3 절 실시간 시뮬레이션의 적용 예

1. 발전소 운전원 훈련용 시뮬레이터의 개발

본 절에서는 영광 3, 4 호와 고리 2 호기등의 원자력 발전소와 보령 3, 4 호기등의 화

력 발전소의 모든 계통들을 모델링하여 정상 및 비정상 운전상태 등을 시뮬레이션하기 위한 완전 복제형 시뮬레이터의 개발에 대하여 알아본다[2]. 완전 복제형 시뮬레이터란 완전형 시뮬레이터 중에서도 실제 대상 시스템과 외관 및 성능면에서 차이가 없는 시뮬레이터를 말한다.

가) 시뮬레이터의 구성

발전소 운전원 훈련용 시뮬레이터는 실제 주제어반과 똑같이 구현된 모의 제어반과 운전원에게 각종 훈련 상황을 부여하고 훈련 결과를 분석하는 강사 조작반, 그리고 컴퓨터실로 구성된다. 실제 발전소의 주제어반은 원자로나 보일러등 발전 계통과 연결되지만 모의 제어반의 모든 계장제어 장치는 시뮬레이터 컴퓨터로 연결되어 컴퓨터는 발전 계통 모델을 수행한 후 결과를 다시 모의 제어반으로 보낸다. 강사는 강사 조작반을 통하여 임의의 상황을 모의제어반에 나타나게 하고, 운전원은 주어진 상황에 대처해야 하는데 운전원의 대처 상황은 모두 컴퓨터에 기록되며 이 기록은 추후 평가자료로 쓰인다.

나) 시뮬레이터의 기능

시뮬레이터의 주요 기능은 강사로 하여금 정상 운전 및 비정상 운전을 비롯한 각종 설비의 성능 저하, 고장 등으로 인한 비상 사태등을 포함하는 임의의 상황을 부여하고 그 결과를 분석하게 하는 것이다. 이러한 역할을 수행하기 위한 시뮬레이터의 주요 기능은 다음과 같다.

- initial condition : 실시간 시뮬레이션 시작 조건
- backtrack : 어느 이전 시간으로 되돌아감
- fast/slow time : 실시간 대비 1/10 - 10 배의 속도

- malfunction : 발전소 부속 장비의 고장 및 성능 저하
- replay : 동일한 시뮬레이션 환경 재생
- override : 어느 순간의 환경의 재생
- snapshot : 어느 순간의 환경을 저장
- freeze : 시뮬레이션의 의도적 정지

다) 개발 과정

일반적으로 시뮬레이션의 개발은 모델 구축과 시뮬레이션 수행을 통한 시스템의 검증으로 나눌 수 있는데, 모델 구축은 자료 수집, 기초 설계 및 상세 설계를 거치며 검증은 독자 검증 (Non-Integrated Test) 및 통합연동검증 (Integrated Test)으로 나누어 차례로 진행된다.

자료 수집 대상은 발전소 구성 요소에 대한 도면, 운영 절차서, 운전 기록 등을 포함하며, 기초 설계 단계에서는 수집된 자료들을 바탕으로 1, 2 차 계통 및 컴퓨터 시스템을 분석하여 구성 요소별 시뮬레이션 범위 및 기능을 확정하고 시뮬레이션 Diagram 을 도출해 낸 뒤 Configuration Management System 이라 불리는 일종의 데이타베이스를 구축한다.

상세 설계 단계에서는 기초 설계에서 완성된 구성 요소 모델에 실제 데이타를 입력시켜 모델 및 해당 데이타에 대한 보다 세부적인 검증을 수행하면서 최종 데이타베이스를 구축한다. Coding 은 Autocode Generator 등을 활용함으로 상세 설계 과정의 일부로 간주된다.

라) 실시간 시뮬레이션의 구현 기술

(1) 컴퓨터 기술

발전소 시스템과 같은 복잡한 시뮬레이션에서는 그 작업의 다양함과 방대함으로 인하여 일반 프로그래밍보다는 Modular 프로그래밍, 데이타 abstraction, task 분리 등을 포함한 객체 지향형 환경이 여러면에서 유리하다. 특히 실시간 환경에서 운영되는 시뮬레이터에서는 해당 컴퓨터 계산의 정확한 결과를 정확한 시간에 도출해야 하는데 전체적인 컴퓨터 환경은 synchronized semi-independent task 들로 구성되며 각 작업의 동작은 Event-Driven Scheduling 에 의해서 조정된다.

운전원 훈련 및 면허 시험에 적용되는 발전소 시뮬레이터에 관한 기준인 ANSI/ANS-3.5 에 의해서 시뮬레이션의 주요 변수별 허용 오차 범위를 엄격히 제한하고 있다. 그 자세한 사항은 다음과 같다.

- 1% 이내 : 온도 - 평균, 고온, 저온
  출력 (MWe)
  노심 흐름 (feed flow)
  노심 냉각시스템 압력
  증기발생기 압력
- 2% 이내 : 증기발생기 유체 흐름 (feed flow)
  노심 냉각시스템 유체 흐름
  충전 흐름 (charging flow)
  증기 흐름
  터빈 1 단 압력
- 10% 이내 : 언급되지 않은 사항

이러한 기준을 충족시키기 위해서는 Time Based Scheduling 에서와 같은 간단한 interrupt

service routine 만으로는 부족하며 외부의 비동기 event 의 우선 순위에 따라 현재 수행 중인 작업을 pre-empty 시킬 수 있는 Priority Event Driven Scheduling 이 필수적이다.

(2) 하드웨어 기술

시뮬레이터의 하드웨어 부분은 모의제어반에 부착하는 각종 계기, 제어기 및 이를 컴퓨터로 연결하는, 대략 20,000 에 달하는 신호점을 처리하는 입출력 제어 시스템을 포함하여 각종 기기의 특성을 분석하여 이 것들을 시뮬레이션 목적에 맞게 수정, 설계할 수 있는 능력이 관건이다. 모의제어반의 기기들을 컴퓨터로 연결시키기 위해서 마이크로 프로세서가

내장된 지능형 입출력 제어기가 반드시 필요하다. 이러한 제어기는 모든 신호를 실시간으로 처리하고 신호검증 및 시뮬레이션 컴퓨터와의 인터페이스를 단순화시켜줌으로써 컴퓨터의 부담을 줄여주는데 신호의 변환, 즉 아날로그와 디지탈 신호간의 상호 변환 기능을 포함한다. 일반적으로 CPI, VMIC, DIGI-3 등의 제품이 통용되고 있으며 하드웨어 기술자들은 기기 및 소요 전력 등을 감안하여 기기의 신호들을 모듈별로 묶은 후 상세 배선 설계도까지 작성해야 한다. 대상 시스템 가운데 소프트웨어로 구현하기 매우 어렵거나 소프트웨어 구현 시 처리 시간등이 시뮬레이션의 실시간 범위를 벗어나는 경우는 주요 제어 부분에 실제 제어기를 사용하는 Stimulation 방법도 병행되고 있다.

2. 교육훈련용 Nuclear Simulator 개발

원자력 발전소 운전 요원과 규제 기관의 감독 요원은 발전소 운전에 관한 기본 지식와 반 원리 뿐만 아니라 발전소 계통의 독특성과 과도 현상에 대한 정성적 정량적인 이해를 할 수 있도록 교육, 훈련되어야 한다. 그런데 이러한 훈련이 정상 가동을 최대의 목적으로 하는 실제 발전소에서 행해진다는 것은 시간적, 경제적으로 매우 어려운 일이며, 따라서 시뮬레이터를 이용하는 것이 효과적이다. 본 절에서 소개하는 Nuclear

Simulator 는 실제 발전소의 제어반과 같은 기능을 가진 모의 장치로서 실제 발전소에서 경험하기 어려운 사고에 대한 훈련과 각 단계별로 다양한 운전 상태 의 경험 또는 운전이 가능하며, 지나갔던 운전 상황을 반복하여 재현하거나, 현재 상태를 정지 및 천천히 동작시켜 계통의 변화를 이해 또는 분석할 때 상당한 도움을 준다[3]. 또한 발전소 운영 보수의 감독자, 보수요원, 규제 기관의 운영감독요원 등의 기본 교육에도 필수적이며, 나아가서는 발전소 핵증기 공급계통 (nuclear steam supply system) 및 부품 설계, setpoint 연구, 그리고 안전성 연구에도 활용할 수 있다. 이를 위하여 Nuclear Simulator 는 원자력 발전소의 가동, 예비 점검, 예열, hot start-up, 운전 정지, 출력 조정, 정상 또는 특정한 사고시의 운전 조건을 포함한 각종 운전 모드를 모의화하여, 그 결과를 운전원 판넬의 각종 계기와 고해상도 칼라 화면에 나타낸다.

시뮬레이터는 그 성능 및 규모에 따라서 full scope simulator 와 compact simulator 로 구분된다. Full scope simulator 는 특정 발전소의 운전 요원을 훈련시키는 것이 주목적으로 해당 발전소의 주제어실의 판넬과 각종 계기들을 정밀하게 복사하고, 모든 운전 상태가 동일한 성능을 갖도록 하여 운전요원이 시뮬레이터의 반응과 실제 발전소 주제어실과의 차이를 발견하지 못하도록 설계하는 것이다. 그리고 compact simulator 는 표준형 발전소를 간단하게 축소시켜 모델화하거나, 발전소의 일부분을 모델화하여 고도의 밀집형으로 설계된 운전원 판넬을 통하여 훈련원에게 계통의 일부 또는 전체에 대한 특정한 의사 결정의 효과를 관찰하게 함으로써 계통 및 동력학에 관한 포괄적인 이해를 가능하게 하여 발전소 운전에 관한 기본 지식과 일반적인 원리를 습득하게 한다. 이 연구의 Nuclear Simulator 는 compact simulator 로 구현되었다.

가. 원자력 발전소 시뮬레이터의 요구 조건

원자력 발전소 시뮬레이터는 주로 운전 요원의 초기 및 재교육 훈련을 제공하기 위한

훈련 도구이다. 따라서 시뮬레이터는 운전원의 적절한 조치, 부적절한 조치, 자동 조정 또는 발전소 고유의 운전 특성을 받아서 나오는 시뮬레이터의 반응과 모발전소의 반응과의 차이를 느끼지 못하도록 설계되어야 한다. 이러한 설계 조건을 만족시키기 위한 상세한 요구 사항은 다음과 같다.

(1) 정상 운전

시뮬레이터는 연속적 실시간으로 발전소의 운전 상태를 시뮬레이션할 수 있는 능력이 있어야 한다. 이를 위해서 고려해야 하는 사항들은 다음과 같다.

- 발전소 기동 - 저온에서 고온 대기까지
- 고온 대기에서 정격 출력까지
- 터빈 기동과 발전기 동기화
- 원자로 정지와 경격 출력까지의 재기동
- 고온 대기시의 운전
- 부하 변동
- 정격 이하의 원자로 냉각수 순환에서 기동과 정지
- 정격 출력에서 발전소 저온 정지
- 발전소 heat balance, 원자로 정지 margin 결정 및 반응도 측정과 같은 core 성능 시험
- 안전 관련 계통의 시험

(2) Malfunction

시뮬레이터는 실시간으로 비정상 또는 비상 사고에 대한 시뮬레이션 능력이 있어야 한다.시뮬레이션할 malfunction 의 형태와 갯수는 운전원 교육 훈련과정, 성능 및 설계

기준, 체계적인 프로세스등이 고려되어야 한다. 그리고 이 malfunction 에는 true/false 만을 나타내는 것과 0 - 100 % 사이의 severity 를 갖는 두 종류가 있다.

(3) 시뮬레이터 환경 조건

시뮬레이터는 컨트롤러, 각종 계기, 스위치, 기록계, 발전소 공정제어 컴퓨터, 경보반 및 기타 man-machine interface 를 위한 판넬이 있어야 하며, 이는 모발전소와 크기, 모양, 색 및 구성에 있어서 기능적 또는 시각적으로 일치해야 한다.

(4) 시뮬레이터 훈련 조건

* 초기조건 : 최소한 20 개의 초기조건을 저장할 수 있어야 한다.

* Malfunction : 운전 경험에 의해서 예측되는 malfunction 을 자유롭게 삽입하고 제거할 수 있는 능력이 있어야 한다.

* halt, fast time, slow time, backtrack, snapshot 등의 기능을 갖추고 있어야 한다.

나. Compact Nuclear Simulator 의 구현

CNS (Compact Nuclear Simulator)는 원자력 발전소 기초 교육, 계통 설계 및 안전성 연구에 필요한 주요 계통 또는 일부만을 모형화하여 제작한 시뮬레이션 장치이다.

(1) CNS 의 하드웨어

CNS 의 하드웨어는 주제어실의 제어반을 축소한 operator console, 실제 발전소와 유사한 상황을 모형화하는데 필요한 모든 데이타를 발생시키는 컴퓨터 조직, 컴퓨터와 console 사이의 입출력 정보를 전달하는 interface 카드, 3 개의 컬러 CRT 에 표시되는 full graphic display 계통, 훈련원에게 각종 명령을 내리고 훈련원의 훈련 내용을 감시할 수 있는 강사 조작반으로 구성된다.

(2) CNS 의 소프트웨어

CNS 의 소프트웨어는 발전소 각 계통의 동작을 표시한 수학적 모델링 프로그램, 강사 조작반에서의 각종 명령을 처리해주는 supervisor 프로그램, 공용 데이타베이스를 관리하는 공통 데이타 파일 (CDF) 프로그램, 훈련 시작 전 판넬상의 램프, 기록계, 스위치 등 하드웨어의 동작 여부를 점검할 수 있는 프로그램으로 구성된다.

이들 소프트웨어와 하드웨어 구성 요소들간의 상관 관계는 그림 1 에 잘 나타나 있다.

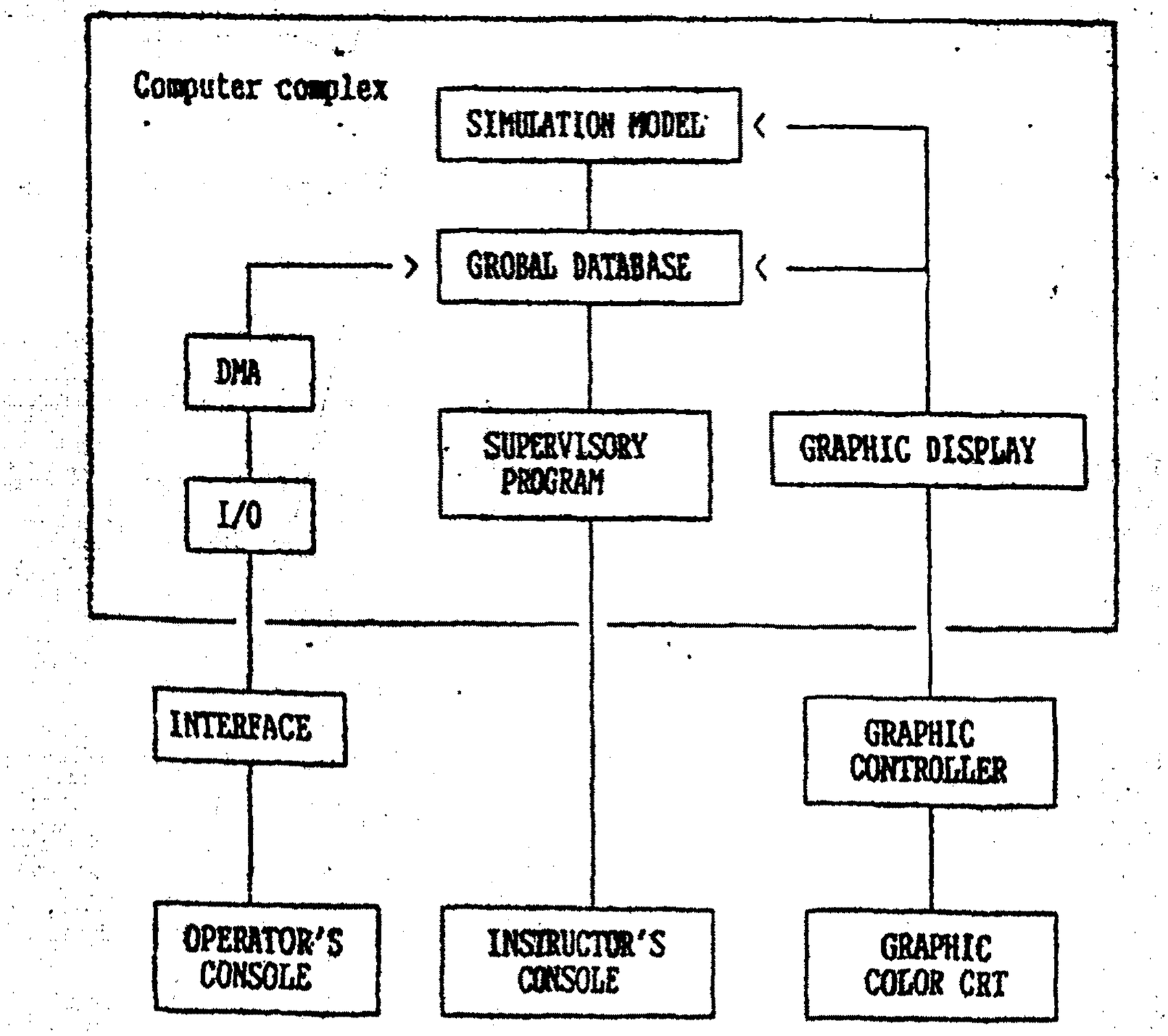


그림 1 : KAERI-CNS 의 블록 다이어그램

(3) CNS 의 그래픽 시스템

CNS 를 위한 그래픽 시스템은 3 개의 동일한 디스플레이 채널로 구성되어 있고, 각 채널은 1 Mbyte 의 내부 메모리를 가진 CPU 카드, 각 1 개씩의 디스크 제어 카드, 그래픽 디스플레이 제어 카드, 통신 카드, 그리고 19" 컬러 모니터로 이루어져 있고, 또

한 3 채널이 공유하는 각 1 개씩의 floppy disk drive 와 hard disk drive, hard copy unit 으로 구성되어 있다.

3 스테인레스 스틸 생산 공장의 시뮬레이션

가. 상황 분석

스테인레스 스틸 공장의 생산 공정은 POSCO 의 처녀 조업 설비로서 각 공정별 기술 분석은 잘 되어있으나 하나의 종합 시스템으로서 운영될 때의 기술 및 관리상의 문제점에 대한 분석 및 연구는 부족한 상황이다. 그 이유는 스테인레스 스틸 공장이 과거의 경험이나 기술 축적이 없는 상태에서 처음으로 건설되고 있기 때문이다[7].

(1) 스테인레스 스틸 생산 공정

POSCO 의 스테인레스 스틸 생산 공장에서는 Austenite, Ferrite, Martensite 등 3 가지 강종을 코일과 선재로 생산하고 있다. 코일 생산 공정은 크게 나누어 1) 전기로, 2) 정련로, 3) 연주, 4) 냉각 공정, 5) 연마 공정, 6) 열연, 7) 소둔산세 등으로 되어 있으며, 선재 생산 공정은 1) 전기로, 2) 정련로, 3) 연주, 4) 연마, 5) 강편, 6) 선재 등으로 되어 있다. 이 공정은 그림 2 에 잘 나타나있다. 전기로에서는 원료로부터 용탕을 생산하고 이 용탕은 Ladle 에 의해 정련로로 옮겨진다. 용탕은 여기서 정련된 후 옹강이 되어 Ar-bubbling 을 거친 후 Ladle 에 의해 연주 공정으로 넘겨진다. 연주 공정에서는 코일 생산을 위하여 Slab 을, 선재 생산을 위하여 Bloom 을 만들어 낸다.

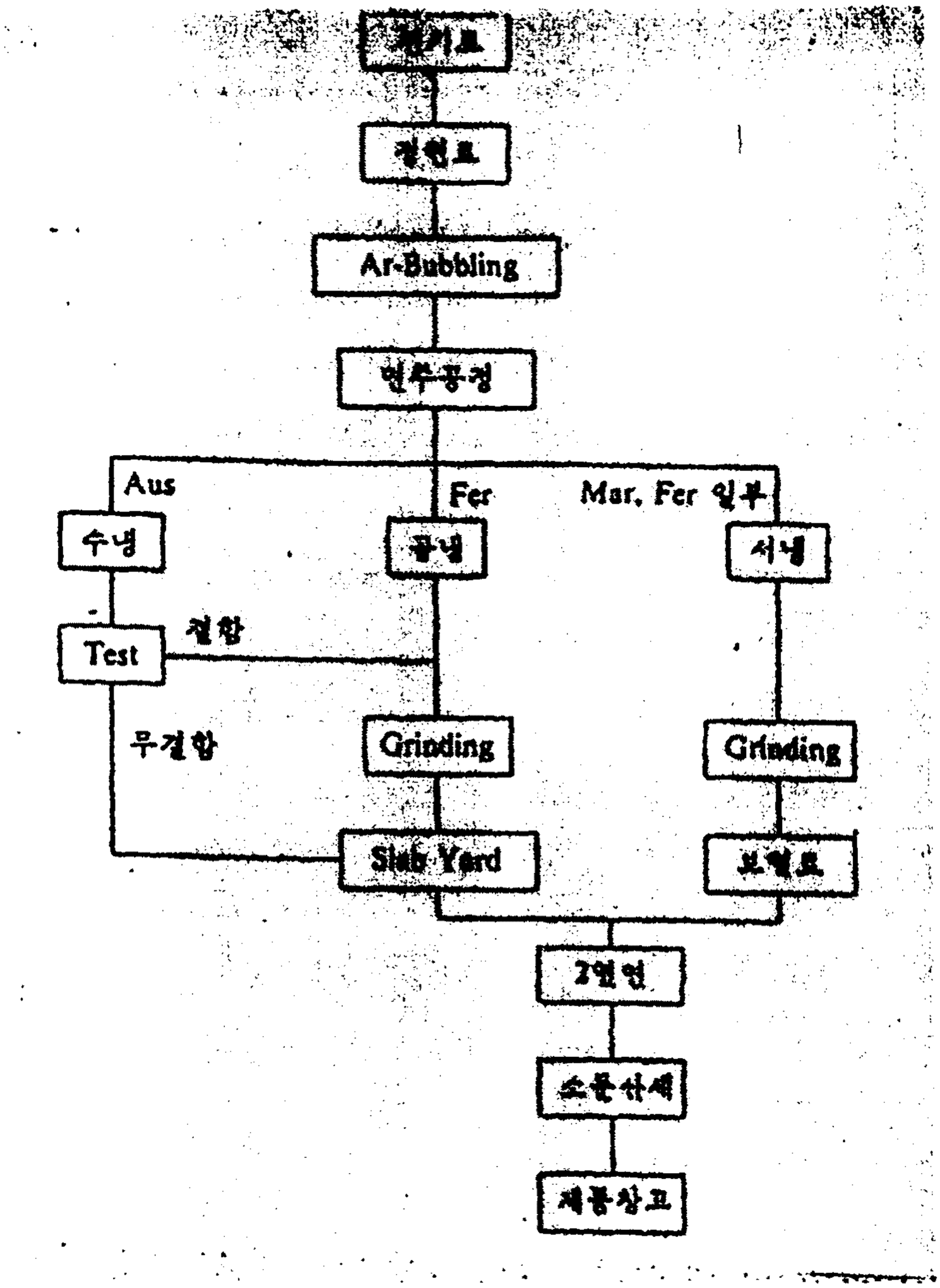


그림 2 : 스테인레스 스틸 생산 공정

(2) 시뮬레이션 대상 공정

스테인레스 스틸 생산 공장의 전 공정에 대하여 공정을 분석한 결과 선재의 생산 공정과 코일의 생산 공정의 후공정인 열연 및 소둔산세 공정에서는 별다른 물류 흐름상의 문제점이 예상되지 않았다. 따라서 시뮬레이션의 대상으로는 코일 생산 공정중에서 문제점이 예상되는 냉각 공정과 연마 공정을 선정하였고 상세히 공정 분석을 하였다. Austenite 는 연주 공정 후에 수냉에 의해 냉각을 한후 품질 검사를 위하여 12 개의 Slab lot 중에서 3 개의 샘플 표면에 이상이 없으면, 3 개의 샘플 Slab 과 lot 중 나머지 9 개의 Slab 은 더이상 연마하지 않고 Slab 야드로 옮겨서 열연 공정을 기다린다. 표면에 이상이 있을 경우에는 불량 표면을 제거하기 위하여 3 개의 샘플 Slab 의 남은 표면을 연마하고 나머지 9 개의 Slab 도 전부 전표면을 연마한 후 Slab 야드로 보내진다. Ferite 는 공기중에 서서히 냉각을 시킨 후 전량 연마를 한 다음 Slab 야드로 옮겨진다. Martensite 와 Ferite 중 일부 (20 %)는 서냉재로서 서냉로에서 천천히 식힌 후 전량 연마를 한 다음 보열로에서 그 다음 공정인 열연 공정을 기다린다.

강종 교체는 Ferite, Martensite, Austenite 순이며, 매주 이 교체 패턴을 반복한다. 연주 공정에서는 2 연 연주를 원칙으로 한다. 스테인레스 스틸의 주당 생산량은 98 Charges 를 예상하고 있으며, 그 구성비는 1) 국제 수요 패턴을 따를 경우, Austenite 가 70 charges, Ferite 가 20 charges, Martensite 가 8 charges 이고, 2) 국내 수요 패턴을 따를 경우, Austenite 가 70 charges, Ferite 가 11 charges, Martensite 가 17 charges 이다.

나. 시뮬레이션 모델의 개발

이 연구에서는 스테인레스 스틸 생산 공정을 SLAM 을 사용하여 시뮬레이션 모델로 개발하였다. 본 시뮬레이션에서는 실제 스테인레스 스틸 생산 작업이 매일 24 시간, 주 단위로 진행됨을 고려하여 각 대안별로 10 주간씩 10 회 반복하여 시뮬레이션을 수행하

였다.

시뮬레이션 모델에서 사용된 resource 는 전기로 1 개, 정련로 1 개, 연주기 1 개, 연마기 2 대,서냉 및 보온로 6 개 등이다. 수냉 및 공냉을 위한 설비에는 제한이 없으며, Slab 야드에 면적도 충분하다. 그리고 운반 수단인 ladel 과 crane 의 수도 충분하다. 전기로 및 정련로에서의 생산 기본 단위는 charge 이고, 한 charge 의 평균 무게는 90t 이다. 한 charge 는 연주 공정후 6 개의 Slab 으로 나누어지며, 각 Slab 의 평균 무게는 15t 이다. 용탕을 계획량인 주단 98 charge 를 생산한 후에 생산의 여력이 있으면 그 주가 끝날 때까지 Austenite 를 생산하는 것을 가정하였다.

각 공정에서 소요되는 시간의 분포와 parameter 값은 다음과 같다.(시간단위 : 분)

- 전기로 : 정규 분포, 평균 100.0, 표준 편차 6.0
- 정련로 : 정규 분포, 평균 85.0, 표준 편차 3.0
- Ar-bubbling : 30.0
- 연주 공정 : 정규 분포, 평균 60.0, 표준 편차 1.5
- 서냉 : 3000.0
- 수냉 : 30.0
- 공냉 : 2880.0
- 시험 연마 : Slab 당 56.0
- 시험편 잔여 연마 : Slab 당 64.0
- 전표면 연마 : Slab 당 120.0

또 각 공정간의 이동에 소요되는 시간은 다음과 같다.(시간단위 : 분)

- 전기로 ---> 정련로 : 20.0

- 정련로 ---> Ar-bubbling : 20.0
- Ar-bubbling ---> 연주 공정 :20.0
- 연주 공정 ---> 냉각 공정 : 10.0

다. 시뮬레이션 결과 분석

스테인레스 스틸 생산 공정에서 코일의 생산시의 공정 분석을 위하여 아래와 같은 여러 대안에 대하여 시뮬레이션을 수행하였다.

대안	생산 패턴	시험연마할 때 Slab 불량으로 불합격할 확률
대안 1	국제 수요 패턴	5 %
대안 2	국제 수요 패턴	10 %
대안 3	국제 수요 패턴	15 %
대안 4	국내 수요 패턴	5 %
대안 5	국내 수요 패턴	10 %
대안 6	국내 수요 패턴	15 %

위의 여섯 가지의 대안에 대하여 시뮬레이션을 수행한 결과 생산 공정의 다른 부문에서는별다른 문제점이 발견되지 않았으나, 모든 대안에 대하여 연마기와 서냉로의 서비스를 기다리는 대기 행렬은 시간이 지남에 따라 계속 늘어났다. 이는 연마기의 대수와 서냉로의 용량이 현재의 계획된 생산량을 감당하기에는 절대적으로 부족함을 나타낸다

따라서 2 차 시뮬레이션에서는 생산량을 감당하기 위해 필요한 각각의 설비 대수를 추정하기 위하여 각 설비의 용량을 충분히 크게 한 다음 1 차 시뮬레이션에서와 같이 6 가지 대안에 대하여 시뮬레이션을 수행하였다. 그 결과 대안 1,4 에서는 연마기위 평균 이용 대수가 2.8 대, 대안 2,5 에서는 3.0 대, 대안 3,6 에서는 3,3 대였다. 또, 필요한 서냉로의 용량은 대안 1,2,3 에서 12 charge, 그리고 대안 4,5,6 에서는 19 charge 였다.

따라서 3 차 시뮬레이션에서는 연마기의 대수를 현재의 2 대에서 3 대로 증가시키고, 서냉로의 용량을 대안 1,2,3 에서는 12 charge, 대안 4,5,6 에서는 19 charge 로 한 다음 다시 시뮬레이션을 수행한 결과, 연마기의 평균 이용 대수는 대안 1 에서 2.78, 대안 2 에서 2.95, 대안 3 에서 3.00, 대안 4 에서 2.80, 대안 5 에서 2.98, 대안 6 에서 3.00 이었으며, 서냉로의 용량은 각 대안에서 충분한 것으로 나타났다. 연마작업에 대한 통계치를 매주마다 조사한 결과 대안 1,2,4,5 에서는 연마기를 기다리는 대기 행렬이 안정 상태에 있었다. 따라서 대안 1,4 와 2,5 에서의 Slab 표면불량률이 각각 5%와 10%이므로 불량률이 10%이하이면, 주어진 스테인레스 스틸 생산량을 생산하기 위해서는 3 대의 연마기가 필요하다고 할 수 있다. 그러나 대안 3 과 6 에서는 연마기를 기다리는 대기 행렬이 시간이 지남에 따라 계속 증가하였다. 이는 대안 3 과 6 에서는 연마기의 대수가 3 대로서는 부족하다는 것을 말한다. 즉 Slab 표면불량률이 10%이상이면 연마기 1 대가 추가로 필요하며, 따라서 필요한 연마기 대수는 4 대이다. 연마기를 4 대로 하여 추가로 시뮬레이션을 수행한 결과 주어진 생산량을 충분히 감당할 수 있었다.

정상 조업시 생산량은 현재의 조업 조건하에서 평균 100.5 charge 까지 가능하다. Slab 야드는 적어도 95 charge 는 수용할 수 있어야 하겠다.

4. 압연 공정의 시뮬레이션을 위한 소프트웨어 POSROL 의 개발

가. 현황 분석

압연 공정에 있어서 품질과 실수율에 영향을 미치는 압연 변수들은 다양하다. 압연 패턴, Slab 의 초기 온도, 롤의 온도, 접촉면에서의 마찰력, 냉각율, 재료의 성질 등의 압연 변수들이 제품의 내부의 온도, 응력, 변형도 분포, 롤의 변형, 롤 토오크, 압하력 그리고 압연된 제품의 형상을 결정한다. 따라서 바람직한 품질의 제품을 생산하기 위해서는 이러한 압연 변수들의 최적 조합을 찾아 적용하여야 한다.

그런데 압연 변수들과 제품의 품질은 상호 밀접히 연관되어 있으므로 하나의 변수를 조절하여 성공적으로 제품의 품질 및 형상을 제어하기는 어렵다. 또한 압연 변수의 조합으 바꾸어 가면서 원하는 결과가 나올 때까지 현장에서 시행 착오적인 실험을 수행하는 것은 조업관계상 불가능하다. 따라서 압연 공정을 시뮬레이션할 수 있는 기술이 절실히 요구되고 있다.

나. POSROL 의 구성

이 연구에서 개발된 압연을 비롯한 소성 가공 공정의 시뮬레이션인 POSROL 은 유한요소 해석 기술에 바탕을 두고 있으며, 종래의 유한요소 수식화 방법과는 다른 새로운 수식화 기법이 적용되었다[8]. 이 소프트웨어는 기능별로 모듈화되어 있는데, 그 구성은 그림 3 과 같다. 즉, 3 차원 시뮬레이션 모듈 POSROL3D, 2 차원 시뮬레이션 모듈 POSROL2D, 그리고 그래픽 출력 모듈 POST & DISPLAY II 로 구성되어 있다. 그리고 그래픽 모듈을 제외한 모든 모듈은 유동 시뮬레이션 프로그램, 온도 (열) 시뮬레이션 프로그램, 그리고 유동과 온도의 혼합 시뮬레이션 프로그램으로 구성되어 있다.

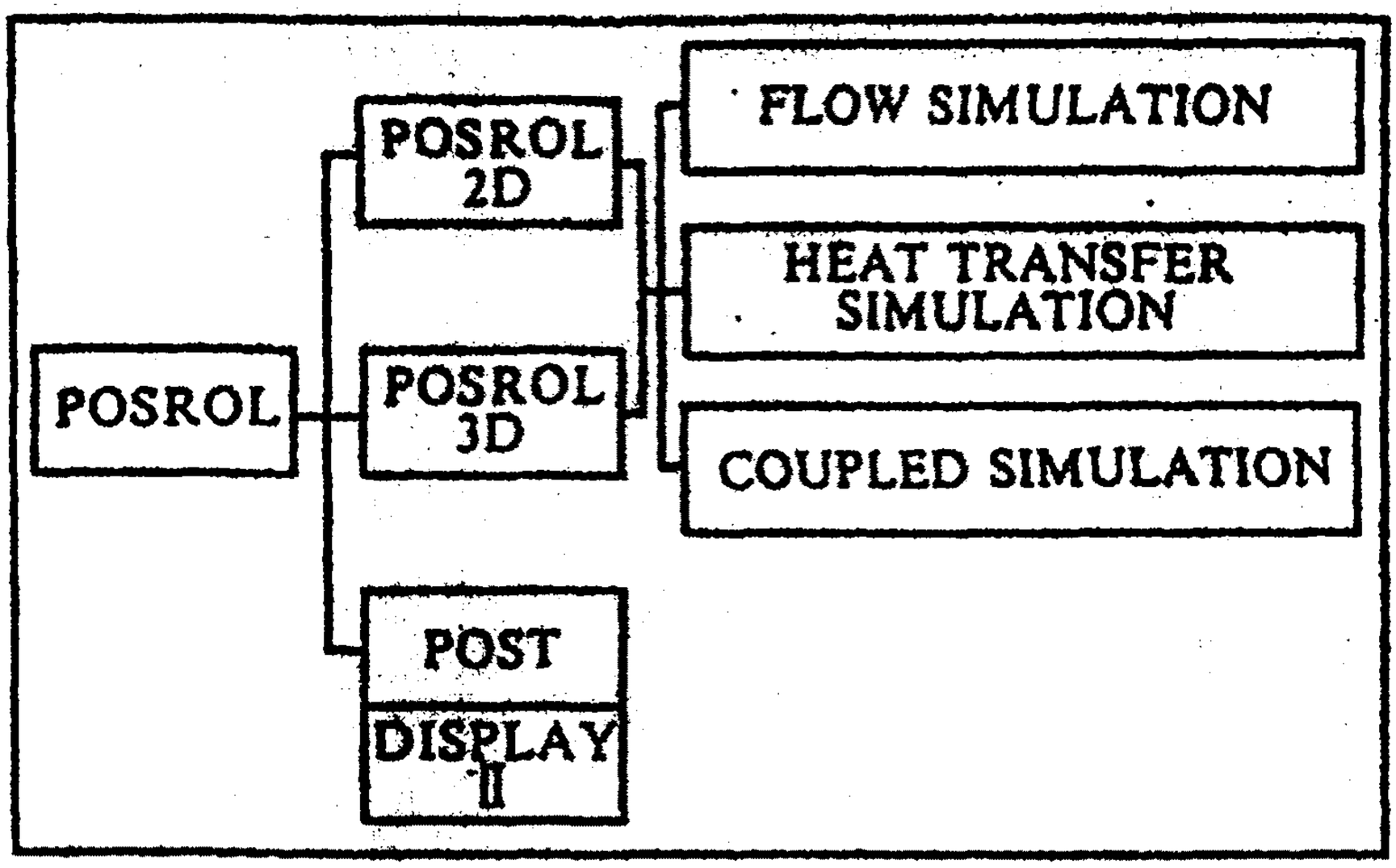


그림 3 : POSROL 의 구성

POSROL2D 와 POSROL3D 의 유동 해석 부분은 유동 문제의 해를 수치적으로 계산하는 프로그램, 입력 자료를 읽어들이는 프로그램, 계산에 필요한 자료와 계산 결과들을 정리하여 디스크에 저장하거나 디스크에서 읽어들이는 프로그램, 그리고 계산된 결과들을 프린트하는 프로그램 등으로 구성되어 있다. 계산되는 계산 결과들은 변형된 형상, 속도장, 응력 및 정수압, 변형도, 변형도 속도 등의 분포, 그리고 Roll 표면에 걸리는 압력과 접선 응력의 분포등이다.

POSROL3D 는 3 차원 유동을 해석하는 기능을 가지고 있다. 후판이나 분괴압연에서 Slab 와 인곳의 변형은 3 차워적이므로 정확한 공정 해석을 위하여 3 차원 시뮬레이션이 요구된다. 그러나 위의 공정에 있어서 대략적인 변형 경향만 살펴볼 경우, 2 차원 해석 목적으로 개발된 POSROL2D 를 사용할 수도 있다. 실제 열연이나 냉연 공정처럼 얇은

판을 압연하는 경우 그 변형은 거의 2 차원적이므로 POSROL2D 를 사용함이 계산 속도 측면에서 매우 효과적이다.

재료가 유동함에 따라 형상이 변화하며 따라서 유효 변형도와 경계 조건도 바뀌게 된다. POSROL3D 와 POSROL2D 는 재료를 구한 속도장에 기초하여 조금씩 이동시켜 나아가 그 때에 해당하는 해를 구하는 과정을 재료과 압연 공정을 완전히 통과할 때까지 반복하는 방법을 사용하고 있다. 따라서 시뮬레이션을 통해 압연의 어느 시점에서든지 그 시점의 압연 상태를 예측할 수 있다.

Process No.	Material Properties	Roll		Thickness of Slab		Coef. of Friction
		Dia. mm	Speed mm/sec	Initial mm	Final mm	
1	Y=50.3 b=0.05 n=0.25 m=0.0	158.75	1.0	6.275	5.38	0.1
2	Y=88.7 b=0.05 n=0.25 m=0.0	120.0	1.0	2.0	1.0	0.14 0.25
3' a	Yo=1.0 n=0.0 m=0.0	90.0	1.0	1.0	a 0.8	0.1 (for a, b, c)
3' b					b 0.6	0.06 (for c)
3' c					c 0.4	0.04—0.6 (for a, b, c)
4	Yo=1.0 n=0.0 m=0.13	900.0	1256.64	2.0	1.2	0.25

그림 4 : 압연 공정의 변수들의 값

POSROL2D 는 속도와 온도, 변형도, 변형도속도 등 모든 변수들이 공간의 고정된 점

들에서 볼 때에 변하지 않는 경우, 재료의 유동을 처음부터 추적하지않고도 이 변수들을 예측하는 기능도 가지고 있다. 재료가 일정하게 움직이는 공정인 열연이나 냉연 공정이 이 경우에 해당된다고 볼 수 있다.

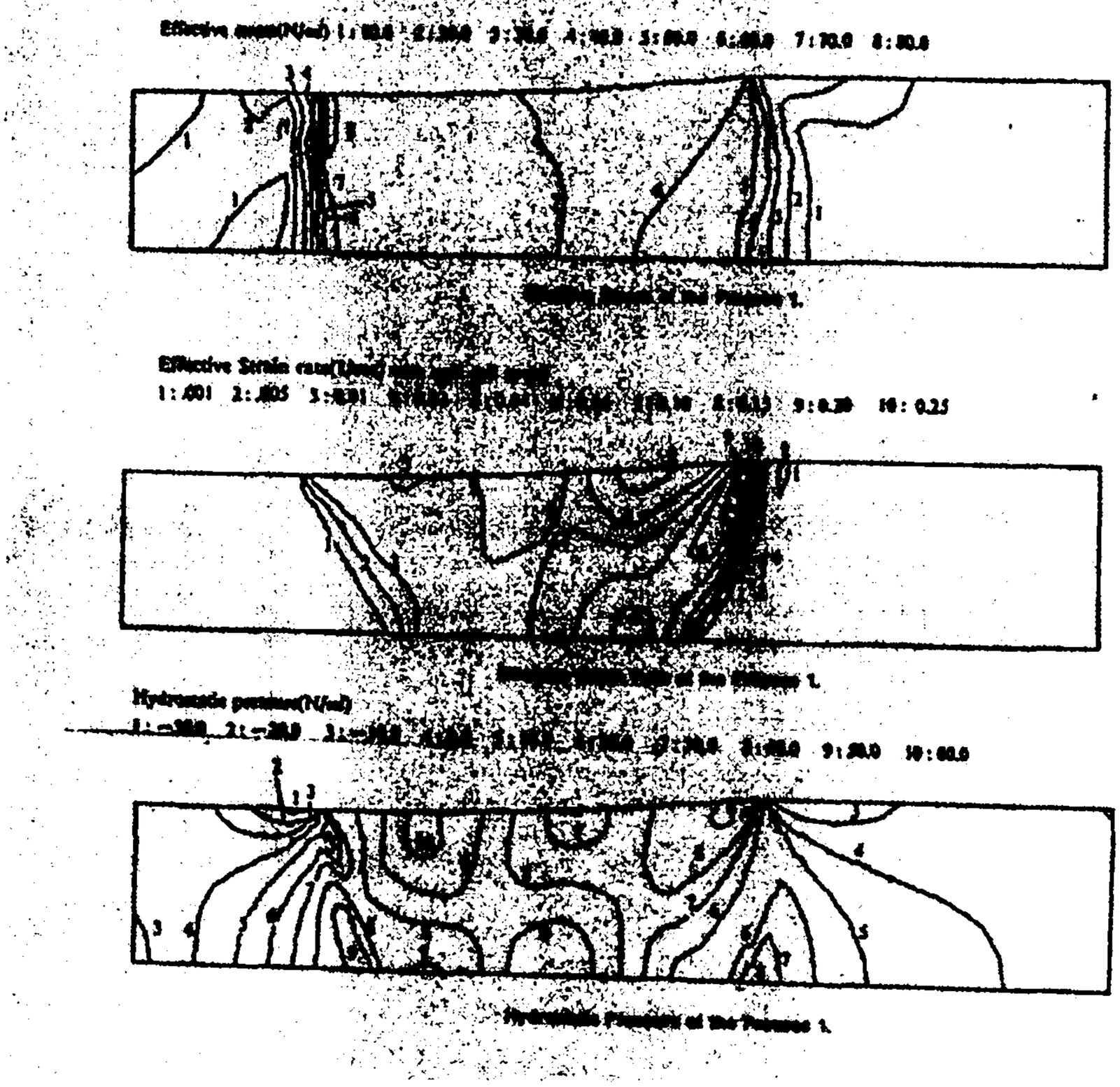
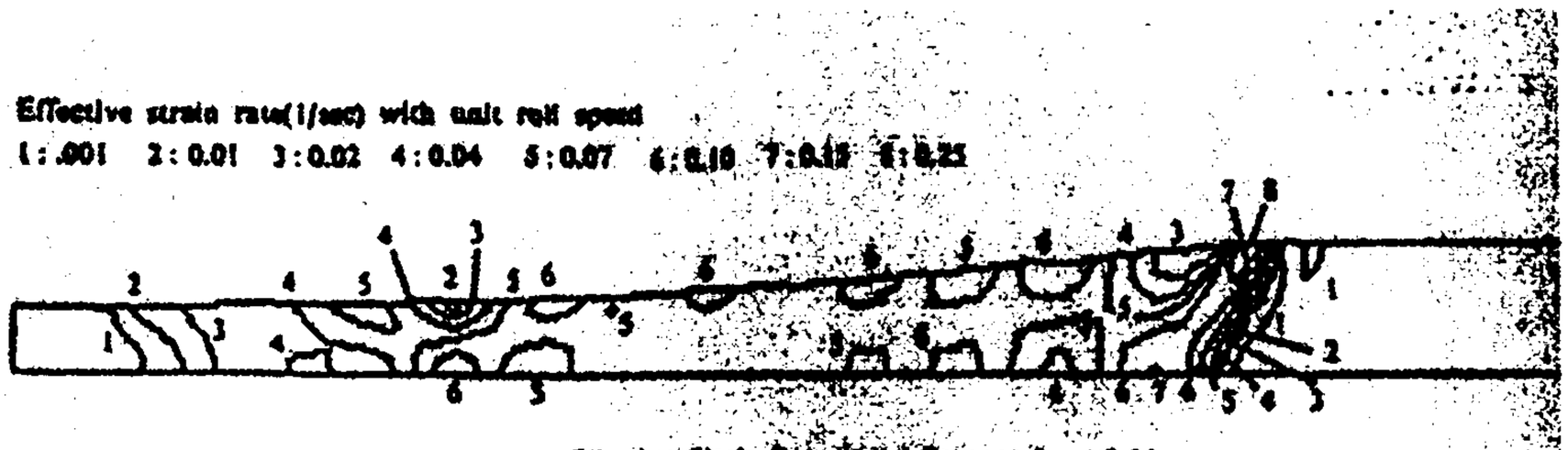


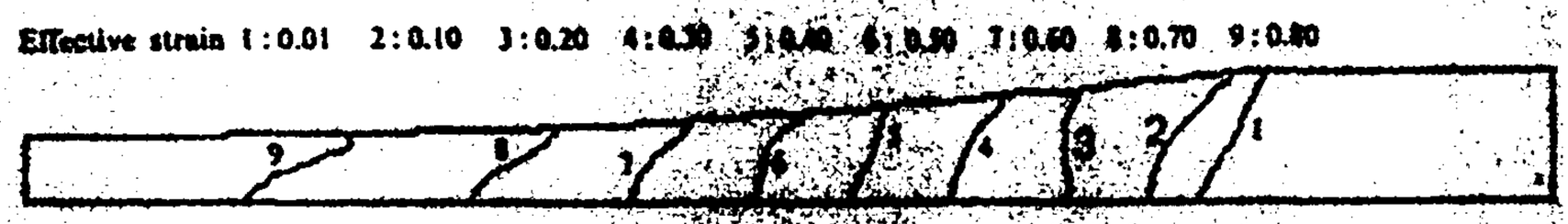
그림 5 : 공정 1의 시뮬레이션 결과

압연된 재료의 상태는 롤과 재료 사이에 작용하는 마찰력의 크기에 영향을 받는다

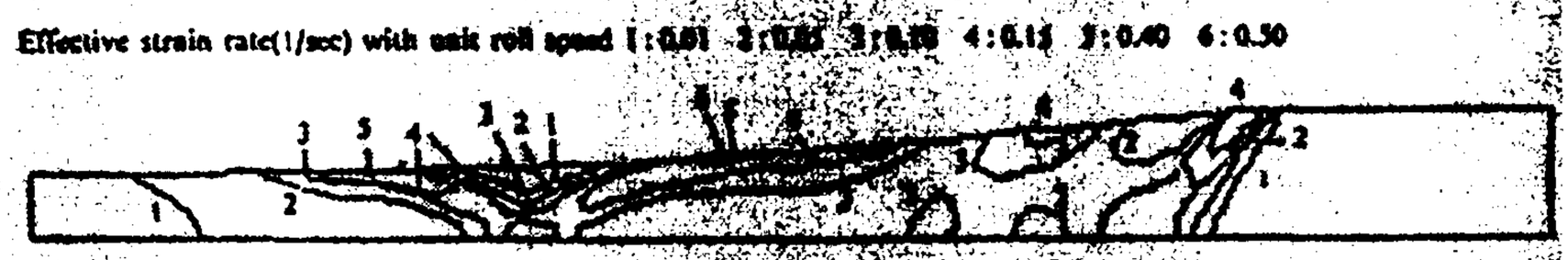
재료의 유동시의마찰력을 결정하는 상수인 Coulomb 마찰 계수는 코드의 입력 자료이다. 해석할려고 하는 공정의 마찰 계수가 알려져 있지 않은 경우에 실험에 의해서 구하거나 또는 시뮬레이션을 실시하여 예측된 형상을 실제 압연된 형상과 비교하여 그 공정의 마찰 계수를 찾을 수 있다. 후판 압연에서는 냉연에 비하여 일반적으로 마찰 계수가 크며, 그 값은 0.3 에서 0.4 사이에 있다.



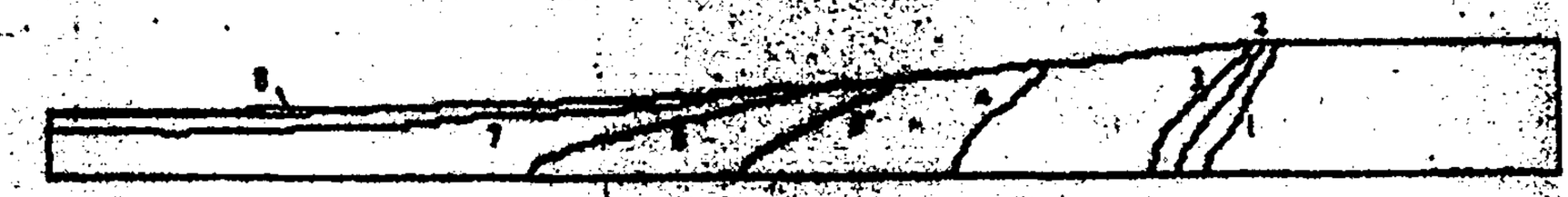
Effective Strain Rate of the Process 2, μ=0.14



Effective Strain of the Process 2, μ=0.14



Effective strain rate of the Process 2, μ=0.25



Effective Strain of the Process 2, μ=0.25

그림 6 : 공정 2의 시뮬레이션 결과

개발된 코드들은 임의의 압연 패턴과 임의의 재질에 대하여 시뮬레이션을 수행할 수 있도록 되어 있다. 압연 패턴이 달라지면, 압하율, 롤의 크기 및 위치 등이 변화하는데 이 때에 적합한 경계 조건을 입력시킬 수 있으며 유동 응력과 온도, 변형도, 변형도속도의 관계식은 서브루틴을 만들어 본 코드와 함께 사용할 수 있다.

다. POSROL을 이용한 압연 공정의 유동 시뮬레이션

(1) 압연 공정의 2차원적 유동 해석

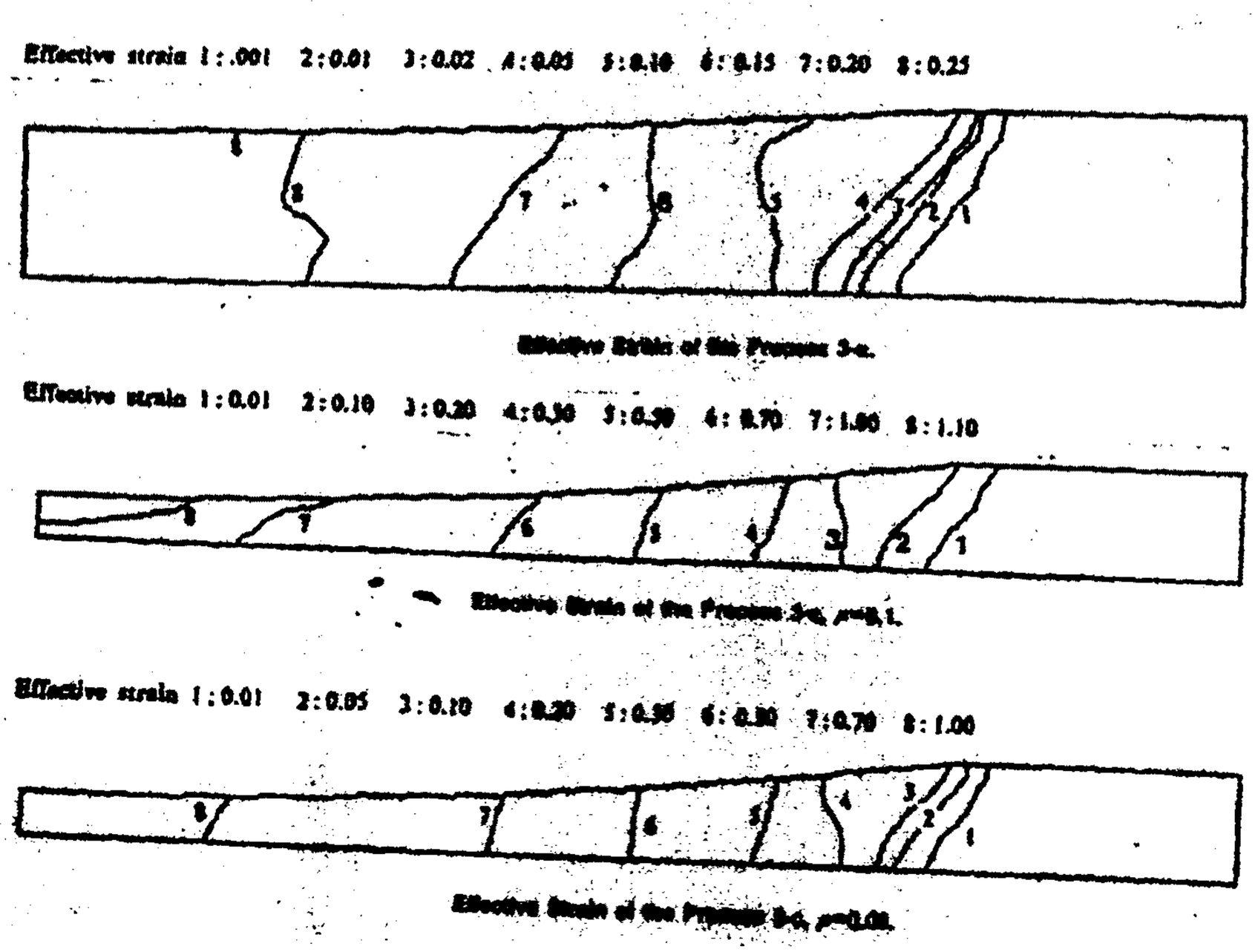


그림 7 : 공정 3의 시뮬레이션 결과

POSROL2D 를 이용해서 시뮬레이션한 공정에 대한 공정 변수로는 각 해석 모델에 대해서 공작물 (Workpiece)의 재질, 롤의 직경, 초기 및 최종 두께, 그리고 마찰 조건들이 필요하다. 각 공정에 대한 이러한 변수들의 값들은 그림 4 에 나타나 있다.

(가) 공정 1 의 시뮬레이션

공정 1 의 시뮬레이션 결과는 그림 5 에 나타나 있다.

(나) 공정 2 의 시뮬레이션

이 공정은 박판 고압축비의 한 예이다. 시뮬레이션의 결과는 Gou-Ji Li 와 Kobayashi 에 의한 해석 결과와 비교되었으며, Lueg 과 Siebel 의 실험 결과와 비교되었다. POSROL2D 에서는 새로 개발된 Coulomb 마찰 모델 (Friction Model)을 사용하여 롤과 공작물의 접촉면에서의 마찰력을 다루고 있다. 그림 6 에서 알 수 있듯이 마찰 계수의 증가는 유효 변형도속도와 유효 변형도를 매우 민감하게 변화시킴을 알 수 있다. 마찰력의 증가에 따라 유효변형도의 분포가 매우 불균일해지는 것은 롤과 재료의 접촉면 주위에서의 전단 변형이 마찰의 증가에 따라 급격히 증가한 까닭으로 판단된다.

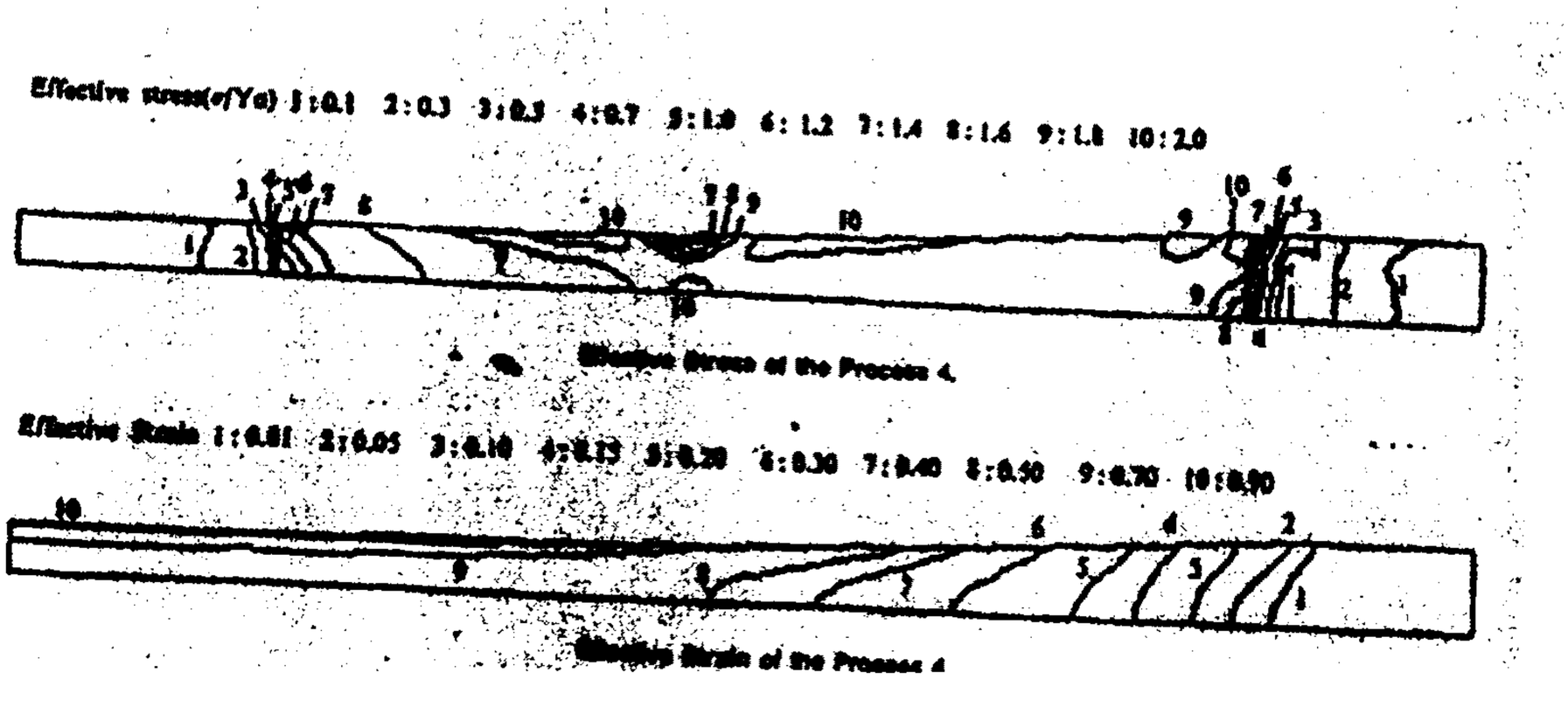


그림 8 : 공정 4의 시뮬레이션 결과

(다) 공정 3의 시뮬레이션

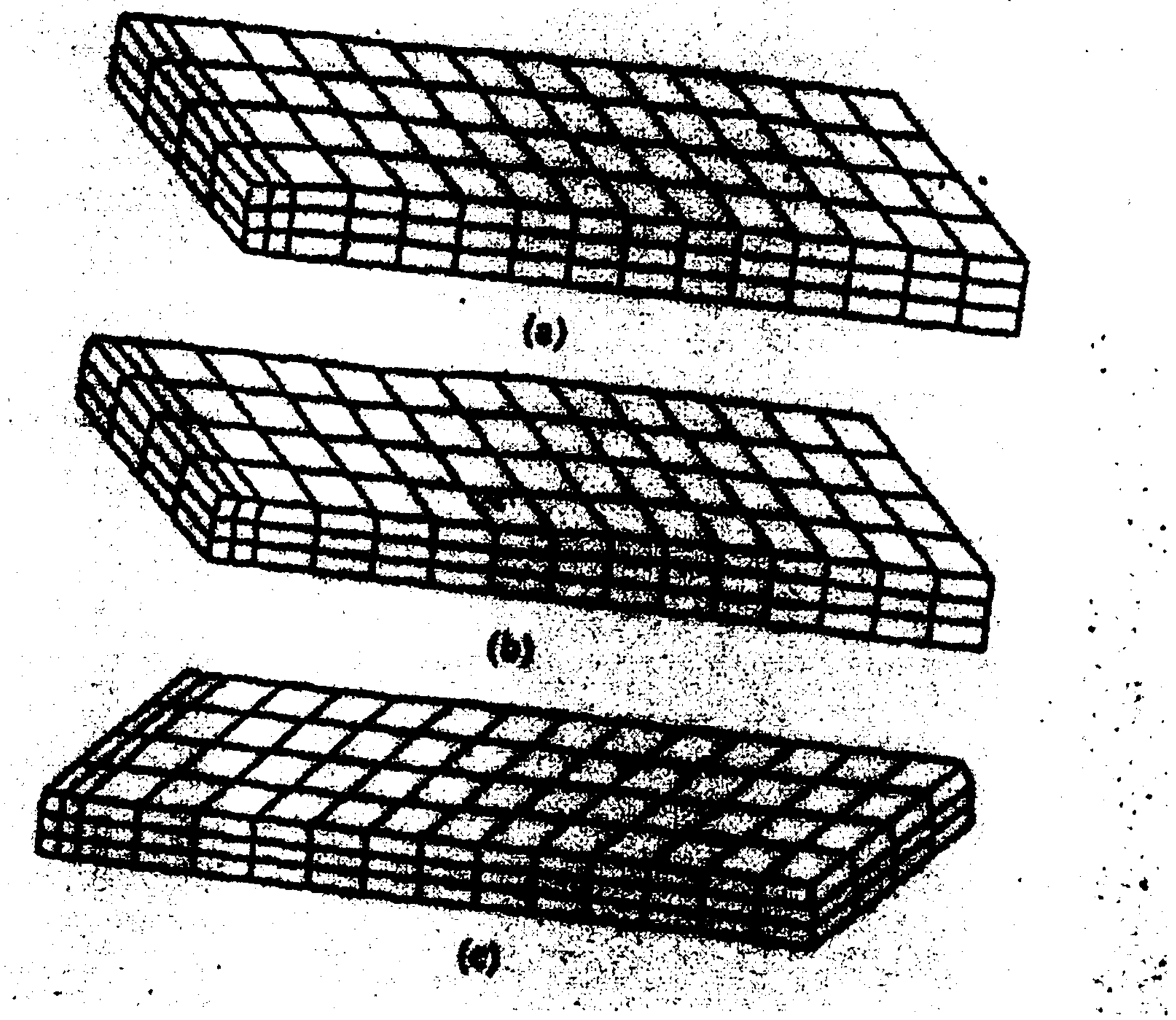


그림 9 : Slab의 형상 변화

공정 3 은 POSROL2D 를 이용해서 완전소성 재료 (Perfectly-plastic Material)에 대하여 압하율을 다양하게 변화시켜서 시뮬레이션을 수행하였다. 또 압하율의 변화에 따른 롤의 압력 분포의 변화 및 변형 특성도 조사하였고, 특히 마찰이 유효 응력의 분포에 미치는 영향에 대해서 조사하였다. 그 시뮬레이션의 결과는 그림 7 에 나타나 있다.

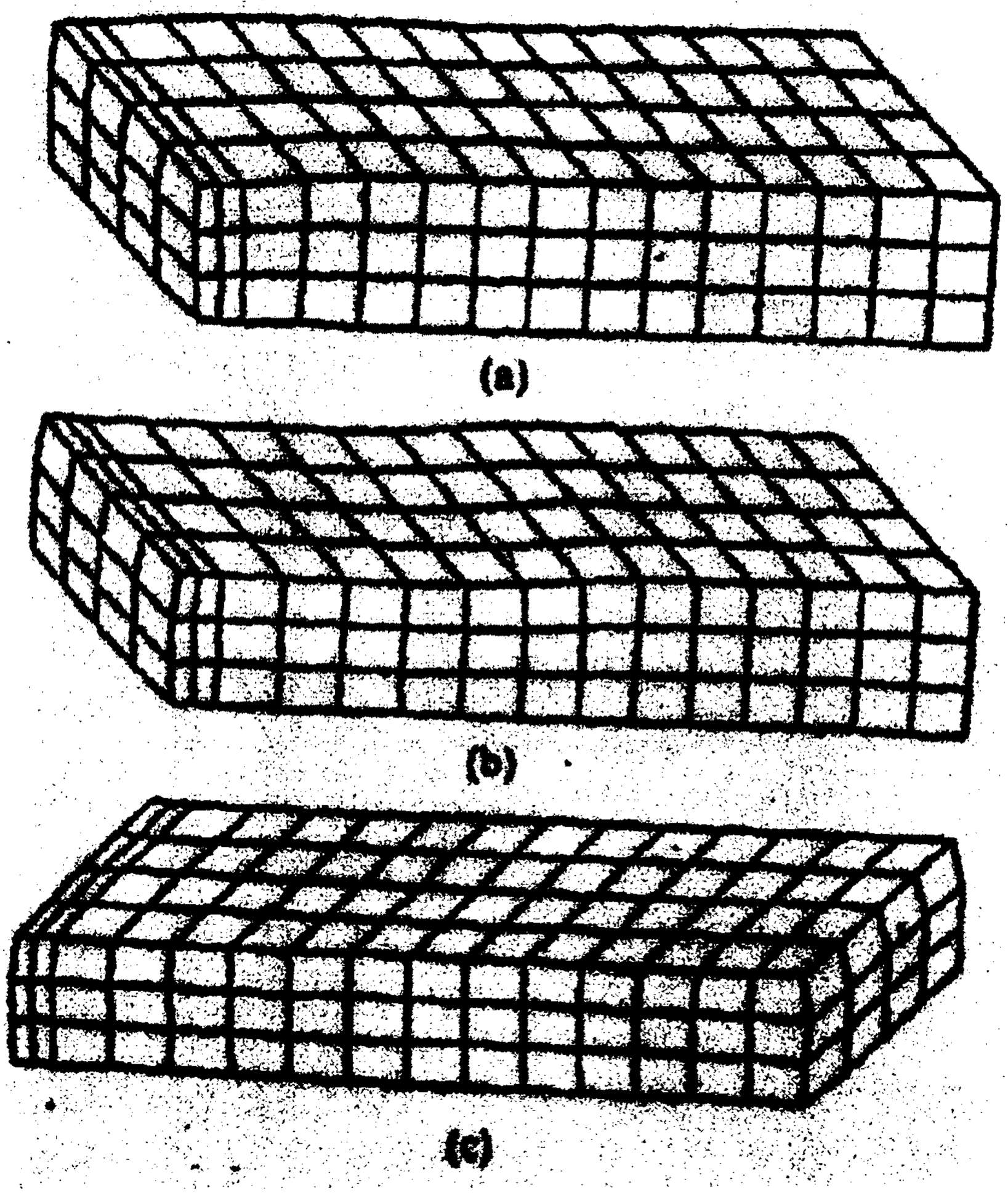


그림 10 : Slab 이 절반정도 압연된 상태

라) 공정 4의 시뮬레이션

공정 4는 유동 응력이 유호변형도 속도에 따라 변하는 경우에 대한 시뮬레이션이다 그 시뮬레이션의 결과는 그림 8에 나타나 있다.

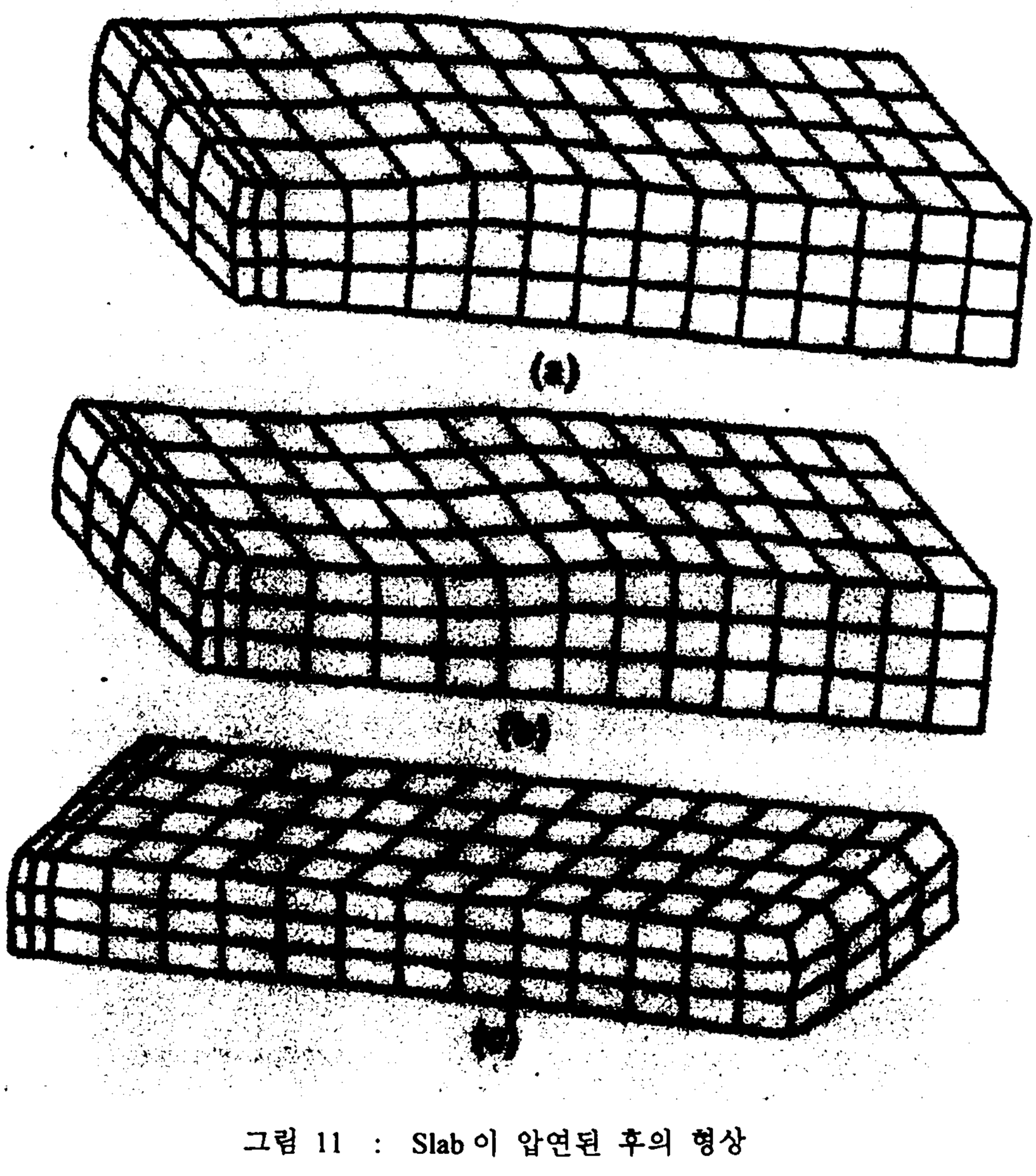


그림 11 : Slab이 압연된 후의 형상

(2) 압연 공정의 3 차원적 유동해석

후판이나 분괴의 압연에 있어서 압연된 Slab 또는 인곳의 형상 예측 능력은 실수율 관련 연구에 매우 중요하다. 압연된 형상은 복잡한 3 차원 구조이며 마찰 계수, 압연 패턴, 압하율, Slab 의 초기 형상 등의 압연 변수에 따라 민감하게 변화한다. 여기서는 3 차원 유동 해석을 할 수 있는 POSROL3D 를 이용하여 후판 압연에 있어서의 후판 형상의 변화를 예측하였다.

시뮬레이션으로 예측된 압연중의 Slab 의 형상 변화는 그림 9 에 나타나 있다.

그리고 그림 10 은 Slab 이 약 절반 가량 압연되었을 때의 형상을 보여준다. 변형이 끝난 앞부분은 압면이 압연 방향으로 약간 돌출이 된 반면, 폭의 변화는 무시할 정도로 작다.

그리고 그림 11 은 압연 완료 후의 형상을 보여준다. 뒷부분의 형상 변화는 앞 부분에 비해서 크며, 뒷변이 압연 방향의 반대 방향으로 돌출되었다. 돌출한 정도는 Slab 두께 방향 중심선에서 최대이다. 앞면과 뒷면이 볼록하게 튀어 나왔으며, 그 최대점은 Slab 의 중심에 있다. 폭방향으로의 돌출은 전후반부 끝부분으로 갈수록 증가하였으나 그 정도는 앞과 뒷면에 비하여 매우 적은 량이다.

라. 기대 효과

이 소프트웨어는 그 개발을 POSCO 에서 지원한 것으로 컴퓨터 시뮬레이션을 수행하여 기존의 후판 압연 공정을 해석하고 최적압연 변수들을 예측하여 실제 공정에 적용,

실수율의 향상과 품질 제어에 기여할 것으로 기대된다.

5. 범용 로봇 제어 언어를 이용한 그래픽 시뮬레이션

가. 로봇 언어의 특징과 추세

1970 년대를 기점으로 컴퓨터 제어 로봇들이 양산되기 시작하였고 이와 관련된 많은 제어 언어들이 나오기 시작하였다. 이 언어들은 로봇 제어에 컴퓨터를 이용하여 프로그램이 가능하도록 하였으므로 단순 동작이나 독립 시스켐의 기능밖에는 할 수 없었던 이전 로봇에 다기능성과 시스템 통합이란 매력적인 면을 제공함으로써 로보틱스 분야에 일획을 그었다. 그러나 이 언드들은 로봇 고유의 특성 표현을 어떻게 하면 잘 할 수 있느냐에 주관심사가 있었으므로 요즈음 접할 수 있는 범용 컴퓨터 언어와는 차이가 있다는 아쉬운 점이 있다. 예로 PUMA 로봇용 VAL 언어와 STANFORD 로봇용 AL 을 들 수 있는데 이들은 로봇의 특성을 효과적으로 구현하는 데에만 강조하였으므로 일반 컴퓨터 언어로서다양한 기능 제공에는 별 노력을 기울이지 않았다. VAL 의 경우 부동 소숫점, 문자 스트링을 사용할 수 없고 서브루틴에도 인자를 전달할 수가 없는 단점이 있다. 그러나, 일반 산업 현장에서 로봇의 복잡성이 증대되고 지능성이 추가되는 시점에서 일반 컴퓨터가 갖는 기능들이 요구되는 것은 당연한 현상이라 하겠다. 이런 부족한 기능을 보완하기 위한 노력이 계속되어 Unimation 사에서 제작한 VAL II 언어가 등장했으나 이도 앞 언어를 약간 보완하는 정도에 그쳤을 뿐 큰 개선은 없었다.

한편, 이런 문제점을 근본적으로 해결하기 위한 노력으로 최근 학계에서는 로봇제어용 언어로 일반 컴퓨터 언어 사용이 바람직하다는 의견이 대두되고 있다. 실제 이를 뒷받침하는 증거로 로봇 제어에 사용되고 있는 프로그램을 살펴보면 작은 일부만이 로봇에 종속적인 문장이고 대부분은 사용자 인터페이스, 데이타베이스 엑세스, 그리고 수

치 제어 등과 같은 것이라는 연구 결과가 발표되었다[9]. 이러한 관점에서 로봇에 종속적인 문장은 서브루틴의 형태로 개발하여 라이브러리 형태로 두고 프로그래머가 필요한 기능의 루틴을 서브루틴 호출 형식으로 로봇 라이브러리 안에 저장되어 있는 함수를 불러 사용하는 방식이 제안되었다. 최근, 일반 사업체에도 이런 설계 방식을 사용하여 로봇 언어를 제작하는 경우를 볼 수 있는데 일반 컴퓨터 언어로서 BASIC 이 주로 사용되고 있고, 더러 PASCAL 이 사용되는 경우도 있는데, 요즈음은 C 에 대한 관심이 높아지고 있다. 이 연구에서 사용하는 기본 제어언어인 RCCL (Robot Control C Library) 도 이러한 서브루틴 호출 방식을 사용하고 있다. RCCL 라이브러리 안에는 각종 로봇에 종속적인 문장들이 서브루틴 호출 형식으로 저장되어 있다.

나. 그래픽 시뮬레이터의 전체 구성

이 연구에서 제안한 그래픽 시뮬레이터의 전체 구조는 그림 12 에 나타나 있다.

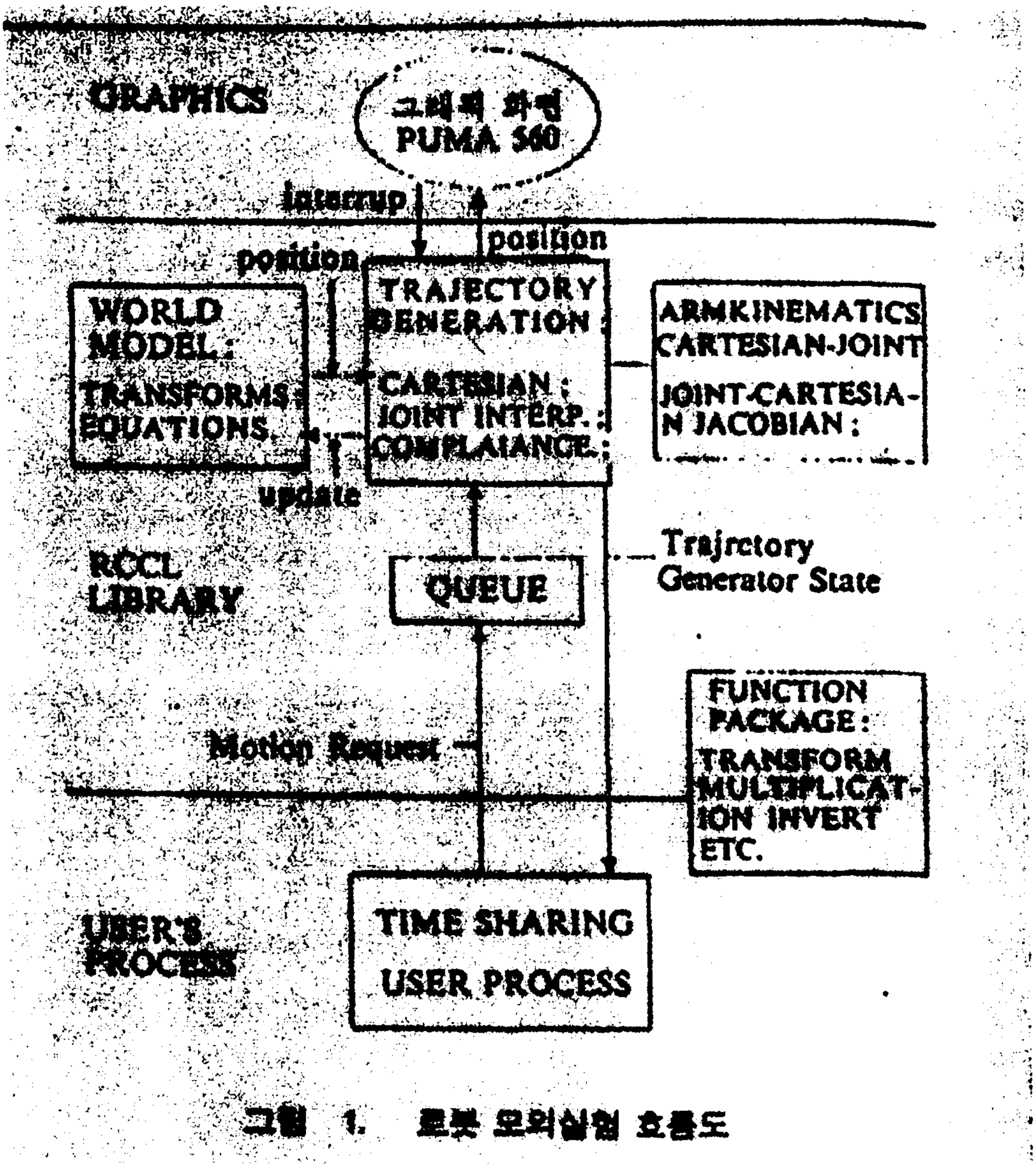


그림 1. 로봇 오퍼실험 효름도

그림 12 : 그래픽 시뮬레이터의 전체 구조

먼저 사용자는 로봇 프로그램을 UNIX 의 VI editor 와 같은 editor 를 이용해서 작성하고 일반적인 컴파일러로 로봇 라이브러리와 연결하여 컴파일한다. 컴파일된 프로그램

온 실행시 먼저 사용자가 프로그램에 명시된 운동 요구 (motion request)를 생성시키며, 생성된 운동 요구는 즉시 운동 요구를 위한 큐 (queue)에 저장된다. 이렇게 저장이 된 운동 요구 사항을 궤적 생성기 (trajectory generator)는 선입-선출 (first-in first-out) 원칙에 입각하여 로봇의 운동 (motion)을 계획하고 그 각 샘플 명령을 일정 간격으로 하위 하드웨어에 보내게 된다. 이 궤적 생성기는 항시 실행되는 것이 아니라 일종의 인터럽트 처리 프로그램과 같은 것으로 활성화시키기 위해서는 하위 시스템에서 인터럽트를 걸어주어야만 한다. 이런 구조의 형태는 실제 시스템 구성시 하드웨어 종속적인 제어부 (controller)와 하드웨어 독립적인 작업 계획 (task planning)부와의 통신을 위해서는 필수적이고 매력적인 기법이다. 일단 인터럽트가 걸리면 궤적 생성기가 활성화되어 큐에 들어있는 운동 요구 정보를 뽑아와서 RCCL 라이브러리 안에 있는 서브 루틴을 이용하여 kinematics 와 jacobian 데이타 등을 갱신시키고 운동을 계획한다. 그런 뒤 로봇의 안전 운행 범위를 검사하고 합당하면 매 샘플 타임 (sampling time) 마다 셋포인트 (setpoint)를 내보내게 된다.

그러면 그래픽 시뮬레이터 (graphic simulator)는 그 내용을 화면에 출력 (display)하게 된다. 이 그래픽 시뮬레이터는 사용자가 원하는 방향이면 어는 곳에서든지 볼 수 있도록 하는 옵션 버튼이 있어 융통성을 제공해주고 있다. 이 그래픽 시뮬레이터에서 출력되는 화면은 그림 13 에 나타나 있다.

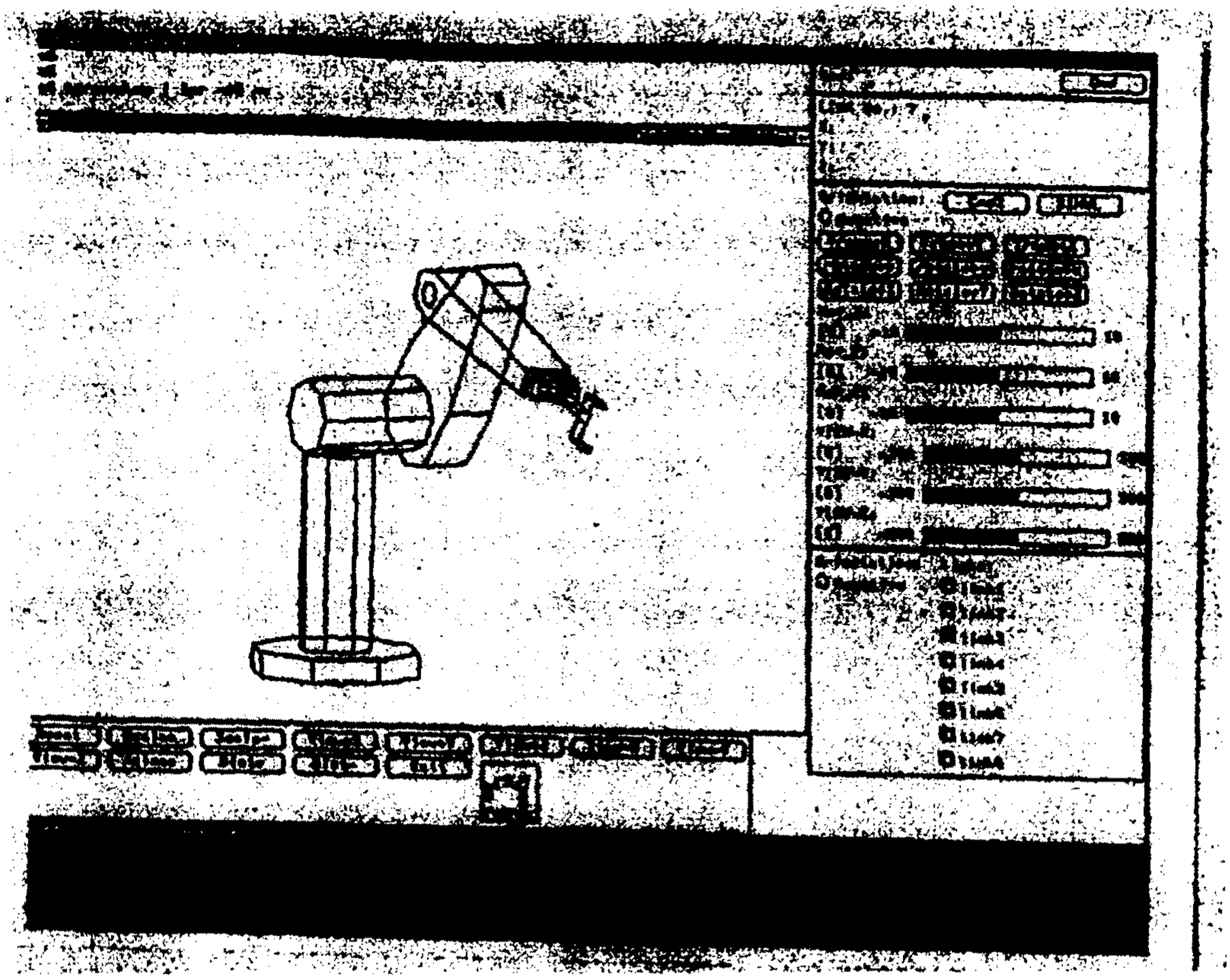


그림 13 : 그래픽 시뮬레이터의 출력 화면

다. 공간 좌표 규정 (World Modeling)

로봇 태스크 (robot task)란 공간에 도달할 점을 규정하고 그 점에 로봇이 도달하도록 매니퓰레이터 (manipulator)를 작용 (action)시키는 것이라고 할 수가 있다. 그러기 위해서는 로봇이 그런 동작을 수행하도록 하는 도구가 필요한데 RCCL에서는 구조화된 위치 표현 방식을 채택하고 있다. 즉, Robot의 각 매니퓰레이터에 고정된 축을 설정하고 축과 축의 상대적인 위치를 homogeneous matrix로 구성을 한다. 그런 다음 각 매

트릭스들을 가고자하는 위치에 대한 닫힌 폐쇄기구 경로 (closed kinematic chain) 식을 만들고 그 방정식이 만족되도록 로봇을 움직이는 방식을 택하고 있다. 그래서, 일반적인 폐쇄기구 경로식을 적으면 다음과 같다.

BASE*T6*TOOL = OBJ*GRASP*DRIVE

여기서 각 변환의 정의는 다음과 같다.

- BASE : 기준 좌표에서 로봇의 어깨까지의 변환
- T6 : 어깨에서 로봇의 선단부 (end effector)까지 변환
- TOOL : 선단부에 도구를 달 때의 변환
- OBJ : 기준 좌표에서 작업 기준 좌표까지의 변환
- GRASP : 도구에 대한 파지 위치를 나타내는 변환

이렇게 식이 적힐 때 변환 T6 는 DRIVE 변환이 원하는 위치에서 identity 가 되도록 변화된다. 즉, 그 T6 의 변환이 로봇의 선단부 동작이 되는 셈이다. RCCL 에서는 위치 방정식의 각 항을 구성하는 homogeneous 변환 matrix 가 아래의 데이타 형 (data type) 이 가능하도록 지원해 준다.

상수 변환 (Constant Transformations) : 이 변환은 궤적 수행시 변화되지 않는 데이타 형이다. 이 형태는 대부분의 로봇 제어 언어에서 볼 수 있는 형이고 실지 프로그램에서 계산의 속도를 증가시키기 위해 궤적을 계산하기 전에 미리 곱해진다.

가변 변환 (Variable Transformations) : 이 변환은 사용자 프로세스의 실행 전 구간에 걸쳐 쓰고 읽을 수 있다. 따라서 이 변환식이 바뀌면 궤적도 즉시 그 변화를 수용하게 된다. 여기서 프로그래머는 의미있는 결과를 얻기 위해 미소 변화를 제공할 수가 있다. 변화를 줄 때 그 시기는 임의로 할 수가 있고 이런 형의 변화는 일반적으로 센서의 정

보를 샘플 레이트에 얻을 수 없을 때 tracking application 에 유용하게 쓰일 수 있다.

홀드 변환 (*Hold Transformations*) : 이 변환은 어느 때라도 사용자 프로세스에 의해 수정될 수가 있다. 이런 류의 변환이 포함되는 운동 요구 (motion request)가 제시되면 시스템은 이 내용을 시스템 내부 스택에 복사해 두게되고 이 때 복사된 부분은 운동 요구가 일부분이 된다. 응용을 보면 사용자 프로세스가 예측 가능 시간내에 위치 정보를 얻을 수 없는 경우, 이 변환을 써서 매니퓰레이터를 정지시키지 않고 입출력을 할 수 있다.

함수화 변환 (Functionally defined Transformations) : 이 변환은 C 함수안에 포함되고 해당 궤적 계산동안에 이 C 함수가 샘플 레이트로 계산된다. 물론 이 함수의 실행 시간은 충분히 짧아야 한다. 만일 계산될 값이 어떤 인자의 함수라면 parametrized 궤적을 얻을 수 있다. 예를 들어 그 값이 센서값의 함수이면 매니퓰레이터를 제어하기 위한 동기된 센서 피드백 값을 얻을 수 있다.

라. 궤적 생성 (Trajectory Generation)

RCCL 은 2 가지 형태의 궤적을 제공하고 있는데 그 하나는 joint mode 이고 다른 하나는 *cartesian mode* 이다.

(1) Joint Mode

각 운동 요구에 의해 최종 매니퓰레이터의 위치식이 정해지면 joint setpoint 가 inverse kinematics 로 정해지고 그 셋포인트를 매 샘플링 시간마다 선형 보간한다. 이런 류의 운동은 아주 효율적이고 단지 조인트 가속도와 최대 조인트 속도에만 제한을 받는다. 이 운동의 궤적은 직선이나 예측 가능한 경로는 아니지만 각 셋포인트 간을 움직일 때 최소 시간이라는 관점이 중요시 될 때 즉, 큰 운동 궤적이 요구될 때는 아주 용이하다.

(2) Cartesian Mode

매니퓰레이터의 경로를 매 샘플 시간마다 DRIVE 함수를 써서 계산하고 그 것이 inverse kinematics 식을 풀면 joint setpoint 가 주어진다. 이 때 그 경로는 직선 경로이고 예측 가능한 경로이다. 단 항상 특이점(singurality)의 가능성이 있는 것이 좀 흠이라 하겠다. 이 운동에 있어 아주 중요한 역할을 하는 것이 DRIVE (s) 함수이다.

마. 동기화 (Synchronization)

동기화는 프로그램이 센서나 운동 종료 조건시 또는 가변 변환 (variable transformation)이 매니퓰레이터의 위치를 변화시키는 것과 같잉 외부의 정보에 의존할 때 필요하다. 보통 로봇 프로그래밍 언어에서는 MOVE 라는 문장이 매니퓰레이터의 운동과 동기화되어있다. 그러나 가장 좋은 경우라면 그 선택이 사용자에게 맡겨져 있어야 할 것이다. 즉, MOVE 실행 후 프로그램이 계속 될 것인지 아니면 그 운동이 끝날 때까지 프로그램을 정지해야 할지를 사용자가 결정하도록 한다는 뜻이다.

RCCL 에서도 move 라는 명령어가 있기는 하나 move 명령어와 동기되어 로봇이 움직이는 것은 아니다. RCCL 에서는 특이하게 큐잉 (queueing) 구조와 동기화 프리미티브 (primitive)를 사용해 다소 복잡하기는 하나 아주 유연성있는 환경을 제공하고 있다. 몇 개의 운동 요구 사항이 먼저 프로그램되고 move 명령에 으해서 운동 (motion)이 실행되는 것이 아니라 호출 순서대로 큐 속으로 들어간다. 그 각각은 큐에서 자신의 운동 (motion)이 수행되기를 기다리고 있는데 이 들을 어떻게 서비스하느냐는 오로지 사용자의 권한이다. 즉, 하이레벨 (high-level) 프로그램에서 사용자가 선택을 한다. 사용자의 선택에 의한 각 운동은 동기화 혹은 비동기화될 수 있다.

바. RCCL 을 이용한 프로그래밍

(1) 운동 좌표 표현 (Location Description)

RCCL 함수는 사용자가 동적으로 homogeneous 변환을 생성할 수 있게 해준다. 기본적인 함수 호출은 다음과 같다.

t = newtrans (type);

여기서 t 는 생성된 homogeneous 변환의 포인터이다. 그리고, type 은 원하는 형태의 변환을 규정한다. 즉, 앞에서 기술한 const, hold, varb 형을 말한다. 여기서 생성된 변환식의 값은 항등 변환식인데 이 기능을 원하면서 초기화 과정을 원한다면 다음 함수 호출 형태를 취할 수 있다. 예를 들어,

t = gcntr_rot (px, py, pz, v, a);

여기서 px, py, pz 는 병진운동 부분을 나타내고, v, a 는 v 벡터를 중심으로 a 각도 만큼 회전을 뜻한다. 위와 더불어 오일러 각도, roll pitch yaw 각도 등을 다루는 함수가 RCCL 에 있다. 지금까지의 자료를 바탕으로 RCCL 에서는 위치방정식을 makeposition 이라는 함수를 호출하여 만든다. makeposition 에서 기술된 위치 방정식은 궤적 생성기에서 자동적으로 어떤 특성을 갖도록 DRIVE 변환을 계산하도록 되어 있다.

makeposition 은 가변 인수를 인자로 쓰고 있는데 이는 C 언어에서 variable argument 를 제공해 주고있기 때문이다. 함수에서 가변 인자로 사용된 인자들은 스택에 순서대로 차곡 차곡 쌓이므로 그 내용의 순서를 찾는데는 별 무리가 없다. 예를 들어, base,

tool, obj, grasp 이라는 변환이 위의 gentr_trsl ( )이나 gentr_rot ( ), gentr_rpu ( ) 등을 통해서 생성되었다고 가정하면, makeposition 에 대한 C 의 정의는 다음과 같다.

```
POS_PTR makeposition ( lhs[,lhs]..., EQ, rhs, [,rhs],..., TL, tl );
POS_PTR p;
```

위 예제에 해당하는 함수 호출은 다음과 같다.

```
POS_PTR p;
p = makeposition ( base, t6, tool, EQ, obj, grasp, TL, tool );
```

여기서 EQ 는 lhs 와 rhs 를 구별하기 위한 구별자이고 TL 은 tool 이 무엇인가를 알려주는 구별자이다. makeposition 함수는 변환 그래프를 만들어주고 그 ring data 구조의 포인터를 반환하는 역할을 한다. 사용자에게 position 은 C 구조체로 구현되어 있는데 그 정의는 다음과 같다.

```
struct position {
        char *name;
        int code;
        float scale;
        event end;
}
```

여기서 code 는 운동 (motion)의 종료시 그 이유를 표시하고 scale 은 운동 (motion) 시 0 에서 1 로 변화하는데 DRIVE 변환의 인자로써 쓰이며, 궤적 경로의 중간점에서

사용자 프로세스를 동기화시키는 작용을 한다. 마지막으로 end 는 해당 운동 (motion) 의 종료를 알리는 event count 이다.

(2) 운동 기술 (Motion Specification)

다음 함수 호출은 궤적 생성기에 운동 요구를 전달하는 작용을 한다.

move (p);

이 함수를 호출하면 실제 앞에서 만든 위치방정식에 대한 정보와 아래에 기술된 운동에 관한 정보를 테이블로 구성하여 큐에 저장시키는 기능을 한다.

setvel (tv, rv) : 병진과 회전 속도 결정

setmod (m) : catesian 혹은 joint motion

setconf (c) : arm configuration 의 변화 요구

sample (s) : 샘플 주기를 변화

(3) 동기화 (Synchronization)

RCCL 은 event 를 사용하여 사용자 프로그램과 매니퓰레이터 운동을 동기화 한다. 운동 요구는 큐에 들어가고 한 운동 요구가 끝나면 다음 운동 요구가 큐에서 FIFO 형태로 불려진다. 많은 경우에 프로그램과 매니퓰레이터 운동을 동기화시키기 위해 event 가 필요하다. RCCL 에서 event 는 정수로 정의되어 있다.

typedef int event;

event 는 원래 count 이다. 그 것이 양수이면 몇 개의 운동 요구가 기다리고 있음을 나타낸다. event 발생시마다 하나씩 줄어가고 결국은 0 이 된다. 이 때는 기다리고 있는 운동 요구가 아무도 없음을 나타낸다. RCCL 은 completed 와 end 라는 event 를 사용하는데 전자는 운동 큐가 비었을 때에 사용되고 후자는 한 운동이 끝났을 때에 발생한다. 사용자 프로그램은 event 와 동기화시키기 위하여 프리미티브 waitfor 를 사용한다.

6. 직접 구동 로봇의 그래픽 시뮬레이션 시스템

가. 연구의 필요성

공장 자동화의 실현은 현재 국내의 모든 기업들이 안고 있는 가장 중요한 당면과제중의 하나이다. 최적의 자동화 공정 설계를 위해서 현장의 공장 경영자들이나 공장 설계자들은 새로운 계획이나 아이디어들에 대한 여러 가지의 대안들에 대해서 가능한 최선의 분석 방법과 객관적인 자료들에 근거하여 이들 각각을 평가하여 이 중에서 분석 결과가 가장 좋은 대안을 선택한다. 이러한 분석 방법에 있어서, 실제 시스템의 동작 과정 및 그 결과를 가장 유사하게 분석 평가할 수 있는 방법으로 시뮬레이션을 많이 이용하고 있다. 그리고 엔지니어링 웍스테이션의 폭넓은 보급과 컴퓨터 계산 능력의 향상에 힘입어 시뮬레이션 소프트웨어에서의 그래픽 시스템의 중요성이 날로 중대되어가고 있다. 로봇 제어 시스템에서의 그래픽 시뮬레이터는 로봇 프로그래밍에 있어서 계획된 동작 경로가 방해물과 충돌을 하는지의 여부를 시각적으로 검증하는데에 사용된다. 그래픽 시뮬레이터를 통해서 동작 계획의 정확성이 충분히 확인되고 난 후에 로봇을 작업시킴으로써 계획된 경로의 오차로 인하여 값비싼 로봇이 충돌이나 다른 미쳐 예측하지 않았던 상황으로 말미암아 파손되는 것을 막을 수 있다[12].

나. 그래픽 시뮬레이션 시스템의 구성

로봇 시스템의 그래픽 시뮬레이션 시스템은 입력 로봇의 컴퓨터 표현에 관한 객체 모델링 부분과 저장된 로봇 정보와 입력받은 제어 명령에 의해 실제 로봇의 동작을 화면에 출력하는 로봇 제어 부분으로 나눌 수 있다.

(1) 객체 모델링 시스템

이 연구에서는 로봇과 로봇의 작업장을 나타내기 위해 볼록한 (convex) 성질을 가진 기본 물체를 기본으로 하고 이들 기본 물체들로부터 계층적으로 상위의 물체들을 구성하였다. 모델링의 기본이 되는 기본 물체는 물체들의 충돌 계산이 용이하도록 Star-Edge Boundary Representation 방법을 사용하였다. 이 방법은 물체를 면 (face), 모서리 선분 (edge) 그리고 점 (vertex)들의 관계들로서 나타낸다. 이 표현 방법의 구현을 위해서는 기억 용량의 절약을 위해 자료의 첨가와 제거의 장점을 가진 연계 리스트 (linked list)를 사용하였다.

(가) 자료 구조

로봇 시스템을 위한 자료 구조는 최하위의 기본 물체 (object)로부터 로봇이나 작업장과 같은 최상위의 물체를 계층적으로 구성한다. 즉, 면, 모서리 선분 그리고 점들의 상호 관계로 표현된 기본 물체들로부터 하나의 로봇 매니퓰레이터를 나타내며 이들 매니퓰레이터들과 작업 대상물, 그리고 그 밖의 주변 물체들로써 전체적인 로봇 시스템을 구성한다. 이 때, 기본 물체는 여러 가지 연산들을 용이하게 하기 위해 볼록한 성질을 가진다. 구현 과정은 로봇 시스템의 묘사 화일로부터 시스템을 구성하는 물체에 대한 정보를 입력받아 그 물체의 자료 구조에 필요한 기억 용량을 할당하고 이 새로운

물체를 이미 읽어들인 기존의 물체들과의 상호 관계에 따라 포인터를 통한 연계 리스트를 생성하여 연결한다.

(나) 로봇 시스템 입력

로봇과 그의 작업장에 대한 정보를 입력하기 위한 컴파일러를 구현하기 위해서 Lex와 YACC를 사용하여 로봇의 묘사 화일 (Robot Description File)로부터 어휘 분석과 구문 분석을 하여 필요한 자료 구조를 생성, 입력된 로봇 시스템의 정보를 저장한다. 따라서 로봇 묘사 화일을 받아 들여서 먼저 Lex에서 토큰 (token)이라는 기본 단위를 생성한다. 이 때 토큰은 모델링 과정에서 가정한 매니퓰레이터, 링크, 관절, 기본 물체, face, edge, vertex 등 필요한 모든 자료 구조에 대응되어야 한다. 즉, 묘사 화일에서 axis, edges, point, define_arm 과 같은 단어들을 입력받으면, TAXIS, TFACES, TEDGES, TPOINT, TDEF_ARM 등과 같은 토큰을 생성한다. 이 때 물론 숫자나 문자열을 받으면 해당값을 가진다. 이와 같은 토큰들은 다음 단계인 구문분석 단계의 입력이 된다. 구문분석 단계에서 YACC은 Lex로부터 생성된 토큰들을 받아 묘사 화일의 구조적 오류를 검사하고 올바른 입력 형태에 대해서는 해당되는 자료 구조를 위해 기억 장치를 할당하여 입력된 자료를 저장한다. 예를 들면, setup-system이라고 나타낸 전체 시스템은 시스템의 시작을 나타내는 TSETUP이라는 토큰과 시스템의 이름을 나타내는 TIDENT라는 토큰, 매니퓰레이터들을 나타내는 statements 그리고 시스템의 끝을 가리키는 TEND_SETUP이라는 토큰들로 이루어져 있다. 또 statements는 하나 또는 여러 개의 arm_state로 구성되어 있다. arm_state는 하나의 토큰이나 여러 개의 토큰들의 복합어로써 구성 될 수 있다. YACC은 입력된 묘사 화일이 위와 같은 구조가 아닐 때는 에러를 표시하고 올바른 입력에 대해서는 지정된 함수를 호출하여 필요한 작업을 행한다. 즉 start_arm 조건을 만족하면 arm_alloc ( )이라는 함수를 호출하여 매니퓰레이터 자료 구조를 위한 기억

장치를 할당한다.

(2) 링크 상호 관계 및 관절 제어 방법

로봇 제어부 (robot controller)는 외부로부터 받은 제어 명령을 해석하여 저장된 객체 모델들의 정보를 변환시켜 결과를 화면을 통하여 보여준다. 자료 구조에서 알 수 있듯이 각 링크의 관절은 기준 좌표계를 가지며 제차 변환 행렬 (homogeneous transformation matrix)를 이용하여 기본 링크 (base link)로부터의 수직적 관계에 의해 작업 공간에서의 절대적인 현재 위치를 계산한다. 로봇 제어 명령은 컴퓨터 그래픽스를 이용한 대화적인 방법과 Command Interpreter 를 이용하여 키보드나 화일로부터 입력을 받는 방법이 있다. 다음 단계에서 입력된 제어 명령을 해석하여 저장된 객체 모델 정보를 변환시킴으로써 해당 명령을 수행시킨다. 마지막으로 수행 결과를 컴퓨터 그래픽스의 다양한 기능들을 이용하여 화면에 나타낸다. 대화적인 로봇 제어를 위해서는 먼저 묘사 화일로부터 입력된 로봇의 링크 정보에 따라 대항 링크 갯수만큼의 제어 버튼을 만들이 이들 제어 버튼들로부터 제어 명령을 받는다. off-line 제어는 로봇의 현재 정보를 파라미터로 받음으로써 수행할 수 있다. 그림 14 는 postech ddarm 의 조인트 1 과 조인트 2 가 90 도의 각도를 가지는 경우를 나타낸다.

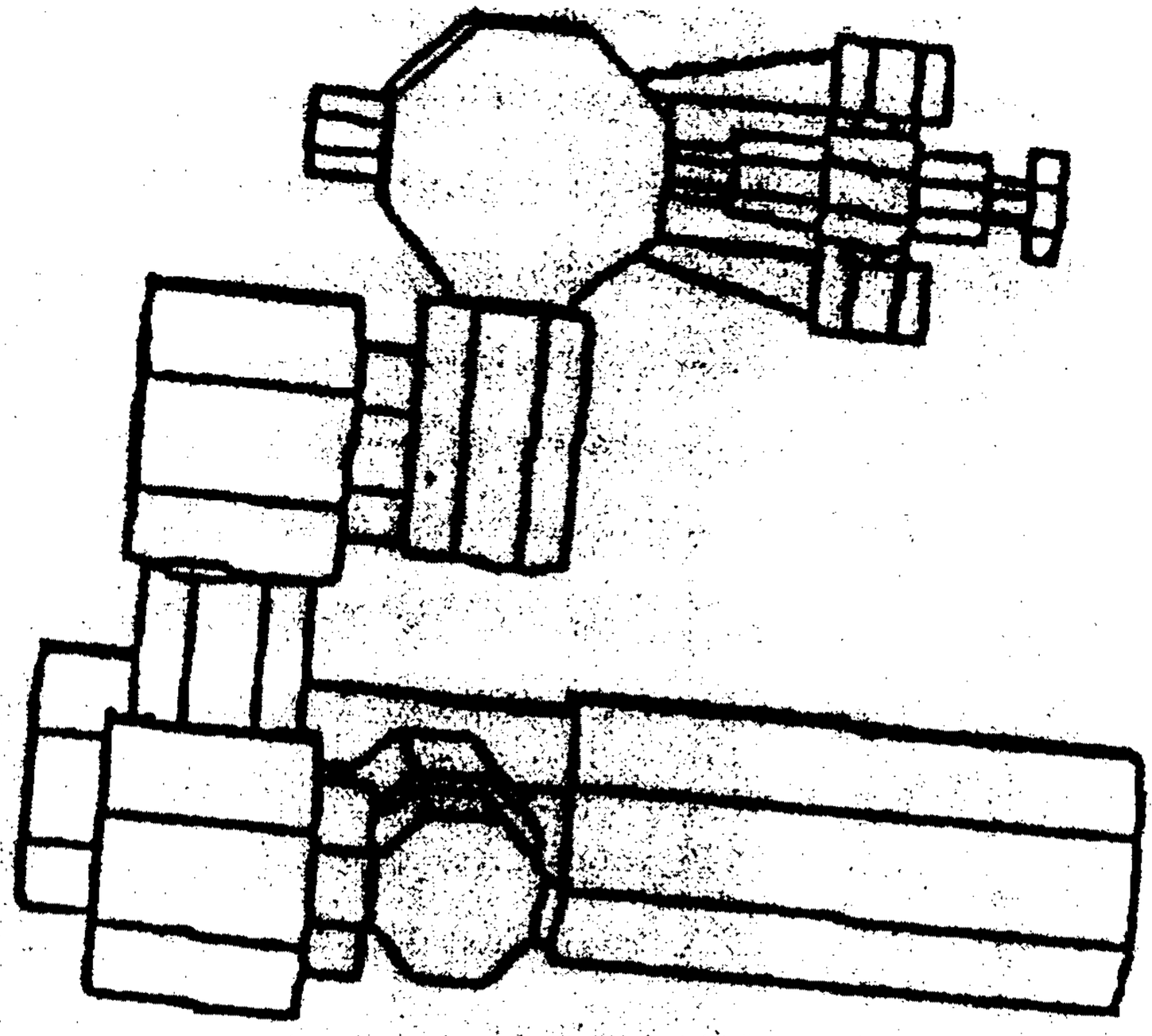


그림 14 : off-line 제어의 예

관절의 제어는 로봇의 종류에 따라 각 해당 링크들이 가지는 상대적인 위치로부터 링크들의 수직적 관계에 의해 절대적인 좌표를 계산한다. 예를 들면, 그림 15 는 그래픽 제어의 경우 원하는 링크의 버튼을 누르면 해당 링크가 지정된 운동을 한다.

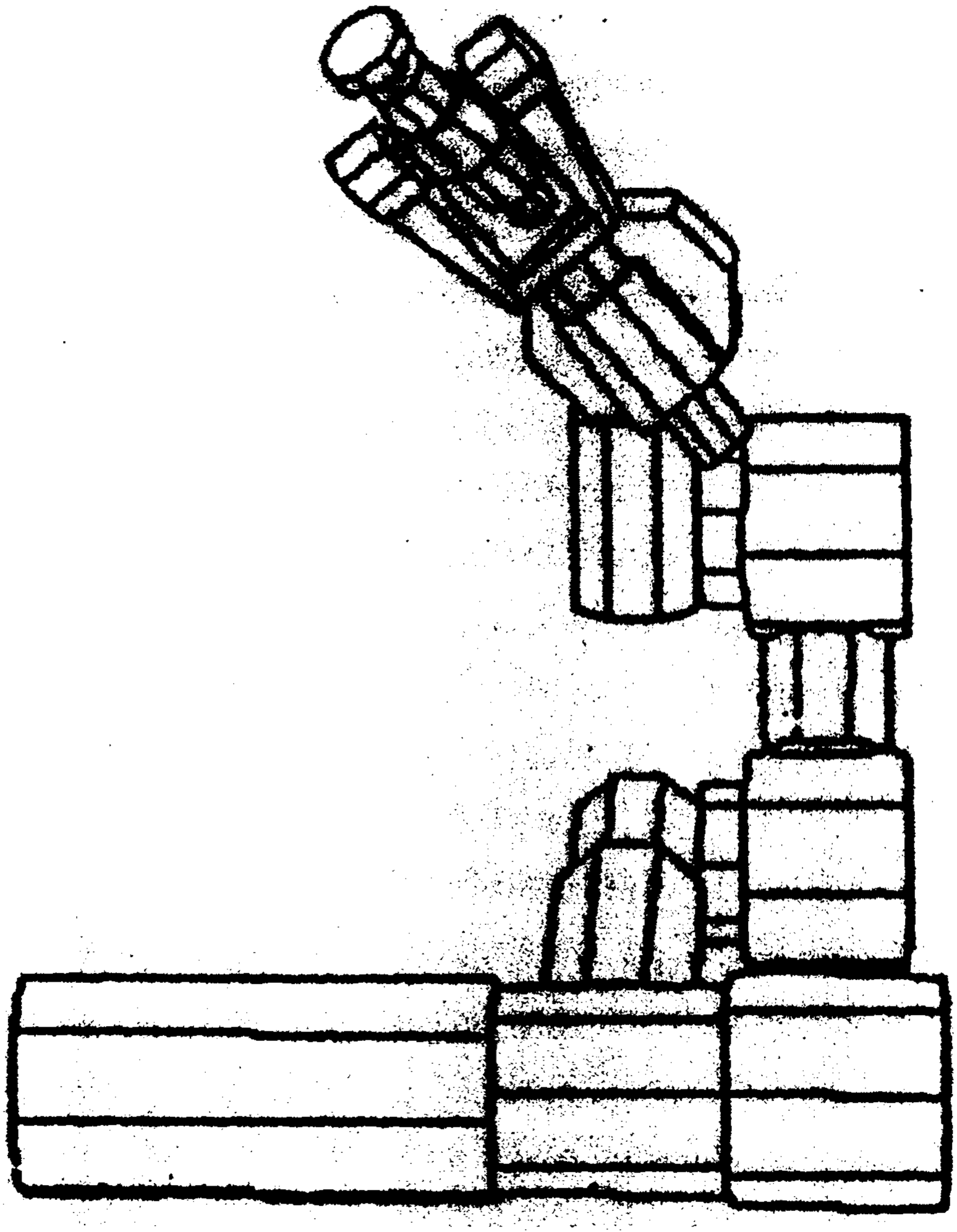


그림 15 : on-line 제어의 예

7. 전로 반응 공정에 대한 동적 모델링 및 시뮬레이션

가. 전로 공정의 소개

1949 년도 오스트리아의 Linz & Donawitz 두 회사에서 순수한 산고를 노상부에 불어 넣이 줌으로써 극저탄소까지의 탈탄이 가능하게 하고 경제적이며 생산성을 향상시킨 전로 (LD Converter)를 개발하였다. BOF (Basic Oxygen Furnace)라 불리우기도 하는 전로는 용광로로부터 나온 용융된 쇳물, 즉 용선중에 포함된 다량의 탄소 성분과 인, 규소, 망간 등의 불순물들을 산소와의 화학 반응을 통해 적정량 이하로 낮춤으로써 양질의 강철을 생산하는 거대한 화학반응기라고 할 수 있다.

전로내의 반응으로 인한 성분들의 동적 특성은 조업의 주 목적인 탄소 함량 조절과 온도 조절에 있어 많은 영향을 미치므로 성분들의 동적 거동에 대한 이론적 규명은 전로조업 연구에 있어 중요한 역할을 하여 왔다. 1960 년대 이후 많은 연구에도 불고하고 전로에서 일어나는 현상의 복잡성과 각 상황에 따른 다양한 조건들과 조업시 측정키 어려운 여러 인자들이 존재함으로 전로내의 각 성분의 동적 특성에 관한 연구는 아직도 미흡하다고 할 수 있다. 현재의 조업에 있어서는 대부분이 시간에 무고나하게 입력과 출력에 대한 총괄지수식을 세워 운영하는 정적 모델을 적용하다가 이에 대한 오차를 보완하기 위해 취련 후 반기쯤에 동적 모델을 적용하고 있는 실정이다.

이 연구에서는 전로의 반응 기구과 물질 전달, 열 전달을 규명함으로써 한 철강회사에서 보다 효율적인 양질의 강을 생산하고 이 때 부수적으로 발생되는 유용한 부생 가스의 효율적 이용 및 전로의 이론적 조업 방안을 모색한다[10]. 이를 위해서는 먼저 주어진 계에서 가장 중요한 문제는 탄소 함량을 제거하는 탈탄 반응을 통하여 원하는 성분 함량의 용강을 만들어 내는 것으로 이에 영향을 주는 여러 요소들을 고려하여야 한

다. 요인들로써는 장입되는 용선의 조건과 주원료중 하나로 들어가는 고철의 정확한 성분을 알아야 하고 온도 조절 목적으로 투입되어지는 철광석 (소결광)의 화학반응시 산소의 거동 문제, 또한 조업시 용선중 다른 성분들 (Si, Mn, P, FeO 등)이 온도와 탈탄 반응에 미치는 영향과 반응에 참여하는 최적 산소량 결정 문제 등을 알아야 하며, 또한 출강시 목표로 잡은 온도와 탄소성분함량 예측 문제, 반응시 부생 가스의 최적 수량 결정들을 알아야 한다.

이 연구에서는 이러한 문제들을 해결하기 위해서 노내에서 일어나는 화학반응식들을 세웠고 시간에 따른 각 성분의 물질 수지식과 에너지 수지식을 세움으로써 동적 모델을 설정한 후 이에 대한 시뮬레이션을 수행하였다. 시뮬레이션 방법으로는 몇가지 가정들을 세운 다음 Runge-Kutta 방밥을 이용하여 미분방정식을 풀었고 최적화 기법을 통해서 모델식에 나타난 많은 매개변수값들을 추정해 냄으로써 기존의 현장 조업 자료와 일치하도록 하여 각 성분들의 동역학적 변화를 계산하는 조업식을 유도하였다.

나. 전로 공정의 동적 모델 설정

(1) 계의 설정

산화 반응을 통해 탄소 함량과 온도를 적정값으로 조절함으로써 양질의 강철을 생산하는 것을 주목적으로 하는 전로 공정에 있어서 용철내에 존재하는 불순물들의 화학반응은 탄소의 거동과 온도에 많은 영향을 주며 또한 노내에서 균일한 상태가 아닌 복잡한 거동을 보임으로써 편의상 몇 개의 영역을 나누에 계를 설정하였다.

용철내의 성분들의 모든 화학 반응은 lance 로부터 산소와 가스가 접촉하는 계면에서만 일어나는 것으로 가정하여 용철이 혼화되어 녹는 영역과 반응하는 영역을 함께 묶어서 영역 1 로 설정하였고, slag 영역을 영역 2 로 설정하였다. 또한 취련중 용철로부터

물리적인 힘에 의해 튀어나오는 용철 입자들이 slag 과 생성되는 기체들과 함께 혼화되는 영역을 영역 3 으로 설정하여 노내의 계를 세 영역 (zone)으로 설정하였다. 또한 노 상부의 관을 통해 기체가 나가는 가스 영역을 세웠다.

영역 1 에서의 특성을 살펴보면 lance 로부터 음속 이상의 속력으로 강하게 공급되어지는 산소와 노 밑으로부터 흡입되어지는 질소와 아르곤에 의하여 용철은 고르게 혼화되어지며 이 때 용철중 반응 성분들인 탄소, 망간, 인, 규소 등은 반응 계면으로 이동되어져서 반응을 일으켜 탄소를 제외한 모든 성분들은 slag 이 된다. 이 때 온도 보정을 위해 취련중에 투입되어지는 소결광은 환원 반응을 일으켜 Fe 는 영역 1 에 속하게 되고 해리된 산소는 전량이 산화 반응의 공급원으로 쓰이는 것으로 하였다. 또한 소결광과 용철중 고체로 장입되어지는 고철 (scrap)과 냉선 등은 취련서 곧바로 액상으로 변화되지 않고 취련 말기까지 서서히 용융되어진다. 또한 투입되어지는 고철에 있어 예측치 못한 불순물들이 있는 경우 (산화성 물질) 노내 온도에 영향을 주며 용철이 전로밖으로 넘쳐 나가는 슬로핑의 원인중 하나가 되기도 한다.

영역 2 는 용철중 탄소이외의 산화반응물들이 취련 초기에 장입된 부원료 (석회석) 과 반응하여 slag 을 형성하는 영역이다. 규소 (Si)는 전로 조업에 있어서 상당히 중요한 역할을 하는 원소로 용철중 규소 성분은 전량 반응하여 slag 화 된다. 반응이 크고 발열량이 매우 높으므로 대부분 취련 초기 (반응 초기 5 분간)에 완전 산화되므로 초기 온도에 많은 영향을 준다. 또한 용철중 초기 규소량을 앎으로 조업시 염기도를 결정하게 되면 초기에 규소량을 앎으로 조업시 염기도를 결정하게 되면 초기에 투입될 부원료량을 최적으로 투입할 수 있게 되어 효율적인 조업을 하게 된다. 인과 망간의 경우는 가역 반응이 되어 취련 중간쯤 용철중 인과 망간 성분이 증가하는 경향을 보인다. 철의 산화반응으로 FeO 와 일부의 소결광이 생성되어지며 이들은 또한 반응중 다시 환원되어지기도 하여 중요한 산소 공급원이 된다. 이 영역에서는 lance 로부터의 산

소와 직접 반응을 일으키지 않는다. 초기 부원료로 투입되어지는 생석회, 생 돌로마이트, 경소 돌로마이트 등은 고체상이므로 반응시 이 성분들의 용융은 취련 말기까지 계속된다. 취련시는 격렬한 반응과 생성 기체등으로 인하여 일정한 형태를 갖추지 못하나 취련 말기쯤 slag 층이 형성되어 진다. Slag는 취련후 일부 노내에 잔류되어 노벽을 보호하는 역할을 하기도 한다.

영역 3 에서는 취련시 강하게 흡입되어지는 산소와 용철계면이 충돌함으로써 불순물 성분들을 포함한 용철입자들이 튀어오르게 되고 이와 함께 탈탄 반응으로 생성되어지는 기체는 slag과 용철입자돌의 겉보기 부피를 크게 하면서 가스 영역으로 올라가게 된다. 고철내의 불순물들에 의한 갑작스러운 반응이나 lance로부터의 산소의 투입 속도가 약할 때 (soft blowing) 이 영역의 부피가 극히 커지면서 슬로핑이 일어나게된다. 산화 반응에 있어서 용철 입자들의 반응 계면은 상당히 크므로 산소와의 직접 반응으로 탈탄 반응시 영향을 준다.

(2) 화학 반응

(가) 영역 1 에서의 화학 반응

이 영역에서의 대표적인 반응물은 탄소, 규소, 인, 망간, 철 등이다. 탄소, 인 등 고체 입자를 지닌 반응물들은 융점이 매우 높으므로 조업 온도에서 고체 입자로 액상인 철쪽으로 녹아들어가게 되고 이 상태에서 기체 산소와 화학 반응을 일으키게 되므로 전형적인 비균일 반응이 된다.

(나) 영역 2 에서의 화학 반응

영역 2는 부원료로 투입되는 생석회와 영역 1 에서 반응된 산화생성물로 이루어진 slag 영역을 말한다. 이 영역에서 주된 반응은 산화인과 산화규소 등의 CaO 와의 반응,

MnO 의 가역 반응, MgO 의 반응, FeO 의 반응등이 있다. 영역 1 의 계면에서 산소와 반응한 인은 CaO 와 결합하여 매우 안정한 인산염화합물을 생성한다.

(다) 영역 3 에서의 화학 반응

영역 3 의 반응 메카니즘은 많은 연구에도 불구하고 정량적인 규명이 어렵다. 다만 영역 1 에서 일어난 탈탄 반응에 의해 생성된 일산화탄소의 일부가 이 영역에서 산소와 2 차 연소 반응을 일으켜 이산화탄소를 생성한다고 추정하였다.

(3) 물질수지식

이 연구에서의 물질 수지식을 세우는데에는 다음과 같은 가정을 하였다.

- 요철중 반응하는 성분들에 있어 다른 영역으로의 대류물질전달을 무시한다.
- 반응, 생성하는 기체들은 모두 이상 기체로 가정한다.
- 용철중 C, Si, Mn, P, Fe 이외의 다른 성분들의 동적 거동에 의한 영향은 무시한다.
- 장입되어지는 주 워료중 고철의 성분은 Fe 로 가정한다.
- 철광석 (소결광)의 주성분은 Fe2O3 로 가정한다.
- 반응시 분진(fume)의 양은 무시한다.
- 투입되는 생석회는 100% CaO 로 가정한다.

(가) 영역 1 에서의 물질 수지

영역 1 에서의 총괄물질수지식은 용철중 각 성분의 산화반응으로 인한 변화율과 취련중 투입되는 철광석의 변화량, 산소의 함량, 고철의 변화량, 철의 산화 반등들을 고려하여 세웠다.

(나) 영역 2 에서의 물질 수지

영역 2 에서의 동역학적 거동은 용선에서의 각 성분들의 동적 변화량과 밀접한 관계를 맺고 있다. 즉 용선중 망간, 인, 규소, 철의 산화물들은 모두 slag 이 된다.

(다) 영역 3 에서의 물질 수지

영역 3 은 반응중 용철입자들과 slag, 또한 생성 기체들, 분진들이 어우러져 emulsion을 형성하고 있는 영역이므로 반응물들의 정량적 거동을 규명하기가 어렵고 탄소에 비하여 다른 성분들의 영향이 매우 적으므로 탄소이외의 성분들의 변화에 대한 고려 사항을 무시하였고 탄소의 화학 반응은 영역 1 에서와 같은 메카니즘을 따른다고 가정하여서 이 영역 1 에서의 탈탄 효과를 반응속도 상수항에 넣어 고려하였다.

(라) 에너지 수지식

에너지 수지식에 대한 가정은 다음과 같다.

- 각 영역에서의 온도는 서로 같다.
- 노내 온도는 용철의 온도와 같다.
- 총괄 에너지 수지는 다음과 같다.

다. 동적 시뮬레이션

(1) 수치 해법

본 공정모델식을 시뮬레이션하는데 있어서 반응계가 비선형 미분방정식을 포함하고 있는 계이고 해석학적 방법이 불가능하였으므로 4 계 Runge-Kutta 방법을 적용하였다. 이 연구에서 시뮬레이션을 수행하는 공정 모델식은 용선의 물질 수지식, 에너지 수지식, 각 성분 (탄소, 망간, 인)의 물질 수지식 등 총 7 개의 미분방정식으로 구성되어 있다. 그리고 이 식들간의 인자들이 서로 연결되어 있으므로 같은 시간대에 성분들의 함량 및 온도를 추정할 수 있도록 하기 위해 한 구간의 시간 간격을 이동알 때마다 Runge-Kutta 방법을 적용하여 7 개의 미분방정식들을 동시에 풀 수 있도록 하였다.

(2) 최적화

이 공정모델식에서 최적화를 위해 설정된 매개변수들은 반응 성분인 탄소, 인, 망간, 규소에 대하여 확산 계수와 반응계면적, 경막 두께의 함수로 나타나는 반응속도상수 4 개만을 설정하였고 에너지 수지식에서 고려하지않은 항돌은 기존의 여러 자료들을 참조하여 상수값으로 초기화하였다.

매개 변수들을 최적화하는데 있어서 전로 반응계는 비선형계이므로 이론함량값을 계산하는데 있어서 해석적 방법으로 적분하여 풀 수 없었으므로 각 성분들의 오차값의 제곱의 합으로 나타낸 목적함수를 수식적으로 설정할 수 없었기 때문에 최적화 기법으로 각 매개 변수에 단계값을 주어 변화한 목적 함수값과 전 단계의 목적 함수값을 비교하여 단계적으로 최적 매개 변수 값을 추적해 가는 Hooke and Jeeves pattern search 방법을 사용하였다. 이 연구에서 적용된 시뮬레이션의 흐름도는 그림 16 에 나타나 있다.

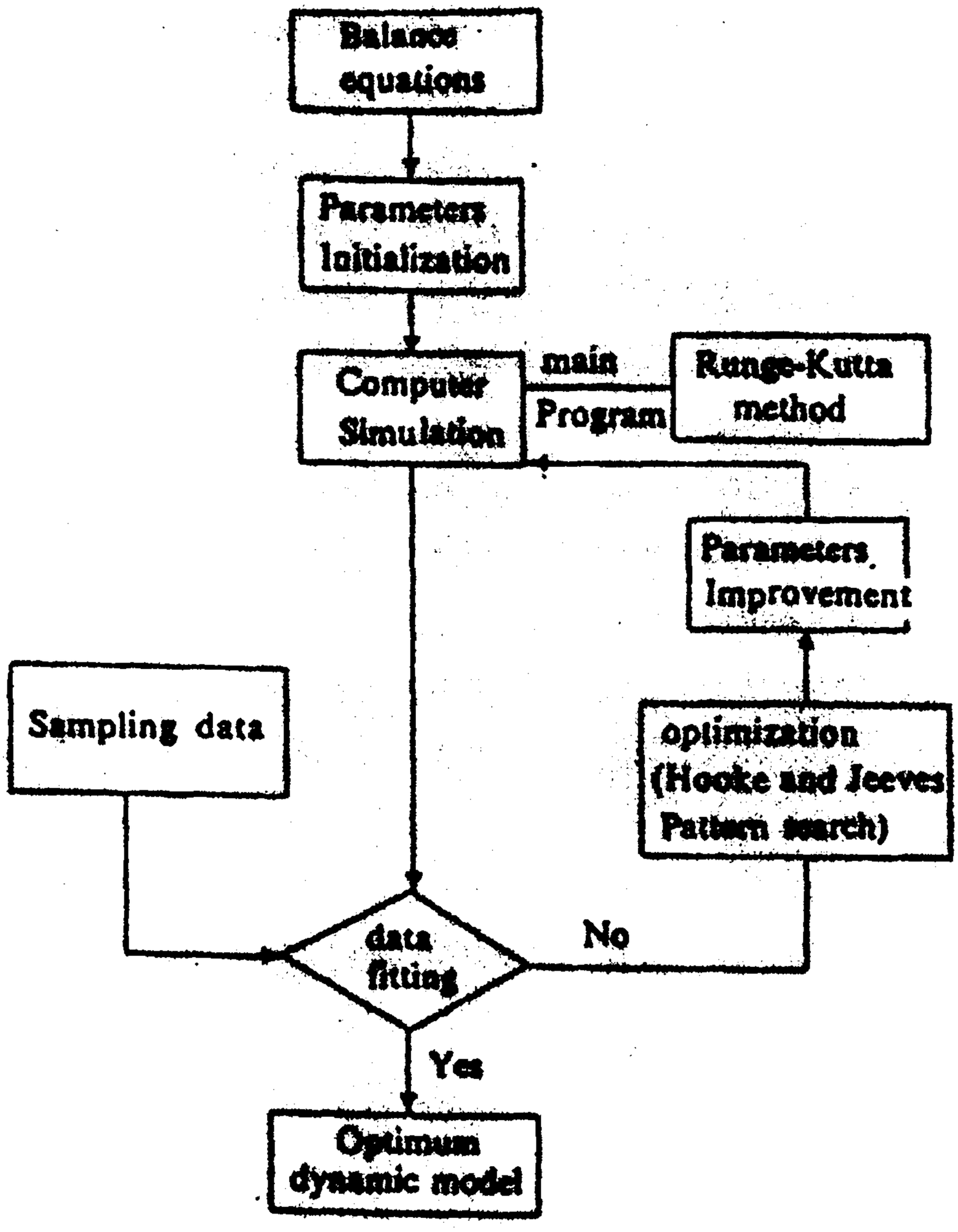


그림 16 : LD 공정을 위한 시뮬레이션 flow chart

설정한 공정모델식 (물질 수지, 에너지 수지)에 있어서 조업 조건과 초기 조건을 세우고 매개 변수들을 초기화하였다. 초기화한 매개 변수값은 탄소, 인, 망간, 규소의

반응 속도 상수값으로 각각 48, 0.018, 0.2948, 71 을 선정하였다. 단계값은 각각에 대하여 0.0025 를 취하였고 실제 조업시 채취한 데이타들은 탄소, 인, 망간, 규소의 질량백분율 값으로 이들과 같은 시간대에서 시뮬레이션에 의해 구한 이론값들의 차,즉 오차의 제곱의 합을 목적 함수로 놓아 데이타를 추적하였다.

8. 소성 가공 공정의 자동 유한요소 시뮬레이션

가. 문제점

금속 가동 공정 중 유한 요소법을 이용하여 소성 변형 문제를 해석할 때 나타나는 문제점 중의 하나는 요소의 뒤틀림이다. 즉, 변형이 진행되어갈 때 유한요소 체계중의 어떤 요소들이 심하게 뒤틀어진다는 것이다. 아러한 문제는 유한 요소 수식화가 재료의 변형에 따라 요소의 형상이 변하는 재료의 거동 방식에 기초할 때 항상 일어날 수 있다. 요소의 뒤틀림은 해의 정확성에 영향을 준다. 주목해야 할 것은 시뮬레이션중 잦은 격자의 재구성이 필요하다는 것이다. 특히 복잡한 형상의 가공 공정을 시뮬레이션 할 때에는 더욱 그러하다. 이 연구에서는 시뮬레이션 도중에 완전한 자동의 격자 재궁성을 하여, 공정 시뮬레이션을 처음부터 끝까지 자동적으로 수행하는 기능을 개발한다 [11].

현재까지 임의 형상의 물체에 대한 자동 격자재구성에 대한 많은 접근 방법이 제시되었지만, 그 중 격자법이 관심이 끈다. 왜냐하면, 이 방법은 소성 변형의 해석을 위해 가장 효율적이라고 알려진 사각형, 육각형 요소를 구성할 수 있기 때문이다. 이 연구에서 사용하는 격자법은 안내 격자라고 불리는 선정된 사각형의 안내에 따라 해석 영역에서 사각형 요소가 구성되며, 경계선이 통과하는 사각형들에 대해서만 요소 분할을 한다.

소성 가공중 재료가 겪는 변형 과정에 따라 변화하는 몇 가지 물리량이 있는데, 유효 변형률, 다공질 금속이 경우의 상대 밀도와 온도가 그것이다. 소성 가공공정의 시뮬레이션 과정에서 유효 변형률과 상대 밀도는 변형 과정에 따라 계속 새로운 값으로 계산되어 각 격자의 값으로 저장된다. 새로운 유한 요소 체계가 생성될 때마다 과정에 의존하는 물리량에 관한 계산이 필요하므로, 이러한 요소값의 이동에서 발생하는 오차는 새의 정확성에 영향을 미치므로 최소화되어야 하며, 잦은 격자 재구성이 필요한 시뮬레이션에서는 특히 그러하다.

나. 격자의 구성

해석 영역의 경계는 해석 영역의 경계상에 있는 이전 격자점에 의해 정의되거나, 이전 유한 요소 체계가 없을 때에는 사용자가 정의한 좌표계상의 점의 집합으로 정의된다. 해석 영역의 경계에 대한 정보가 주어지면 새로운 유한 요소 체계가 구성될 수 있다. 이 연구에서는 유한 요소 체계가 미리 정의된 직사각형들로 구성되었으며, 유한 요소의 구성에 사용된 사각형의 집합을 안내 격자라 한다.

격자 구성의 과정은 다음의 네 단계로 나뉠 수 있다.

1. IN, OUT, NIO 격자의 구분
2. 절점의 구성
3. 요소의 구성
4. 격자의 최적화

(1) IN, OUT, NIO 격자의 구분

안내 격자내의 사각형은 세 가지로 분류될 수 있는데, 해석 영역내의 격자를 IN, 해석 영역과 겹쳐지지 않는 격자를 OUT, 경계선을 통과하는 격자는 NIO라 한다. 이중에서 IN 격자와 NIO 격자가 격자 구성에 사용되며, OUT 격자는 격자 구성에 사용되지 않는다. 격자의 종류를 구분하는 데는 두 단계가 있다. 첫째로, 경계선과 안내선이 만나는 모든 점을 조사한다. 교점은 격자의 변 또는 모서리에 존재할 것이다. 만약, 교점이 모서리에 위치하면 그 모서리를 공유하는 네 격자는 NIO가 된다. 다음으로 격자의 어느 한 모서리가 해석 영역내에 있으면, IN 격자, 그렇지 않으면 OUT 격자가 된다.

(2) 절점의 구성

절점은 해석 영역의 경계선상에 있는 이전 절점에 형성되고, 또한 경계선과 안내선의 각 교점에도 형성된다. 이렇게 하면 경계선상의 절점이 구성되는데 격자의 재구성이 자주 수행되는 때에는 그 결과로 경계선상에 너무 많은 절점이 생길 수 있다. 따라서 다음의 방법으로 새로이 생긴 절점의 일부는 제거되어야 한다 : 잇달은 세 절점이 한 평면상에 있을 때, 가운데 있는 절점은 잉여 절점으로 간주되어 제거되어야 한다. 평면에 대한 판별 조건은 세 절점으로 이루어지는 각이 175도와 185도 사이에 있는 것이다. 이와같은 제거법에서 제외되는 것은 교점에 위치하는 절점이다. 왜냐하면, 절점이 교점상에 존재하지 않는다면 요소 구성에 있어서 어려움이 따르기 때문이다. 해석 영역내의 절점의 구성은 모든 IN 격자의 각 모서리에 절점을 형성함으로써 완성된다.

(3) 요소의 구성

IN 격자의 면적은 해석 영역에 속하는 사각형 요소로 채워진다. IN 격자의 각 모서

리에는 절점이 있으므로, 요소는 사각형 모양을 가지는 하나의 요소로 자동적으로 이동된다.반면, 각 NIO 사각형의 면적은 해석 영역에 속하는 요소로 부분적으로 채워진다. 그 요소는 반시계방향으로 경계선을 따라갈 때 좌측으로 존재한다. 특별한 경우, 요소가 사각형 또는 삼각형일 때 그 요소는 하나의 요소로 이동된다. 그러나 그 요소는 일반적이로 다각형이므로 여러개의 요소로 분해할 필요가 있다.

분해는 요소에서 사각형과 삼각형 요소를 떼어냄으로써 완성된다. 요소를 분해함으로써 얻어지는 요소의 수는 가능한 한 많은 수의 사각형을 분리해 낼 때 최소화된다는 것을 주목하며 다음의 방법이 채택되었다.

먼저 요소의 경계선상에 있는 절점에 연속적으로 번호를 매긴다. 루프를 형성한 후에 루프에서 네 개의 연속된 점집합을 검사한다. 만일 그러한 점집합이 사각형을 형성한다면, 사각형을 요소로부터 분리한다. 만일 루프에 그러한 점집합이 없다면 연속한 세 개의 점집합을 검사한다. 점집합이 삼각형을 형성한다면, 그 삼각형을 요소로부터 분리한다. 그 다음으로, 요소의 나머지 부분의 분해를 위해 두번째 루프를 구성한다. 이 과정을 요소의 나머지 부분이 사각형 또는 삼각형으로 될 때까지 계속한다. 분리하는 방법은 유일하게 존재하지 않으며, 사각형 또는 삼각형으로 분해하는 여러가지 방법이 존재할 수 있다. 여기서 다른 방법이 택해지면, 다른 조합이 형성될 수 있다. 여기서 여러가지 경우 중에서 어떤 것이 최적의 조합을 선택하는가에 대한 의문이 생긴다. 공정 시뮬레이션 중 함수 행렬식이 음으로 될 가능성을 줄이는 것에 초점을 둔다면 선정 조건을 선정할 수 있을 것이다. 함수 행력식이 음이될 가능성은 요소의 최대 내각의 크기가 감소할수록 줄어든다.

따라서 각 조합의 내각을 비교하여 가장 작은 경우의 조합을 선택한다. 이 조건을 적용했을 때, 둘 이상의 조합이 똑같이 조건에 부합될 수 있는데, 이 때는 예각을 조사하여 예각중 큰 것을 가진 조합을 선택한다.

(4) 격자의 최적화

격자 구성후 예각을 가지는 요소뿐 아니라, 크기가 작은 요소가 새로운 격자의 경계에 생길 수 있다. 삼각형이 대부분인 이 요소들은 계산 효율과 해의 정확성의 입장에서 바람직하지 않다. 그러한 요소의 제거가 격자의 간단한 조정에 의해 얻어지는 많은 경우가 있다. 아래에는 이러한 경우 및 이에 관계된 방법이 기술되어 있다.

예각을 가진 삼각형 요소 : 요소의 절점을 A, B, C 라고 하면, 절점 A, C 가 경계선에 있고 예각이 절점 A 에서 형성된다면, 해석 영역 내의 절점 B 를 경계 절점 B 로 이동시킨다. 예각이 절점 A 에서 형성되다면 경계선의 두 절점중 하나를 이동시킨다. 경계선상의 임의의 절점을 이동함에 따라 경계선이 바뀌게 되는데, 경계선의 변화가 최소화되어야 한다. 이는 평면상의 절점을 다른 절점으로 이동함으로써 달성된다. 2 단계에서 제시된 조건이 평면의 정의에 사용되었다.

크기가 작은 요소 : 두 개의 절점 B 와 C 가 경계선상에 존재할 때, 절점 A 는 절점 B 와 C 중 가까운 점으로 이동한다. 요소의 모든 절점이 경계선 상에 존재한다면, 가장 평평함 면상의 절점이 가장 가까운 다른 절점으로 이동한다.

다. 요소값을 변환

유한 요소 체계의 각 요소값을 새로운 유한 요소 체계의 요소값으로 이동시키는 방법은 다음의 3 단계의 방법에 의해서 이루어진다.

단계 1 : 이전 유한요소체계의 요소값을 동일 유한요소체계의 절점값으로 이동한다.

단계 2 : 이전 유한요소체계에서의 절점값을 결정하기위해 이동될 절점이 이전 유한

요소체계의 어느 요소에 포함되어 있는지를 찾는다.

단계 3 : 새로운 유한요소체계에서의 절점값을 동일 유한요소체계의 요소값으로 이동한다.

라. 시뮬레이션의 수행

계산의 재개에 앞서 요소값은 물론, 경계 조건도 이전 유한요소체계에서 새로운 유한요소체계로 이동되어야 한다. 새로운 유한요소체계의 경계선 상의 격자점 가운데에는, 이전 유한요소체계에서 옮겨진 격자점도 있고, 새로이 생성된 것도 있다. 이전 유한요소체계의 격자점에 대한 경계조건은 알려져 있는 것이 명백하고, 이는 새로이 생성된 격자점에 대한 경계조건을 결정하는데 충분하다.

이는 금형과 접촉하는 두 격자점 사이에 위치하는 격자점은 금형과 접촉하고 있어야 한다는 등의 간단단 조건을 사용하여 얻어질 수 있다.

유한 요소 체계 구성에 관계된 모든 방법과, 요소값의 이동, 경계 조건의 결정은 기존의 소성 가공 공정의 시뮬레이션을 위한 유한 요소 코드에 적용되었다. 이리하여 구성된 자동 시뮬레이션을 위해 필요한 일반적 입력 정보외에, 이 코드는 자동 시뮬레이션의 실행을 위해 다음의 입력 정보를 필요로 한다.

- 안내격자내의 각 요소의 좌표점
- 각 좌표점의 x 와 y 방향의 속도
- 다음의 요소 재구성을 수행하기까지 진행되어야 할 변형 단계의 수
- 유한 요소 체계 최적화를 위한 임계치

첫 번째 입력 정보는 안내격자의 형상을 결정한다. 해석 영역내에서 유한 요소 체계

의 밀도 분포가 적절하게 되도록 안내 격자의 설계에 촛점을 맞추면 시뮬레이션 과정에서의 해의 정확도와 계산 효율이 높아질 것이다. 두 번째 입력 정보는 안내 격자의 변형에 쓰인다. 이 작업은 시뮬레이션중 계속 변하는 해석 영역의 형상에 맞춰 안내 격자를 개정하도록 한다. 공간상에 고정되어지는 안내 격자와 비교하여볼 때, 변형하는 안내 격자는 아음에 보이겠지만, 계산 효율의 측면에서 잇점을 가진다. 세 번째 입력 정보는 시뮬레이션중의 요소 재구성의 횟수를 제어하는데 쓰인다. 이 정보에 의해 결정되는 변형 단계의 수의 간격에 맞추어 요소 재구성을 할 뿐만 아니라, 요소의 재구성은 함수 행렬식이 음이 되어도 실행된다. 이러한 시뮬레이션에 대한 flow chart 는 그림 17 과 같다.

9. 과학적 가시화와 실시간 시뮬레이션을 이용한 고로 내부의 가시화

현재 POSCO 에서는 기술 혁신의 일환으로 S/R, S/C project 등 새로운 공정 방법들에 대한 기술 투자가 이루어지고 있다. 그런데 아직 생산라인이 갖추어지지 않은 새로운 공법들에 대한 효율적인 연구를 위해서는 가상적인 공장 및 생산 설비를 실시간 시뮬레이션과 과학적 가시화를 사용하여 연구의 성과 및 파급 효과를 미리 예측하는 것이 필요하다.이렇게 함으로써 가장 이상적인 설비 및 기술을 개발할 수 있고, 위험 비용을 최소화하며, 보다 빠른 의사 결정을 할 수 있다. 본 절에서는 POSCO 의 신기술중의 하나인 고로에 철광석및 코크스를 공급하는 새로운 공정에 대한 실시간 시뮬레이션을 통한 과학적 가시화에 대해서 알아본다[4].

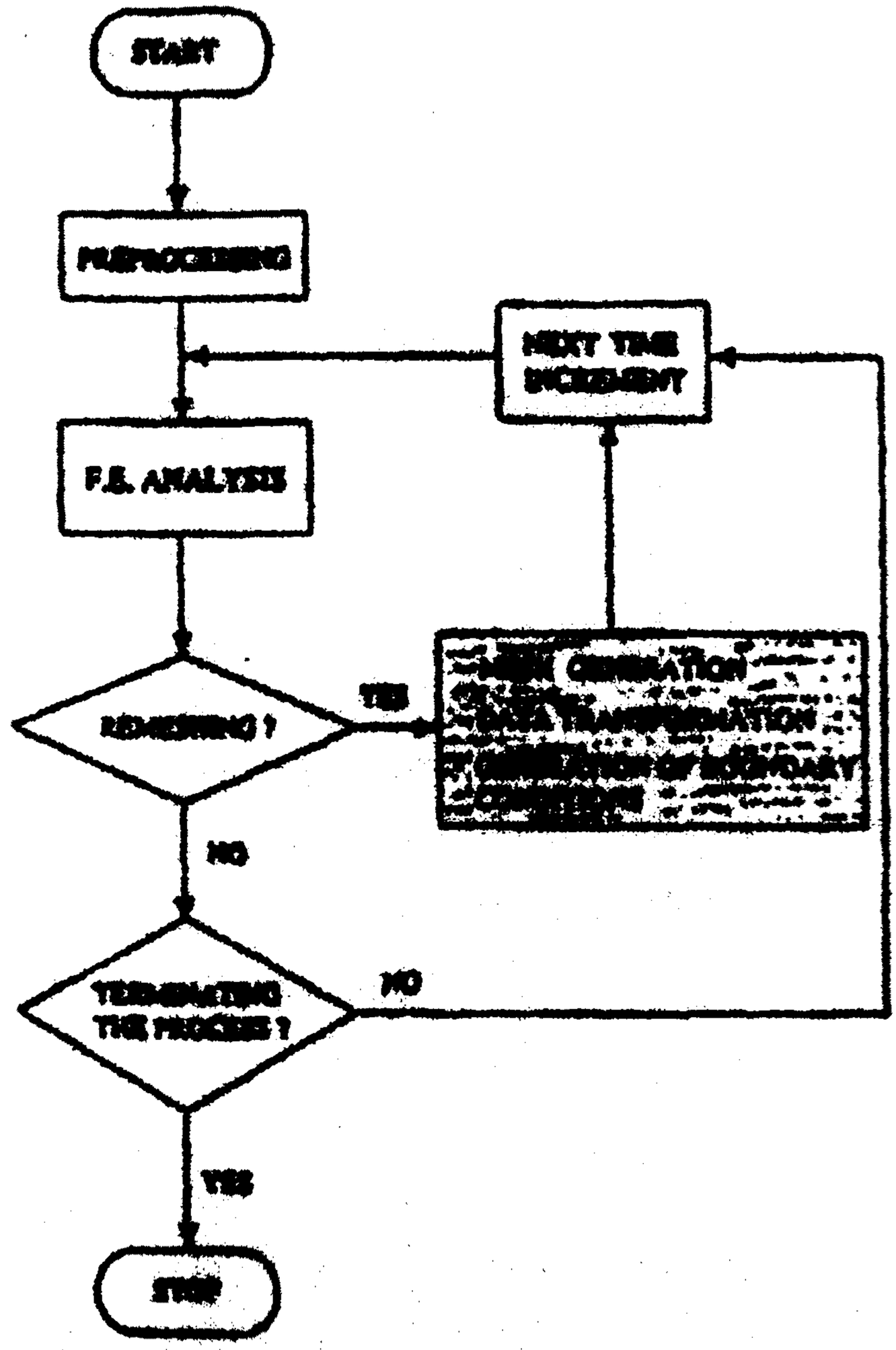


그림 17 : 자동 유한 요소 시뮬레이션을 위한 flow chart

가. 고로 내부의 실시간 시뮬레이션

고로와 같이 일단 작업이 시작되면 내부의 상태를 전혀 알 수 없는 경우에는 물리적 현상에 근거한 시뮬레이션을 수행해서 필요한 자료를 얻는다. 대부분의 경우에 시뮬레이션의 대상이 되는 시스템은 비선형 편미분방정식으로 나타난다. 연속 체게 시뮬레이션에서 사용되는 비선형 편미분방정식을 풀기 위해서는 수치해석학적인 방법을 사용하여 근사해를 구하는 것이 일반적이다. 고로의 경우 낙하하는 철광석 또는 코크스의 다음 순간의 위치가 현재 위치와 현재 속도, 가속도 등에 의해서 결정되는 상미분방정식으로 나타난다. 이러한 경우에는 Runge-Kutta 방법을 이용한 수치해석학적인 알고리즘을 사용할 수 있다. 그러나 고로 내부에서 한번 작업에 추적해야할 대상이 되는 철광석들이 대단히 많기 때문에 이들을 효율적으로 근사하여 실시간 시뮬레이션하는 것은 매우 성능이 좋은 컴퓨터를 사용하여도 상당한 시간을 요구하는 일이 될 것이다.

나. 고로 내부의 과학적 가시화

이러한 시뮬레이션을 통해서 얻어지는 자료로부터 현재의 상태를 표시하거나 특정한 대상체가 시공간에서 그리는 궤적, 또는 특정 시간의 공간의 벡터 필드 등을 표현하기 위해서는 다양한 모델링 및 렌더링 기법이 필요하다. 모든 자료를 특정 시간에 정확한 위치에 방향에서 그려주는 방법은 전통적인 그래픽스 기법으로 많은 연구가 되어왔다. 그러나 실험 자료가 방대한 시뮬레이션의 특성상 수억 또는 수십억 개의 polygon 을 정확히 빠른 시간에 그리기 위해서는 효율적인 모델링과 렌더링이 매우 중요하다. 본 시뮬레이션에서 얻은 데이타를 가시화하는 방법으로는 철광석과 코크스 가루를 particle system 을 이용한 애니메이션 기법을 이용하였다. 즉, 시뮬레이션을 통해서 주어진 시간에 위치가 계산된 각각의 가루들을 작은 입자로 모델링한 다음, 이 입자들의 위치가

시간마다 변하는 것을 1/24 초마다 렌더링한 다음 이를 연결해서 움직이는 모습을 나타내었다.

10. 압연기 제어 시뮬레이터

제철 공정에서 압연기는 다양한 두께의 강판을 제조하는 중요한 공정이다. 따라서 이 압연 공정을 제어하기 위해서는 숙련된 운전원이 필요한데, 이 운전원을 효과적으로 훈련시키는 것이 압연기 제어 시뮬레이터이다. 본 절에서는 포항제철에서 사용하는 압연기 제어 시스템인 PWS 와 PWS 에 대한 시뮬레이터인 EWS 를 중심으로 이러한 시뮬레이션 과정에 대해서 알아본다[5].

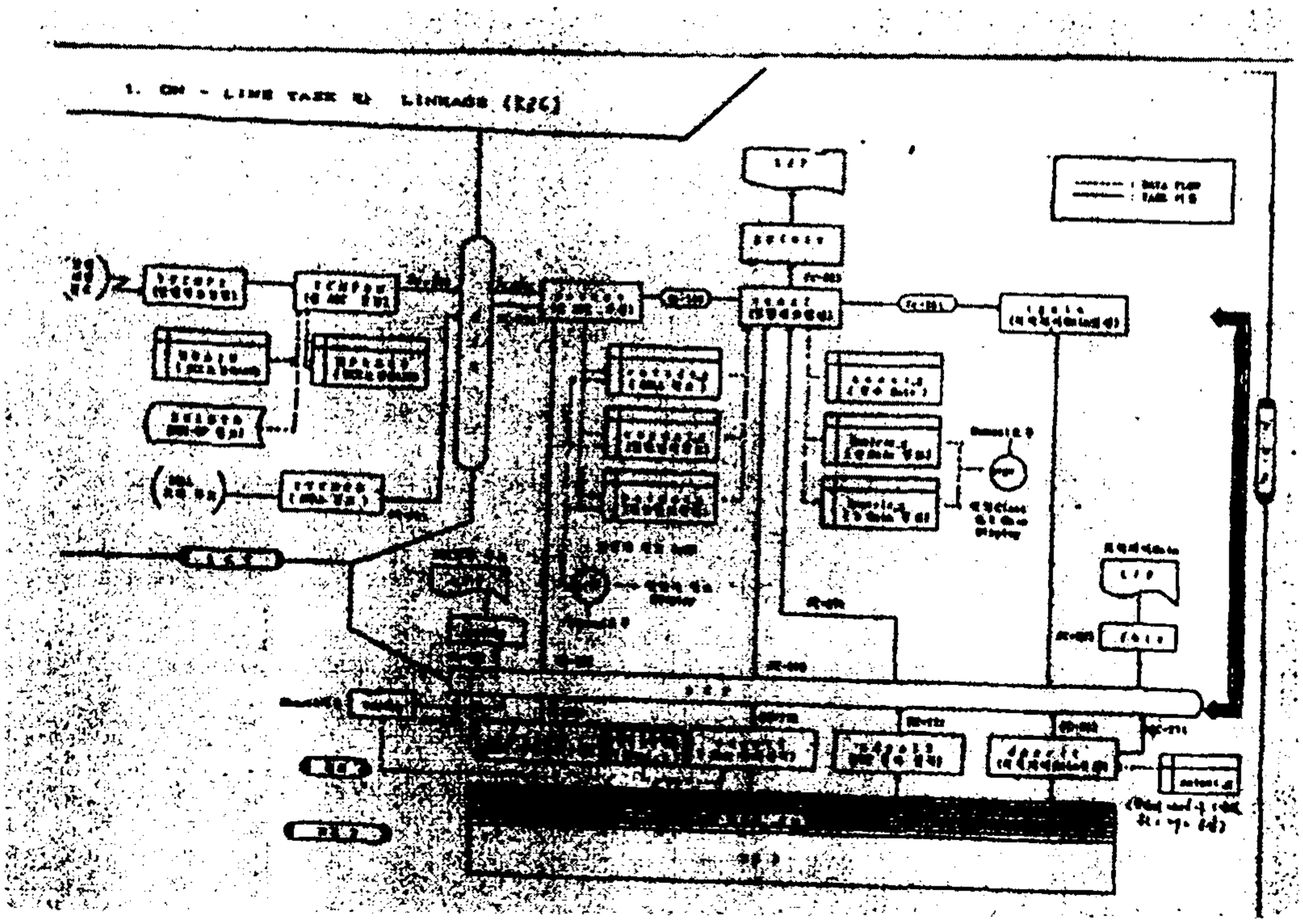


그림 18 : PWS 의 데이타 흐름도

가. PWS

PWS는 압연기 제어 시스템으로 운전원으로부터 압연할 강판의 두께, 넓이, 폭 등의 데이타를 입력 받아서 적당한 압연기의 roller의 속도와 roller의 간격 (gap)을 계산한다.그리고 이 계산된 데이타를 압연기를 제어하는 PLC (Programmable Logic Controller) 에 전송해서 압연기를 제어한다. 이 PWS의 각 모듈간의 데이타의 흐름은 그림 18에 나타나 있다.

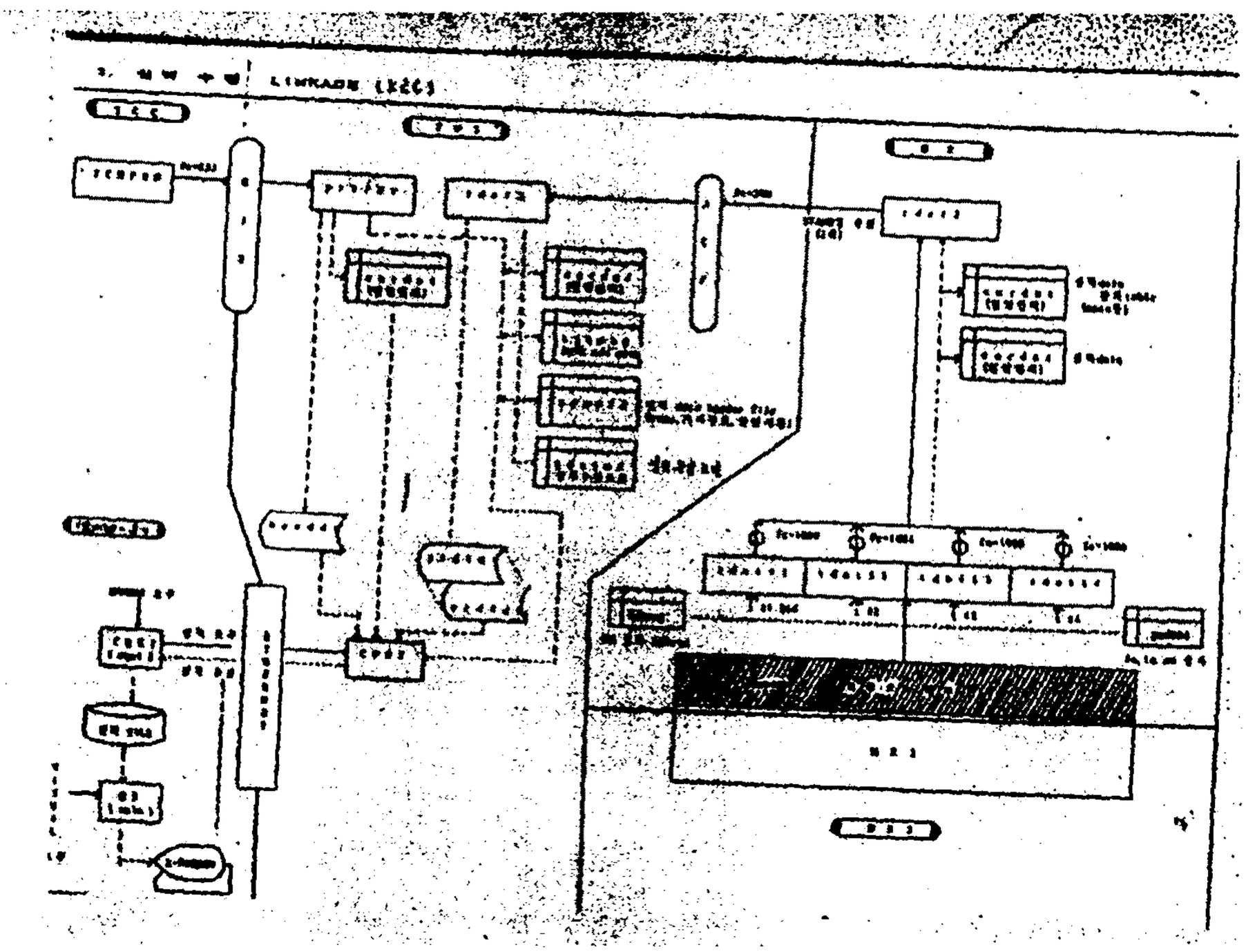


그림 19 : EWS와 PWS간의 데이타의 흐름도

나. EWS

EWS 는 PWS 에 대한 시뮬레이터로 ethernet 으로 연결되어서 서로 데이타를 주고 받는다. 즉, EWS 를 통해서 운전원이 압연 공정을 수행할 강판의 두께, 넓이, 폭 등의 데이타와 공정후의 강판의 두께 등을 입력하면 EWS 는 이 데이타들을 PWS 에 전송해서 압연기 제어에 필요한 PLC 제어용 데이타를 얻는다. 그리고 이 데이타를 가지고 압연 공정에 대한 시뮬레이션을 수행해서 그 결과를 X window 를 통해서 보여준다. EWS 와 PWS 간의 데이타의 흐름은 그림 19 에 나타나 있다.

11. 광양 제철소 냉연 PL/TCM 공정 제어 시스템 설계

본 절에서는 실시간 시뮬레이션을 이용한 광양 제철소의 냉연 PL/TCM 공정 제어 시스템의 설계에 대해서 알아본다[6].

기존의 집중형 구조로 되어있는 냉연 PL/TCM (Picking Line/Tandem Cold Mill) 공정 제어 소프트웨어를 분산형으로 재구성하는 경우 제안된 시스템의 성능을 미리 예측하는 것은 시스템 개발 단계에서 많은 노력을 절감시킴으로 중요하다고 할 수 있다. 여기서 냉연 PL/TCM 공정 제어 시스템의 성능의 적합성을 평가하는 방법으로 시뮬레이션을 이용하는데,이 시스템은 일종의 실시간 시스템으로 볼 수 있으므로, 실시간 시뮬레이션 기법이 이용된다.

현재의 냉연 PL/TCM 시스템은 공정 전체를 포괄적으로 지휘 관리하는 PC, 공정 라인, 그리고 공정라인의 현재 상태 수집 및 제어 데이타에 의한 공정제어를담당하는 PLC 및 계측기들, PC 로부터 자료를 인출하기 위한 사용자 터미널로 구성되어있다. PC 는 현장 라인의 소재의 현 위치를 추적하여 해당 작업을 수행할 수 있도록 제어한다.

공정라인은 크게 입측라인, 중앙라인(온라인), 출측라인 들로 구성된다. 입측라인은 열연 코일을 중앙라인의 시작인 POR 로 이동시키며 각종 기기의 자동설정을 수행한다. 중앙라인은 입측라인에서 받은 열연 코일을 용접하여 이어진 형태의 스트립(strip)으로 만든 후 연속적으로 압연을 수행한다. 출측라인은 압연이 완료된 스트립을 냉연 코일로 자른 후 각 코일의 무게를 측정하는등 실적을 수집한다.

냉연 PL/TCM 공정 제어 시스템을 시뮬레이션할 때에 고려해야 하는 실시간 요구사항은 주기적인 처리(실적 데이타의 sampling 과 트랙킹), 공정상의 순차적인 사건 처리, GUI 와 오퍼레이터에 관련된 것으로 구분할 수 있다. 그리고 각각 고려해야 하는 시간 제약 사항은 다음과 같다.

- 주기적인 처리에 대한 시간 제약 사항 : 다음 사건이 발생하기 전에 모든 주기적인 작업을 마쳐야 한다
- 공정상에서 차례로 발생되는 사건들 : 사건의 특성에 따라 각 사건들에게 부과되는 시간 제약 조건에 맞게 사건을 처리해야 한다.
- 사용자에게 보여지는 화면에 대한 시간 제약 조건 : 일정한 시간 이내에 화면이 갱신되어야 한다.

냉연 PL/TCM 공정 제어 시스템에 대한 시뮬레이터를 설계할 때에 위의 시간 제약 조건들은 다음과 같은 형태로 반영된다.

- 주기적인 처리를 위해서는 각 주기적인 사건의 마감 시간을 명기해두고, 이에 따라서 필요한 타스크를 주기적으로 기동시킨다.
- 발생하는 각 사건들에 대해서는 제어 시스템에 대해서 사건을 발생시키는 피제어 시스템에서 사건을 발생할 때에 작업에 대한 시간 제약 조건을 표시하게 하

고, 순차적인 사건을 발생시킬 수 있는 기능을 제공한다. 따라서 사건이 발생하였을 때에 알맞은 타스크를 실행시키는 기동 관리자에서 이러한 사항들을 고려해서 타스크를 수행하도록 한다.

- 사용자에게 보여지는 화면 갱신에 대한 시간 제약 사항은 화면은 처음 로드하는 경우 2 초내에 서비스를 제공하여야 하며, 5 초 주기로 화면을 갱신하는 경우는 1 초내에 서비스가 제공되어야 한다는 것으로 정의한다.

4 절 결론

이제까지 살펴본 바와 같이 실시간 시스템은 발전소, 공장, 교통, 통신 등 현재 산업의 많은 부분에서 핵심적인 역할을 수행하고 있다. 따라서 이러한 실시간 시스템을 시뮬레이션하는 실시간 시뮬레이션 기술에 대한 연구는 매우 중요하다. 그런데 실시간 시뮬레이션 연구는 이를 뒷받침하는 하드웨어 기술과 소프트웨어 기술의 발전이 없이는 불가능하다. 먼저 하드웨어 기술로는 현재의 비약적인 컴퓨터의 처리 속도의 발전에도 불구하고 많은 경우 실시간 시뮬레이션을 위해서는 특별히 설계된 병렬처리 컴퓨터가 요구되고 있다. 그리고 소프트웨어 기술로는 실시간 객체 모델링등을 비롯한 모델링 기법과 하드웨어의 병렬화에 따른 병렬 소프트웨어 기술이 요구되고 있다. 따라서 실시간 시뮬레이션 기술의 발전을 위해서는 이러한 요소 기술의 연구가 선행되어야 한다.

참 고 문 헌

[1] 박 찬모, 민 경하,``실시간 시뮬레이션", 정보과학회지 제 13 권, 제 4 호, 1995.

[2] 이 칠기, ``실시간 모의제어 시스템 (Simulator) 국산화 개발", 정보과학회지 제 13 권, 제 4 호, 1995.

[3] 함 창식 외, ``교육훈련용 Nuclear Simulator 개발", 한국 에너지 연구소 계측제어 연구실, 1988.

[4] 이 인권, 최 정주, ``실시간 시뮬레이션을 이용한 고로 내부의 과학적 가시화", 포항공대 컴퓨터 그래픽스 연구실, 1994.

[5] ``압연기 제어 시스템 매뉴얼", POSCO 조업 매뉴얼, 1994.

[6] 박 찬익, ``광양 제철소 냉연 PL/TCM 공정 제어 시스템 설계", 포항공대 System Software 연구실, 1993.

[7] 김 광수, ``스테인레스 스틸 생산 공정의 시뮬레이션 연구'', POSCO 산업과기연 연구 보고서, 1990.

[8] 전 만수, 박 재성, 황 상무, `` 압연공정의 시뮬레이션 I : 유동 해석'', POSCO 산업과기연 연구 보고서. 1990.

[9] 이 덕만, 이 진수, `` 범용 로봇 제어 언어를 이용한 그래픽 시뮬레이션 '', POSCO 산업과기연 연구 보고서, 1991.

[10] 장 근수, 이 인범, 윤 상엽, 정 호철, `` 전로 반응 공정의 동적 해석 및 모사 '',

POSCO 산업과기연 연구 보고서, 1991

[11] 고 영우, 류 성룡, 황 상무, ``소성 가공 공정의 자동 유한 요소 시뮬레이션에 관한 연구 '', POSCO 산업과기연 연구 보고서, 1993.

[12] 문 상룡, 이 관희, 김 명수, `` 직접 구동 로봇트의 그래픽 시뮬레이션 시스템 개발 '', POSCO 산업과기연 연구 보고서, 1992.

여 백

제 2 장 거시적 실시간 시뮬레이터 구축 기술 개발

1 절 서론

컴퓨터 하드웨어의 성능이 급속하게 발전함에 따라 컴퓨터의 응용분야 역시 그 영역을 빠르게 넓혀 가고 있다. 컴퓨터의 고성능화로 가능해진 응용 분야는 일반적으로 대규모의 행동이 복잡한 실시간 컴퓨터 시스템을 요구하고 있다.

실시간 시스템이란 시스템의 행동의 올바름이 기능적인 측면뿐만 아니라 시간적인 측면에 의해서도 결정되는 시스템으로 원자력 발전소 제어 시스템, 기상 인공위성 제어 시스템, 미사일 제어 시스템, 그리고 교통 정보 시스템 등이 이에 속한다. 이런 실시간 시스템은 그 규모가 방대하고, 행동이 복잡하며, 시간적 제약이 엄격한 특징을 가진다. 이러한 복잡한 행동 양식을 보이는 실시간 시스템의 개발에 있어서, 사용자 요구사항의 분석은 매우 어려울 뿐만 아니라 시스템 개발 완료 시까지는 충분한 유효성 검사가 힘든 실정이다.

요구사항의 명세에는 두 일단이 참여하게 된다. 하나는 시스템을 사용하는 단말 사용자이고 다른 하나는 시스템을 분석하는 분석자이다. 일반적으로 단말 사용자는 컴퓨터 시스템에 대한 지식이 없고 분석자는 개발될 특정 시스템에 대한 기반 지식이 없기 때문에 요구사항 명세(Requirement Specification)는 비정형적이거나 모호하거나 불완전한 경우가 많다. 이러한 문제점을 해결하기 위해서 정형적인 방법(Formal Method)이 사용되지만 단말 사용자는 정형적인 명세서(Formal Specification)를 이해하기 힘들기 때문에 요구사항 명세서의 유효성 검사를 하기가 힘들다. 만약 요구사항 명세서의 오류가 소프트웨어 생명주기 후반기에 발견된다면 이 오류를 초기에 고치는 것에 비해 비용이

수백 배에 이른다는 것이 잘 알려져 있다.

사용자 요구사항의 유효성 검사를 위한 방법은 크게 3 가지 부류로 나뉘어 진다. 즉, 정형적인 방법, 명세의 수행과 시뮬레이션, 프로토타이핑 등이 있다. 이들 각각에 대해 아래에서 언급한다.

정형적인 방법에서는 사용자 요구사항이 수학적 이론과 방법을 적용하여 정형적으로 명세화된 후에 안전성, 일치성과 완전성과 같은 시스템 성질이 수학적으로 검증된다. 그러므로 정형적인 방법은 시스템의 올바름(Correctness)를 보장할 수 있다.

시스템이 복잡해지고 실시간 성질을 가짐에 따라, 분석되어야 할 시간적 공간적 범위가 극도로 증가하게 된다. 그러므로 이러한 방법을 전체 시스템 명세에 적용하기에는 비실용적인 면이 있다. 또한 이러한 방법은 최종 시스템이 어떠한 모습으로 행동하는지를 보고 느낄 수 있는 방법을 제공하지 못하기 때문에 사용자로 부터 시스템의 유효성을 검증 받는 방법이라기 보다는 시스템의 분석에 더 적합한 방법이다.

시스템 명세를 수행시키기 위한 많은 노력이 있어 왔고 많은 명세 수행 방법이 제안되었다. 이 방법은 사용자의 유효성 검사를 위해 시뮬레이터를 이용하여 명세를 해석하고 시스템 행동을 시각적으로 보여 준다. 또 실시간 시스템을 위해 추계적인 모델을 사용하여 실시간 시스템의 행동을 명세하고 시뮬레이션하고 유효성을 검증하기도 한다.

이러한 방법의 몇 가지 장점은 다음과 같다.

- 시스템의 모습을 보고 느낄 수 있고 시스템의 설계와 구현 전에 사용자로 부터 빠른 검증을 받을 수 있다는 것이다. 이러한 방법을 사용하면 사용자가 요구사

항 명세 단계에 적극적으로 참여하여 요구사항 추출과 검증 사이의 간격을 극도로 좁힐 수 있고 많은 선택 사항이 경제적으로 평가될 수 있다.

- 정형적인 분석 방법으로는 적합하지 않은 분야를 보완할 수 있다. 예를 들면, 실시간 요구사항의 행동적 민감성 테스트는 정형적인 방법으로는 불가능하다. 즉 명세를 수행시켜 전체 시스템 행동을 시뮬레이션 함으로써 실시간 요구사항에 민감한 기능자를 쉽게 찾아낼 수 있다.

하지만 명세의 수행 방법은 일반적으로 명세에 대해 올바름의 확신도를 높여 주는 역할은 하지만 이를 보장하지는 않는다. 시스템의 어떤 성질의 올바름을 보장하기 위해서는 가능한 모든 경우에 대해 검사를 해봐야 한다. 이것은 시스템의 크기가 커다란 경우에는 거의 불가능하다. 그러므로 정형적인 분석 방법과 명세의 수행을 통한 방법 모두가 요구사항 분석을 위해 필수적이다.

하지만 정형적인 방법과 명세의 수행 방법 모두를 제공하는 방법은 그리 많지 않다. ASADAL(A System Analysis and Design Aid tooL system)은 이들 두 가지 방법을 모두 지원하는 CASE 도구로 개발 중이다. 이 CASE 도구의 특징을 요약하면 다음과 같다.

- 위험 요소가 있는 시스템 성질- 안전성, 생존성, 실시간 반응성-의 검증을 위해 정형적인 분석 방법을 제공한다.
- 명세의 수행을 통해 생명 주기 초기 단계에 요구사항의 유효성 검사 방법을 제공한다.

본 보고서는 ASADAL 의 명세 수행 방법(시뮬레이션)을 통한 분석 방법에 초점을 두고 설명할 것이다.

2 절 ASADAL 방법론의 개요

1. 프로세스 행위

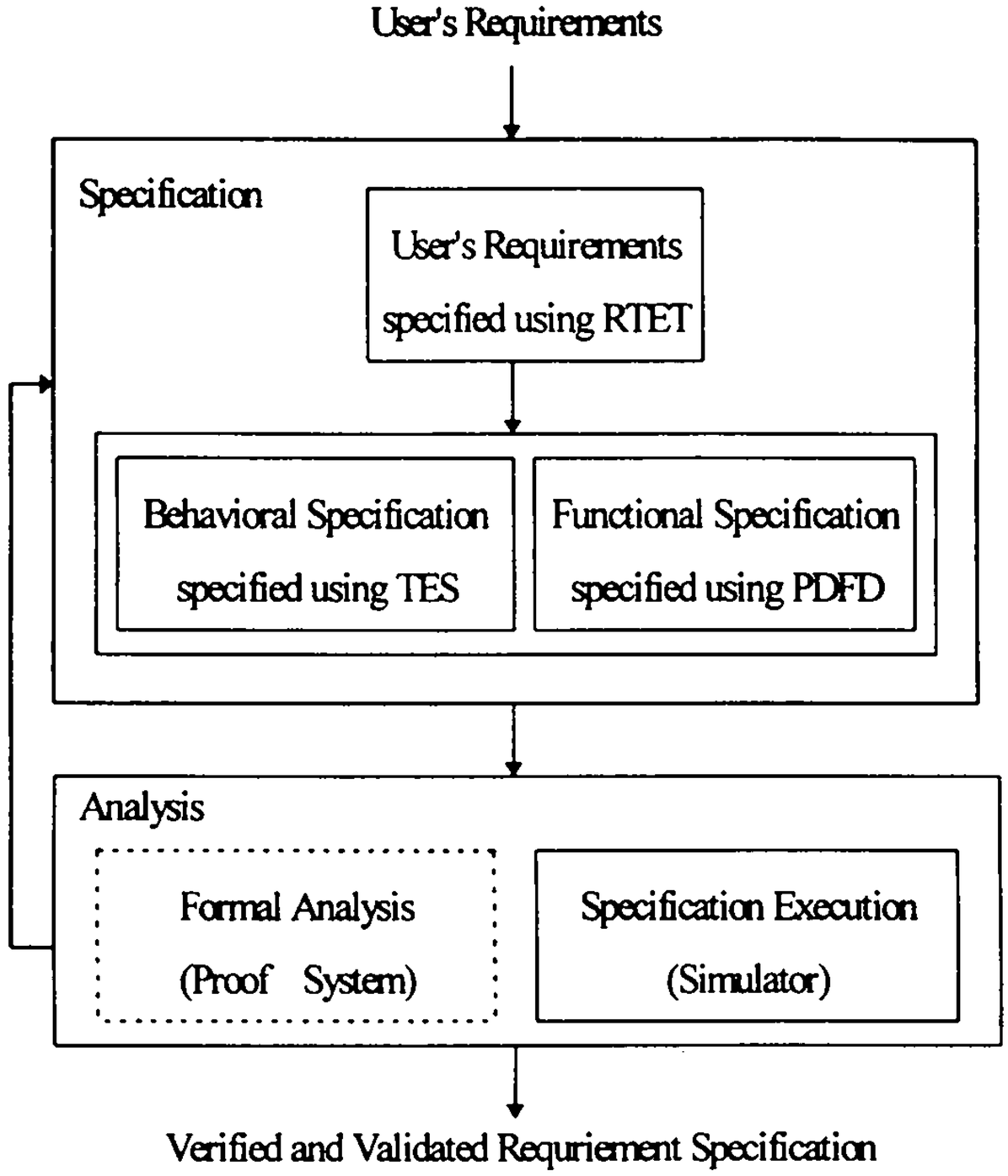


그림 1. ASADAL 의 전체 프로세스

ASADAL 방법론의 전체 프로세스는 그림 1 에 나타난 바와 같이 명세(Specification)와 분석(Analysis)으로 크게 나뉘어 진다. 이들 각각에 대한 내용은 다음과 같다.

요구사항 명세(Requirement Specification)는 사용자의 요구를 만족하는 시스템 모델을

만드는 데 목적이 있다. 이러한 모델을 만들기 위해서 시스템 분석가는 사용자의 요구사항을 찾아 내어 여러 가능성 있는 모델을 만든 후에 이들이 사용자의 요구사항을 만족하는 지를 검증하게 된다. ASADAL 에서는 사용자 요구사항을 명세화하는 방법과 더불어 시스템 모델을 명세화하는 방법을 제공한다. 사용자 요구사항은 RTET(Real Time Event Trace)다이어그램을 사용하여 시나리오 중심으로 명세화한다. 이러한 사용자 요구사항을 바탕으로 시스템 모델을 TES(Time Enriched Statechart)와 PDFD(PARTS Data Flow Diagram)을 이용하여 명세화하고 상세화한다.

명세의 분석에서는 시스템 모델이 일관적이고 정확하고 사용자 요구사항을 모두 만족하는 지를 분석하게 된다. ASADAL 에서는 명세의 분석 방법으로 정형적인 방법(Formal Method)과 명세의 수행 방법(Specification Execution Method) 두 가지를 제공한다. 시스템의 안전성과 같은 위험 요소가 내재된 중요한 요구사항은 정형적인 분석 방법을 통해 분석한다. ASADAL 에서는 시제 논리(Temporal Logic)을 기반으로 한 증명 시스템을 정형적인 분석 방법에 사용한다. 정형적인 분석 방법은 시스템의 행동의 모든 가능성에 대해서 분석하게 되므로 비용이 많이 들게 된다. 그러므로 위험 요소가 덜한 요구사항에 대해서는 명세의 수행 방법을 통해 최종 결과물에 대한 모습을 보고 느낄 수 있어 시스템 개발 초기 단계에 단말 사용자에게 유효성을 검사 받을 수 있어야 한다. ASADAL 에서는 시스템의 시간적, 기능적 행동을 모델링하고 시뮬레이션을 할 수 있는 시뮬레이션 언어와 함께 시뮬레이션 도구를 제공함으로써 사용자에게 시스템의 유효성 검사를 초기 단계에 받는다.

분석된 결과는 명세(Specification)가 다음 단계로 넘어가기에 필요한 모든 정보를 담고 있는가를 결정하는 데 사용된다. 명세가 완전하지 않으면 좀 더 자세히 상세화되고 분석된다. 이러한 과정은 모든 요구사항이 완전히 명세화될 때까지 계속 진행된다.

2. 명세서 수행에 대한 접근 방안

ASADAL 은 소프트웨어 생명주기 초기단계에 명세를 수행시켜 봄으로써 명세의 유효성을 검증하는 방법을 제공한다. 명세의 수행은 ASADAL 시뮬레[이터에 의해서 가능해진다. 시뮬레이터는 그림 2 에서 보는 바와 같이 시뮬레이션 도구와 자료 분석기로 나뉘어진다.

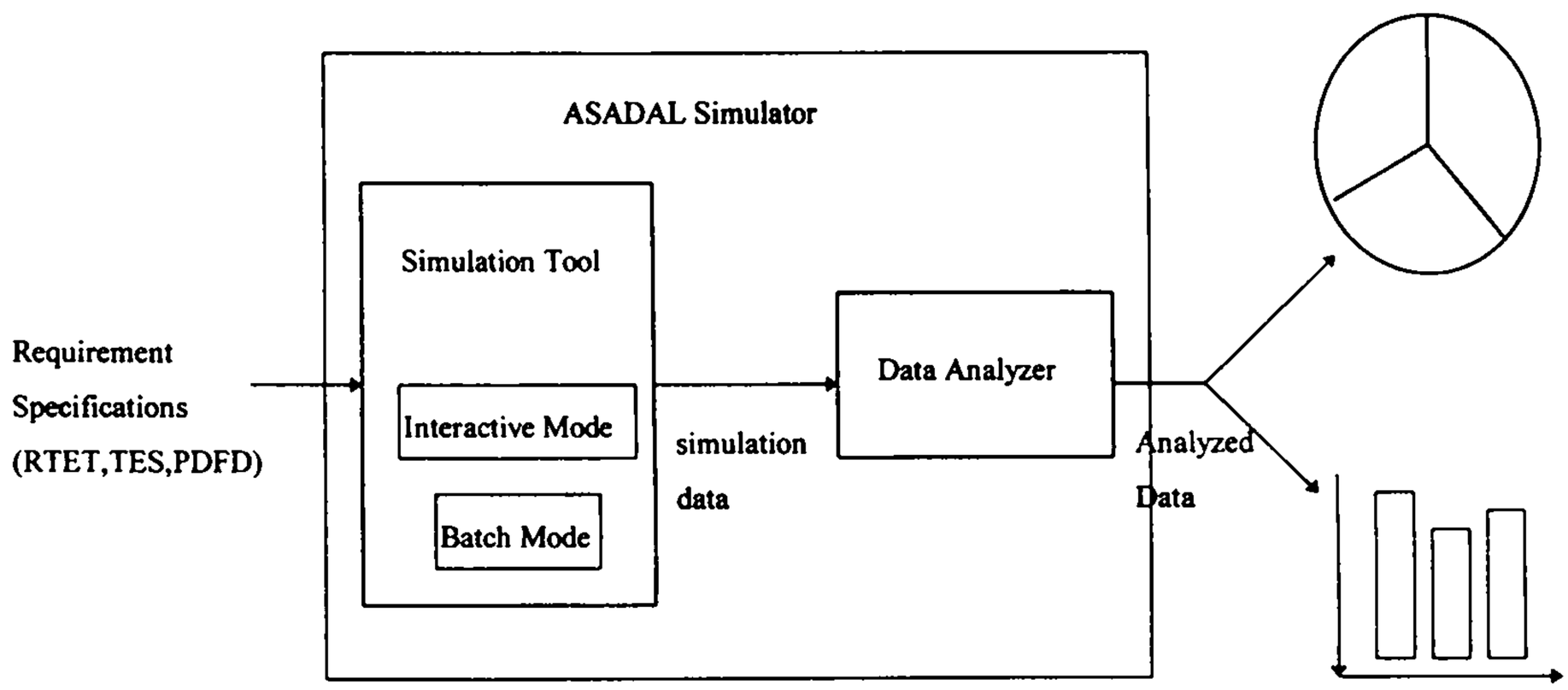


그림 2. ASADAL Simulator 구조

시뮬레이션 도구는 대화형 모드와 배치 모드 두 가지 모드로 명세를 수행시킬 수 있다. 대화형 시뮬레이션은 분석자가 명세의 행동을 단계별로 관찰하고 싶을 때 유용하다. 사용자는 외부 데이타를 발생시키거나 예기치 않은 상황을 맞이 하였을 때 시스템 환경 변수의 값을 바꾼다던가 명세의 잘못된 부분을 디버깅할 수 있다. 배치 시뮬레이션은 대량의 데이타를 다루는 시나리오 중심의 시뮬레이션이나 실시간 시스템의 추계적(Stochastic) 행동을 시뮬레이션할 때 유용하다. 배치 모드 시뮬레이션에서는 시뮬레이션 드라이버(Simulation Driver)가 외부 데이타를 발생시키거나 시뮬레이션 도중에 발

생하는 예기치 않은 상황을 대처하는 역할을 담당하게 된다. 이러한 시뮬레이션 드라이버의 역할은 시뮬레이션 언어를 통해 정의된다. 시뮬레이션 언어에 대해서는 제 4 장에서 언급하겠다.

자료 분석기는 시뮬레이션 도중에 시뮬레이션 자료를 모와서 분석된 자료를 출력한다. 분석된 자료는 통계 패키지를 사용하여 시각적으로 다양하게 보여지게 된다.

명세의 수행을 통해 시스템 행동을 여러 각도로 분석해 볼 수 있다. 명세의 수행을 통한 분석은 두 가지 관점에서 이루워진다. 하나는 사용자 요구사항 명세의 수행을 통해 사용자의 요구사항이 만족되는가를 살펴보는 것이고, 다른 하나는 시스템 모델을 수행시켜 봄으로써 시스템의 내부 행동이 원하는 방식 대로 이루어지는 가를 살펴보는 것이다.

다음 질문들은 사용자 요구사항 명세를 수행시킬 때 내릴 수 있는 질문들이다.

- 외부 자극에 대한 기대한 결과가 제시간에 나오는가?

  사용자가 시스템 행동을 이해하는 방식은 외부 자극에 대한 시스템의 반응을 관찰하는 식이다. 그러므로 ASADAL 에서는 사용자 요구사항 명세서에 외부 자극에 대한 내부 시스템의 반응시간을 나타내는 실시간 요구사항을 기술할 수 있게 한다. 이에 대한 자세한 설명은 3.1 절에서 다룬다. 그러므로 명세의 시뮬레이션 중에 외부 자극에 대한 반응이 실시간 요구사항을 만족하는지를 살펴보아 요구사항이 만족되지 않으면 내부 시스템 명세를 좀더 자세히 분석해 보아야 한다.

- 외부 자극이 수행 시나리오대로 시스템을 통해 전달되는가?

사용자 요구사항은 외부 자극 사건과 이에 대한 시스템의 반응 사건을 시나리오 관점으로 나타낸 것이다. 그러므로 외부 자극에 따라 정해진 시나리오 대로 시스템이 행동하는 가를 살펴봄으로써 사용자의 요구사항이 만족되는 지를 검사할 수 있다. 물론 시나리오가 모든 시스템의 행동을 표현할 수 없으므로 사용자가 관심 있어 하는 시나리오를 기준으로 많은 시나리오를 검사해 보아야 한다.

다음 질문들은 시스템 모델을 수행시켜 볼 때 평가해 볼 질문들이다.

- 시스템의 기능이 원하는 방식대로 수행되는가?

  하나의 작업을 하기 위해서는 시스템의 여러 기본 기능들이 순차적, 혹은 병행적으로 수행됨으로써 가능하다. 그러므로 어떠한 작업을 수행하기 위해 필요한 기능들이 원하는 순서대로 수행되는가를 살펴봄으로써 시스템 모델의 오류를 발견해 낼 수 있다.

- 시스템 기능이 주어진 입력에 대해 원하는 출력을 내는가?

  시스템 기능은 입력을 받아 이를 처리하여 처리된 결과를 내는 방식으로 행동한다. 그러므로 전체 시스템 기능이 제대로 동작하는 지를 살펴보기 위해서는 먼저 세부적인 시스템 기능이 주어진 입력에 대해 원하는 출력을 내는가를 살펴보는 것이 필요하다.

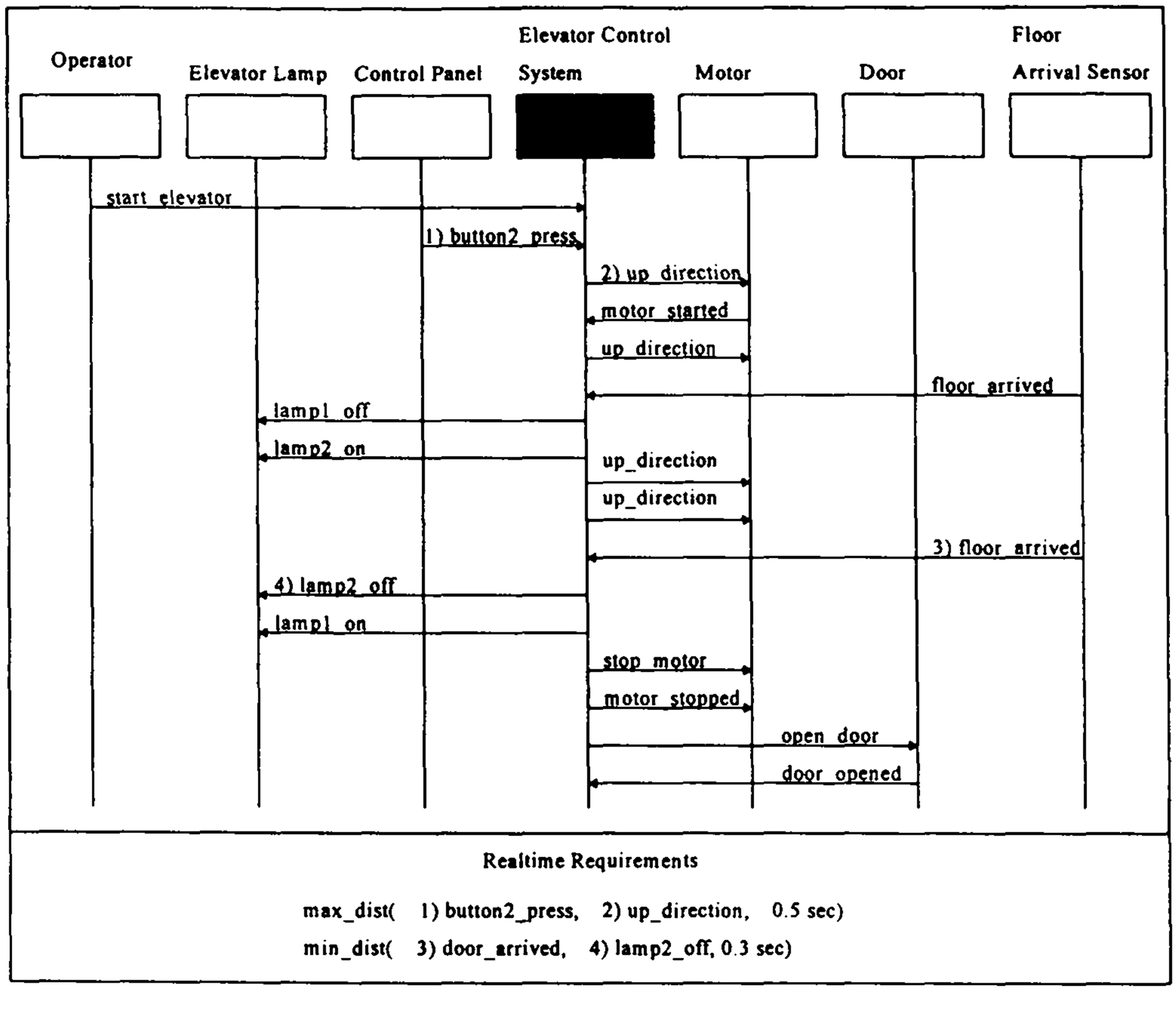


그림 3. 엘리베이터 제어 시스템을 위한 RTET 다이어그램.

- **실시간 요구사항을 만족시키기 위해 어느 기능이 가장 민감하게 반응하는가?**

  실시간 요구사항은 하나의 작업을 수행하는데 필요한 시간을 나타낸 것이다. 그러므로 이 실시간 요구사항을 만족시키기 위해서는 작업에 필요한 기능들의 수행 시간을 지정할 필요가 있다. 이때 어떠한 기능이 빈번히 수행되어 이 기능의 수행 시간이 실시간 요구사항의 만족성 여부를 결정짓는데 중요한 역할을 한다면 이 기능의 수행 시간을 신중히 결정할 필요가 있다.

이 보고서의 전체 구성은 다음과 같다. 제 3 장에서는 사용자 요구사항을 받아들여

이를 명세하고 시스템 모델을 만드는 ASADAL 의 명세 방법에 대해서 언급한다. 시스템 모델이 만들어지면 모델이 사용자 요구사항을 만족하는지를 분석하는 시뮬레이션 방법을 제 4 장에서 설명한다. 제 4 장에서는 시뮬레이션의 기반 개념과 실시간 시스템의 시간적, 기능적 행동을 명세하고 시뮬레이션할 수 있도록 고안된 시뮬레이션 언어에 대해 자세히 다룬다. 제 5 장에는 ASADAL CASE 도구의 기능상, 구조상, 그리고 설계상 특징에 대해서 설명한다. 끝으로 ASADAL 방법론에 대한 요약과 장점, 그리고 앞으로의 할 일에 대해 언급한다.

이 보고서에서 사용되는 모든 예제는 엘리베이터 제어 시스템을 다룬다.

3 절 명세 방법

이장에서는 사용자의 요구사항을 명세화하는 방법과 시스템 모델을 명세화하는 방법 두 가지를 제공하는 ASADAL 의 명세 방법에 대해 언급한다.

1. 사용자 요구사항 명세

사용자 요구사항 명세는 RTET 를 사용한다. RTET 은 시스템과 외부 개체 사이의 사건 흐름을 시나리오 관점에서 명세화한 사용자 관점의 명세화 언어이다. RTET 는 Rumbaugh 의 '사건 추적(Event Trace)'을 기반으로 하는데 대규모의 실시간 시스템을 다루기 위한 몇 개의 특징들을 추가적으로 제공한다.

실시간 시스템은 보통 많은 사건들이 관여된 사건 유도적(Event Driven)시스템이기 때문에 사용자들은 자주 자극 사건에 대한 반응 사건을 알아보는 형식으로 시스템의 행동을 이해한다. 그리고 시나리오의 생성을 통해서 사용자와 대화하고 사용자의 요구

를 명세하는 것이 매우 효과적이다.

그림 3 은 엘리베이터 제어 시스템의 외부 행동 시나리오를 나타낸 RTET 다이어그램이다. 그림 3 에서 Elevator Control System 은 개발할 소프트웨어 시스템을 나타내고 이것은 다시 세부적인 시스템 구성 요소로 나뉘어 질 수 있다. 각각의 구성 요소에 대해 하나의 시스템 모델을 구성하게 된다. 그림 3 에 나타난 시나리오는 다음과 같다.

오퍼레이터가 start 신호를 줌으로써 사건 start_elevator 가 발생하면 Elevator Control System 은 작동을 시작하게 된다. 이때, Control Panel 로 부터 button2 신호가 들어오게 되면 Elevator Control System 은 Motor 에게 위로 움직이게 하고 이에, Motor 는 작동을 시작했다는 반응을 보이게 된다. 엘리베이터가 한 층씩 올라감에 따라 floor_arrived 사건이 발생하고 이에 따라 Elevator Control System 은 Lamp 를 끄고 켜는 제어를 하게 된다. 엘리베이터가 원하는 층으로 움직이다가 요청된 층에 도달하게 되면 Motor 에게 stop 신호를 보내게 된다. 이에 Motor 는 동작을 멈추고 멈추었다는 신호를 반송하게 된다. 엘리베이터가 완전히 멈추게 되면 Elevator Control System 은 Door 에게 문을 열라는 신호를 주기적으로 보내 주게 된다. 문이 완전히 열리면 Door 는 문이 완전히 열렸다는 door_opened 사건을 알려 주고 이에 Elevator Control System 은 Door 에게 문을 닫으라는 신호를 보내 주게 된다. 이때, 위급 상황이 발생하여 emergency_occur 사건이 발생하게 되면 엘리베이터는 작동을 멈추게 되고 오퍼레이터가 restart 신호에 의해 작동을 재계할 때까지 기다린다. 작동이 재계되면 이전에 수행 중이던 문을 닫는 동작을 수행하게 된다.

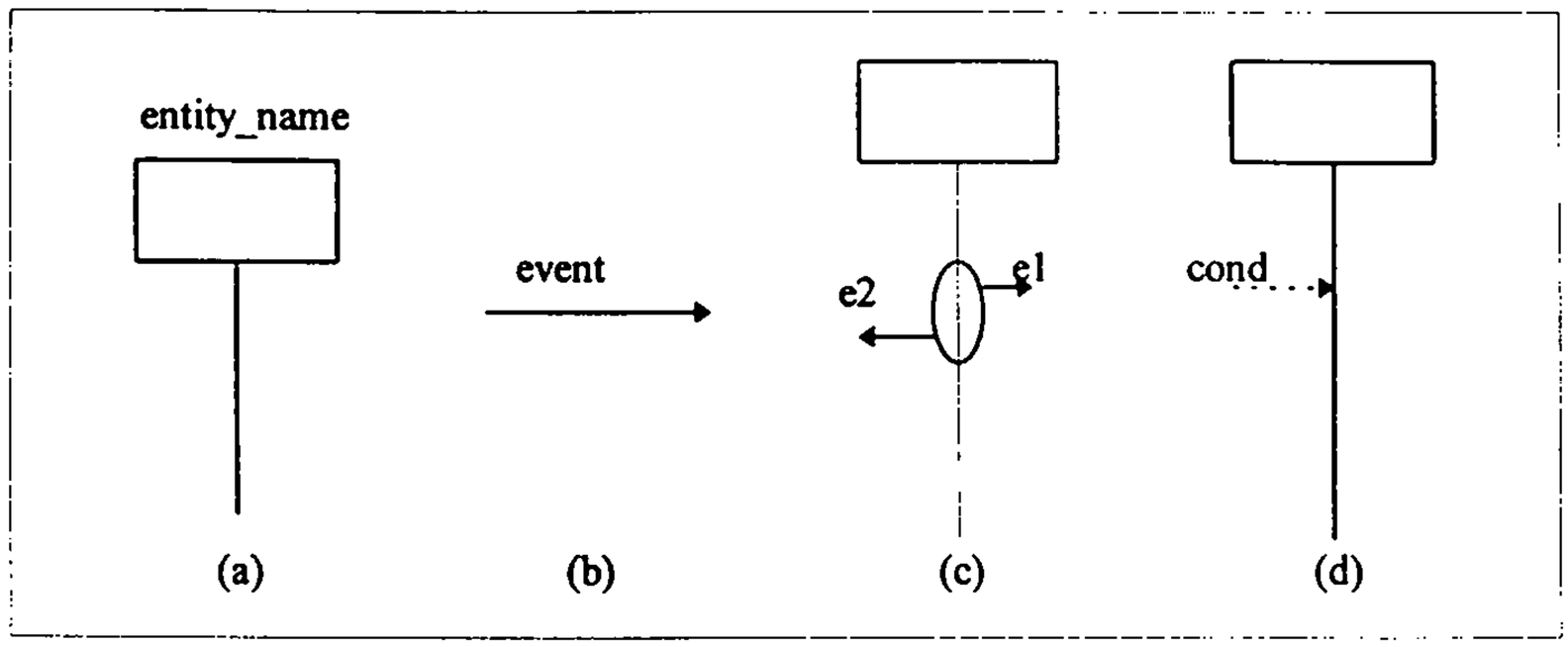


그림 4. RTET 의 기본 구조.

그림 4 는 RTET 의 기본 구조를 보여 준다. (a),(b),(c), 그리고 (d)는 각각 개체, 사건 전달자, 동시 사건 그리고 조건 행동을 나타내는 기본자이다. 이들 각각에 대한 설명 및 RTET 특징은 다음과 같다.

- RTET 명세의 하향식 상세화(Refinement)는 개체를 통해서 이루워진다. 개체는 개체를 이루는 여러 딸림 개체(Sub Entity)들과 그 딸림개체들 간에 전달되는 사건들로 상세화된다.
- 개체간에 흐르는 정보는 사건을 나타낸다. 사건이란 어떠한 일이 일어 났다는 것을 나타내는 신호이므로 어떠한 값을 전달한다기 보다 사건이 발생했다는 사실만을 나타낸다. ASADAL 에서는 이러한 사건들을 기본 사건 함수에 의해 정의함으로써 사용자 요구사항 명세와 시스템 모델에 사용되는 사건들의 일관성 유지시킨다. 가령 그림 3 의 start_elevator 사건이 operator 가 elevator control system 에게 start 라는 제어 신호(Control Signal)를 보냈다는 사건이라면 *gen(start)*라는 사건 함수를 이용하여 사건을 정의할 수 있다. *gen(start)*라는 사건이 시스템의 내부

행동을 제어하는데 사용된다면 시스템 모델 명세에 이 사건이 나타나게 되고 이로써 사용자 요구 명세와 시스템 모델 명세간에 사건의 일관성을 유지하게 된다.

- RTET는 명세를 보다 정확하게 하기 위해서 동시에 발생하는 사건들을 의도적으로 표현하는 그래픽칼한 기호를 제공한다. 이 기호는 타원 형태를 지니고 있으며 자신에게 연결된 모든 사건 전달자의 사건들이 동시에 발생함을 나타낸다.
- RTET는 조건 행동의 명세를 지원한다. 이런 특징은 RTET 명세를 보다 이해하기 쉽게 한다. 예를 들어, 보일러의 밸브를 개폐하여 물을 끓이는 기계를 생각해 보자. 이 기계의 물 온도가 50도 이상이라면 물을 끓이지 않는다고 가정한다. 이러한 시스템의 행동을 정확히 나타내기 위해서는 물 온도가 50도 이상인 조건과 그렇지 않은 조건에 해당되는 두개의 RTET 명세를 생성한다.
- ASADAL은 실시간 요구사항(Real-Time Requirements)를 표현하기 위한 몇 가지 기본자를 제공한다. 이 기본자는 사건들 사이의 시간의 양적인 관계를 나타내는데 사용된다. 가령 예를 들면 그림 3에서 button2_press라는 사건이 발생한 후에 많아야 0.5초 안에는 up_direction 사건이 발생해야 한다는 실시간 요구사항을 명세하고자 하는 경우에는 실시간 명세 기본자를 사용하여 그림 3의 하단부에 나타난 것과 같이 max_dist(button2_press, up_direction,0.5)으로 명세할 수 있다. 다음은 실시간 명세 기본자에 대한 설명이다.
  - jst_dist(e1,e2,tm) : 이 기본자는 사건 e1과 e2 사이에 시간적 거리가 정확히 tm 임을 나타낸다.
  - max_dist(e1,e2,tm) : 이 기본자는 사건 e1과 e2 사이의 시간적 거리가 최대한 tm 임을 나타낸다.
  - min_dist(e1,e2,tm) : 이 기본자는 사건 e1과 e2 사이의 시간적 거리가 적어도 tm 임을 나타낸다.

2. 시스템 모델

시스템 모델은 TES 와 PDFD 를 이용하여 명세되고 상세화된다. ASADAL 은 시스템 모델의 기능적 관점과 행위적 관점을 명료하게 분리시킨다. TES 는 시스템의 내부 행동을 명세하는 데 사용되고 PDFD 는 시스템의 기능을 명세하는 데 사용된다. 이 절에서는 이들 각각에 대해서 언급한다.

가. 기능적 명세

시스템의 기능을 명세하는 데 사용되는 PDFD 는 STATEMATE 의 Activity-Chart 를 기반으로 하여 몇 가지 추가적인 특징을 제공한다.

시스템의 기능(Function)은 직사각형 모양의 기능자(Process)로 표현된다. 각 기능자는 데이타나 제어 신호(Control Signal)을 입력으로 받고 출력으로 처리된 데이타나 제어 신호를 생산한다. 복잡한 기능자는 하향식 기능 분해를 통해 새로운 PDFD 다이어그램에 세부적인 기능으로 상세화된다.

기능자 중에서 PDFD 의 가장 하위 단계에 위치한 것으로 더이상 기능 분해를 할 필요가 없는 것을 기본 기능자(Primitive Process)라고 한다. 기본 기능자의 역할은 기능 명세서(Process Specification)에 기술된다. ASADAL 에서는 전체 명세를 시뮬레이션하기 위해서 기능 명세서를 수행 가능한 정형적인 언어인 시뮬레이션 언어를 이용하여 기술한다. 기본 기능자의 기능 명세서는 기본 기능자로 들어오는 입력 데이타가 있을 때 이를 처리하여 출력 데이타를 발생시키는 형식으로 기술된다. 기본 기능자의 기능 명세

서에 대해서는 제 4 장에서 자세히 다룰 것이다.

기본 기능자에는 프로세싱 시간이 할당될 수 있다. 이 프로세싱 시간은 실험적인 데이타를 바탕으로 한 근사치로서 실시간 요구사항(Real Time Requirements)의 민감성 검사(Sensitivity Test)에 유용하게 사용된다. 예를 들면 각 프로세스(Process)에 프로세싱 시간을 할당한 후 명세를 수행시켜 봄으로써 어떠한 프로세스의 프로세싱 시간이 실시간 요구사항을 만족 시키는데 민감한 가를 찾아내고 이 프로세스의 프로세싱 시간을 적당히 조절하여 다시 명세를 수행시켜 봄으로써 사용자의 요구사항에 보다 부합하는 명세를 만들 수 있다.

PDFD 에서는 의식의 추적성을 높이기 위한 `항구 연결자(Port Connector)'를 제공한다.. 항구 연결자는 기능자(Process)가 기능 분해(Fucntional Decomposition)에 의해서 상세화 될 때, 그 기능자에 연결된 정보 흐름자(Information Flow)의 위치를 다음 단계의 PDFD 에 알려 주는 역할을 한다. 이런 항구 연결자의 존재는 한 기능자와 그 기능자를 기능 분해하여 생성된 PDFD 사이에 의식의 추적성을 높혀준다.

PDFD 는 전체 시스템의 기능을 세부 기능으로 나누고 기능들간의 데이타 전송만을 나타낸 것으로 시스템의 행동에 대해서는 표현하지 않는다. 즉 PDFD 내의 기능자들이 언제 활성화되고 언제 비활성화되는지, 어떠한 순서로 수행이 되는지에 대한 기술은 나타나 있지 않다. 이러한 시스템 행동은 행동 명세서인 TES 에 의해 기술되고 기능 명세서와 행동 명세서의 연결은 PDFD 내의 제어 기능자(Control Process)에 의해 이루어진다. 제어 기능자는 둥근 사각형 모양으로 표현되며 제어 기능자가 속한 PDFD 다이어그램 내의 기능자들이 어느 순간에 활성화되고 자신의 일을 하며 어느 순간에 비활성화되는 지를 나타내는 TES 다이어그램과의 연결자 역할을 한다.

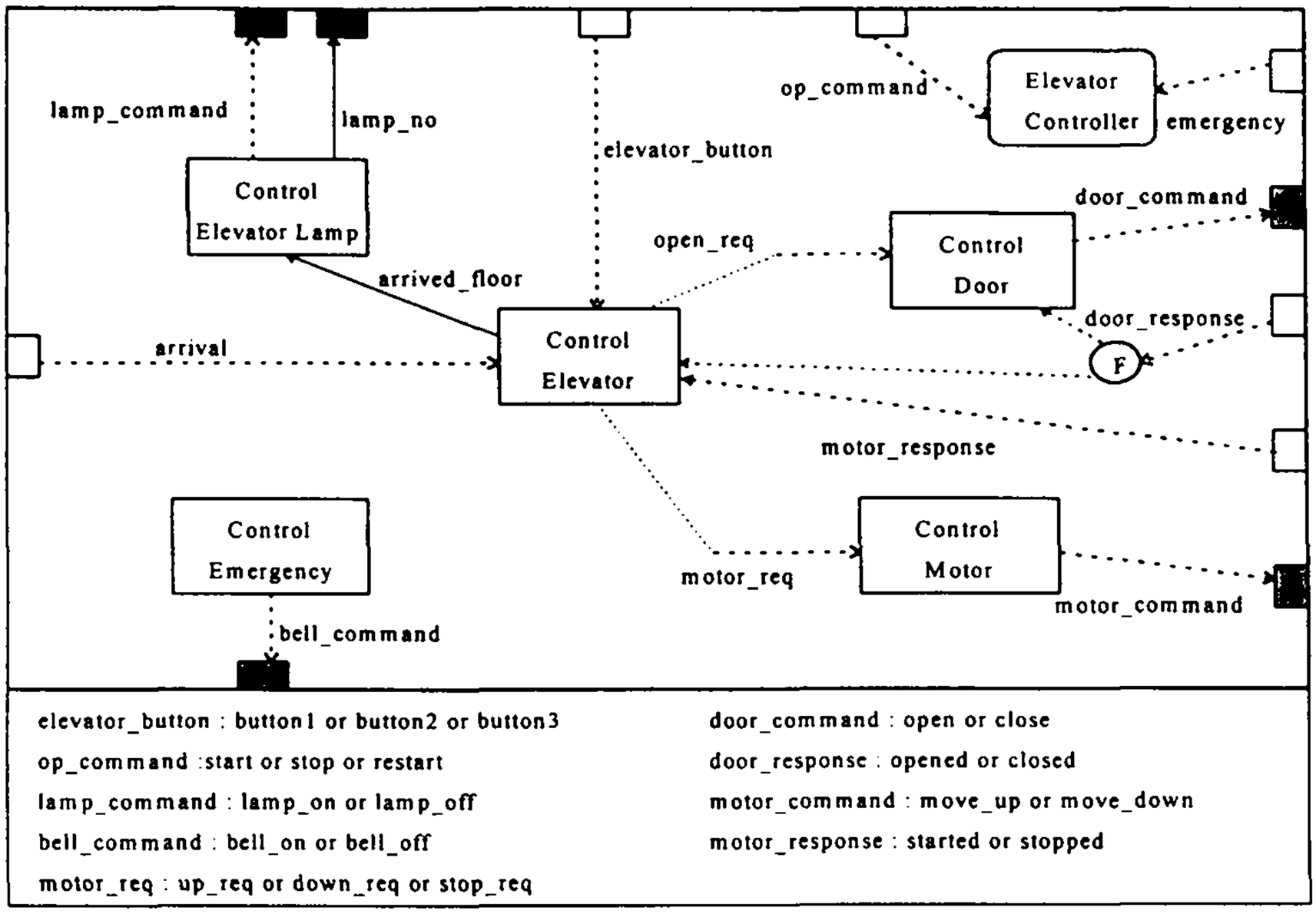


그림 5 엘리베이터 제어 시스템을 위한 PDFD 다이어그램

그림 5와 그림 6는 엘리베이터 제어 시스템의 기능 명세서의 한 예이다. 그림 5는 첫번 째 단계의 기능 명세서이고 그림 6는 그림 5의 Control Elevator 기능자를 기능 분해한 다이어그램이다. 바깥쪽 사각형 테두리에 위치한 작은 사각형은 항구 연결자를 나타낸 것으로 그림 5의 Control Elevator 기능자로 들어오고 나가는 정보 흐름자의 위치가 그림 6의 PDFD 다이어그램 사각형 테두리에 있는 항구 연결자의 위치와 동일한 것을 알 수 있다. 그러므로 PDFD 전체 구조를 따라 정보의 흐름을 쫓아갈 때 의식의 흐름이 끊어지지 않고 자연스럽게 이어질 수 있다.

그림 5의 둥근 사각형 모양의 Elevator Controller와 그림 6의 Request Controller는 각 다이어그램 내의 기능자들의 행동을 제어하는 역할을 하는제어 기능자로서 TES 다이

어그램과 연결되어 있다.

그림 6 의 Input Elevator Request 기능자와 Determine Next Direction 기능자는 더이상 기능 분해를 할 수 없는 기본 기능자이다. 이들 기능자의 기능 명세서는 제 4 장에서 소개된다.

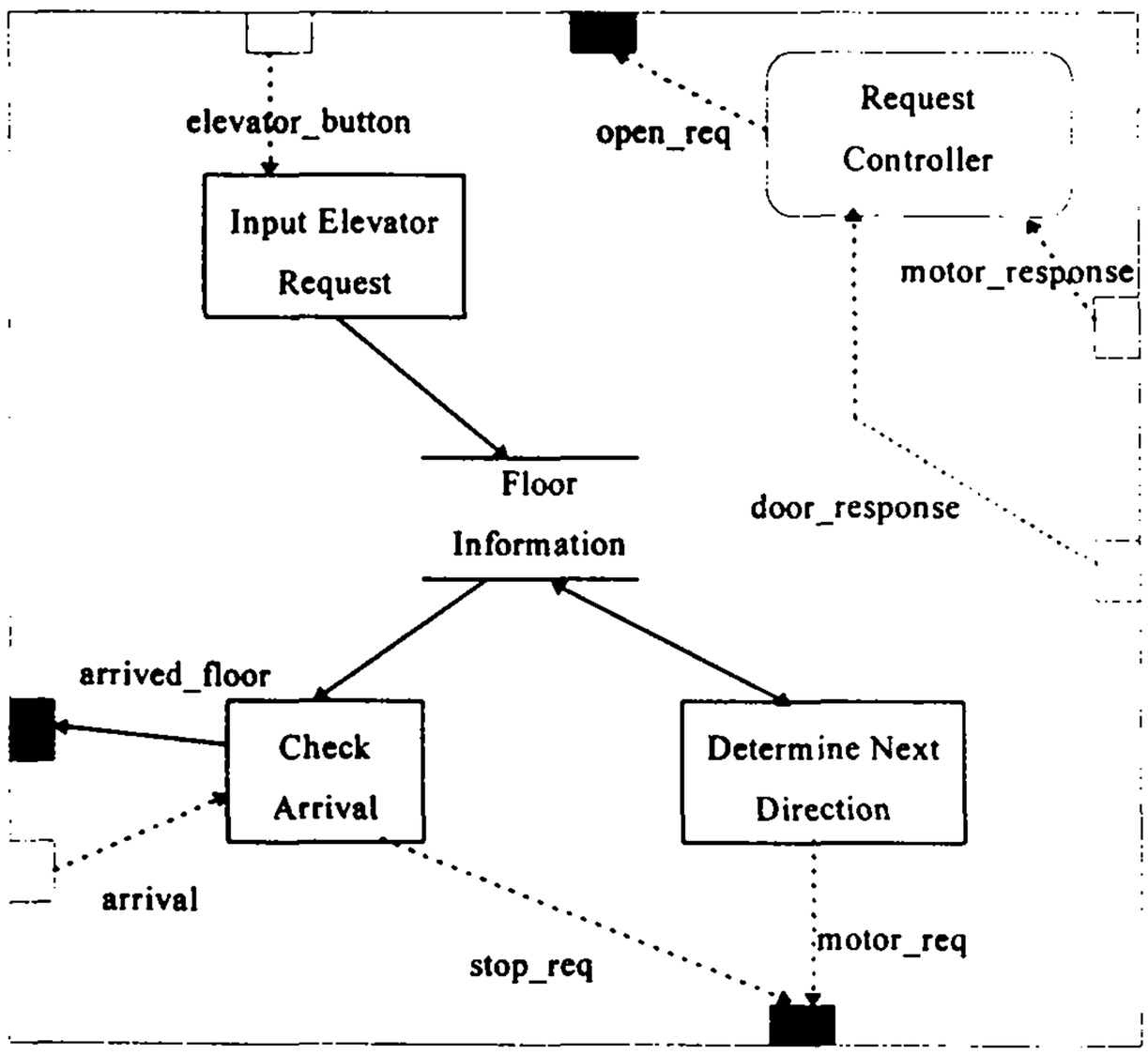


그림 6. 그림 5 의 Control Elevator 로 부터 기능 분해된 PDFD 다이어그램

나. 행위적 명세

행위 명세는 TES 에 의해 작성된다. TES 는 STATEMATE 의 Statechart 에 기반을 두고 실시간 시스템의 다양한 시간적 행동을 표현하기 위해 부가적으로 절대 시간 및 상대 시간 개념을 첨가하였고 몇 가지 구별되는 특징을 가지고 있다. 다음은 이러한 특징들에 대해 요약해 놓은 것이다.

Event	Abbreviation	Meaning
entered(S)	en(S)	State S is entered.
exit(S)	ex(S)	State S is exited.
true(C)	tr(C)	Value of condition C is set to true.
false(C)	fs(C)	Value of condition C is set to false.
arrived(D)	arv(D)	Information D is received. D may be data-item or control signal.
generated(D)	gen(D)	Information D is generated. D may be data-item or control signal.
timeout(E, N)	tm(E, N)	Event is generated N(time) from the most recent occurrence of event E.
teminated(P)	tmd(P)	Process P is terminated.

표 1. ASADA 기본 사건 함수

- ASADAL 은 사건(Event)과 제어 신호(Control Signal)을 구분한다.

  제어 신호는 제어 정보를 나타내는 데이타로서 사건과는 구별된다. 사건은 단순히 어떠한 일이 일어 났다는 것을 나타내는 신호일 뿐이지 어떠한 정보가 흘러가는 것은 아니다. ASADAL 은 기본적인 사건 함수를 제공함으로써 사건을 표현할 수 있게 한다. 이러한 사건 함수로 정의된 사건들만이 상태 전이(State Transition)을 가능하게 한다. 표 1 은 ASADAL 의 기본 사건 함수들을 나타낸 것이다.

- 기능자의 수행 제어를 위해서 기능자는 그것을 제어하는 TES 명세의 한 상태에 할당된다.

예를 들어 그림 4의 기능자 중 "Control Door"기능자가 그림 5의 "Door Processing"상태에 할당된다면 시스템이 수행 도중 "Door Processing"상태로 들어오는 경우에 "Control Door"기능자는 수행되게 된다. 기능자의 구체적인 수행 제어는 그 기능자가 할당된 상태(State)의 상태 명세서를 따른다. 즉, 기능자를 수행시킬 때 처음 부터 새로이 수행 시킬 것인지, 이전에 수행된 것 이후부터 수행 시킬 것인지를 명세한 상태 명세서에 의거하여 기능자를 수행시키게 된다. PDFD의 기능자가 할당된 모든 상태(State)가 기능자의 수행 제어를 위해 상태 명세서를 가질 필요는 없다. 전체 PDFD 구조의 최하위 단계에 위치한 기본 기능자(Primitive Process)가 할당된 상태만이 상태 명세서를 가지면 된다. 왜냐하면 중간 단계의 기능자의 수행 제어는 그 기능자의 자식 다이어그램(Child Diagram)에서 담당하게 되므로 더이상 자식 다이어그램이 없는 기본 기능자의 수행 제어를 위해 상태 명세서가 필요하다.

상태 명세서에는 기능자의 수행 제어를 위해 4가지 기능자 제어 명령어를 사용한다. 즉,Start, Abort, Resume, Suspend 네 가지가 있다. 이러한 기능자 제어 명령어는 기능자가 수행되는 상태(State)에 들어올 때와 나갈 때를 기준으로 사용된다. 예를 들면 어떠한 상태에 들어온 경우 이 상태와 연관된 기능자를 처음부터 수행시키고자 할 때는 Start 명령어를, 이전에 수행되었던 것 이후부터 다시 수행시키고자 할 때는 Resume 명령어를 사용한다. 상태를 빠져 나갈 때도 비슷한 의미로 Abort 나 Suspend 둘 중의 하나를 사용하게 된다.

다음은 상태 명세서의 두 가지 경우를 나타낸 예이다. (a)는 상태를 빠져 나갈 때 이전 상태를 기억한 후 다시 들어올 때 이전 상태부터 재수행하는 경우이

고 (b)는 상태에 들어올 때는 언제나 새로이 처음부터 수행하고 나갈 때는 현재 수행 중인 일을 중단하는 경우의 상태 명세서이다.

(a)	(b)
On Entry resume On Exit suspend	On Entry start On Exit abort

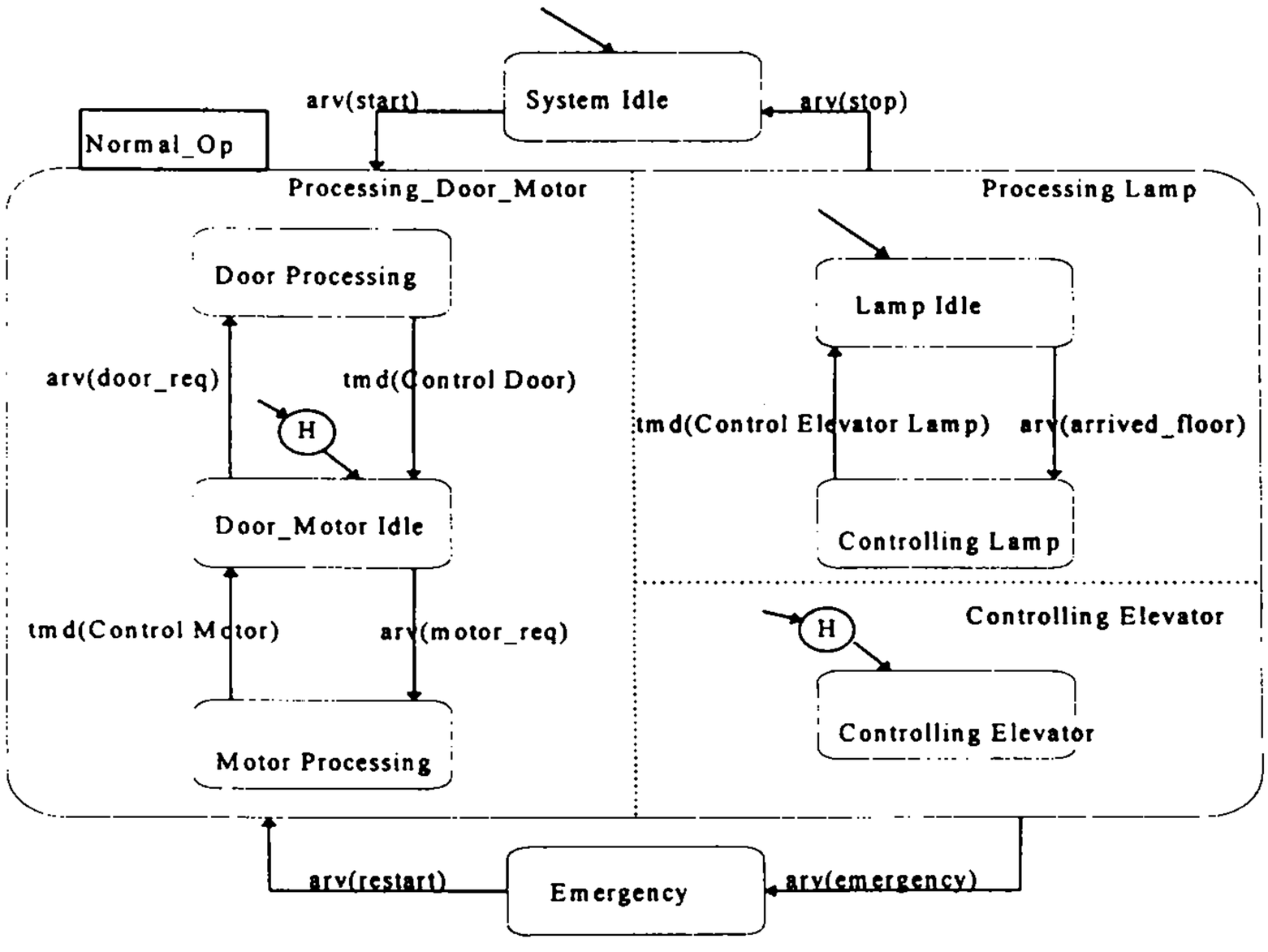


그림 7. 그림 5의 제어 기능자인 Elevator Controller와 연결된 TES 다이어그램

행위 명세에 대한 예는 그림 7과 그림 8에 나타난 바와 같다. 그림 7은 그림 5에

대한 행동 명세서이고 그림 8은 그림 6에 대한 행동 명세서이다.

그림 7에 나타난 시스템 행동을 간단히 설명하면 다음과 같다. 먼저 초기 상태는 `**System Idle**' 상태에서 시작한다. `**start**' 제어 신호가 도달했다는 의미의 `**arv(start)** '사건이 감지되면 시스템 상태는 `**Normal_Op**' 상태로 바뀐다. `**Normal_Op**' 상태는 세 개의 병행적인 상태로 나뉘어진다. 즉, 문과 모터를 처리하는 상태인 `**Processing Door Motor**' 상태, 등(Lamp)를 처리하는 `**Processing Lamp**'상태, 그리고 엘리베이터를 제어하는 `**Controlling Elevator**'상태로 나뉘어 진다. `**Processing Door Motor**'상태로 들어오면 초기에는 디폴트(Default)로 `**Door Motor Idle**'상태에서 시작한다. 이 상태에서 `**arv(door_req)**'사건이 감지되면 시스템은 `**Door Processing**'상태로 들어가게 된다. 이때, `**Door Processing**'상태에 그림 5의 `**Control Door**'기능자가 할당되어 있다면 `**Control Door**'기능자는 활성화되어 수행되게 된다. 만약 `**Control Door**'기능자가 기본 기능자라고 하면 `**Door Processing**'상태에는 `**Control Door**'기능자의 수행 방법을 나타내는 상태 명세서가 기술되어야 하고 이 상태 명세서에 따라 `**Control Door**'기능자가 처음 부터 일을 시작할지, 이전 작업 이후부터 일을 재계할지가 결정 된다. `**Control Door**'기능자가 자신의 일을 끝마치게 되면 `**tmd(Control Door)**'사건이 발생하고 이 사건에 의해 시스템 상태는 다시 `**Door Motor Idle**'상태로 되돌아 간다.

만약 `**Door Processing**'상태에 있을 때, `**arv(emergency)**'사건이 발생하면 시스템의 상태가 `**Emergency**'상태로 바뀜에 따라 현재 수행 중인 `**Control Door**'기능자의 작업이 보류된다. 그런 후에 `**arv(restart)**'사건이 발생하여 시스템이 다시 `**Normal_Op**'상태로 들어올 때, 이번에는 디폴트로 `**Door Motor Idle**'상태로 들어가지 않고 이전에 수행 중이었던 `**Door Processing**'상태로 들어가 이전에 수행된 과정 이후부터 수행을 재계한다. 이처럼 이전 상태를 기억하고 수행을 재계하는 메카니즘을 명세에 나타낸 것이 바로 역사 연결자(History Connector)이다. 역사 연결자는 그림 7에 나타난 바와 같이 원 중

심에 'H'자를 포함은 기호로 표시된다. 역사 연결자가 있는 상태는 그 상태를 빠져나갈 때 현재까지 수행 중인 과정을 저장한 후, 다시 들어올 때 기억된 과정 이후부터 일을 재개하게 된다.

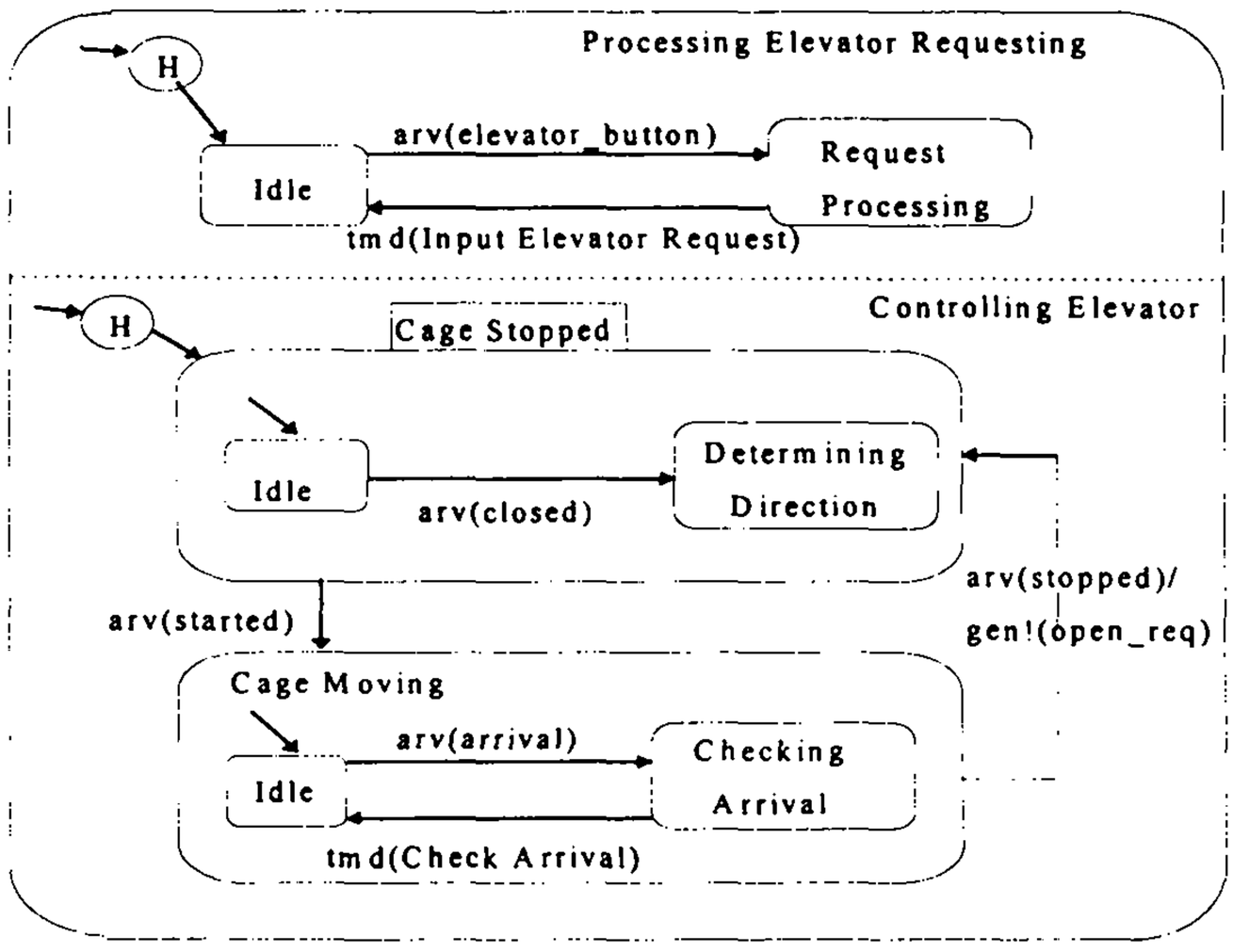


그림 8. 그림 6 의 Request Controller 와 연결된 TES 다이어그램.

4 절 시뮬레이션 방법

시스템 모델이 설정된 후에는 시스템 모델의 일관성을 유지하고 정확하게 작성되었는지, 사용자의 요구사항을 모두 만족하는 지를 검사하여야 한다. 이 장에서는 시스템 모델을 시뮬레이션을 통해 분석하는 방법을 설명한다.

1. 기반 개념

이 절에서는 ASADAL 시뮬레이션의 기반 개념에 대해 자세히 다룬다. ASADAL 시뮬레이션의 기반 개념을 다음의 관점에서 설명한다.

- 시뮬레이션 종류 및 범위
- 외부 개체와 내부 개체
- 시뮬레이션 시간과 진행 방법
- 시뮬레이션 모드
- 추계적 시뮬레이션

가. 시뮬레이션 종류 및 범위

ASADAL은 작성된 명세들을 여러 각도로 시뮬레이션 할 수 있는 기능을 제공함으로써 분석자가 시스템의 행동과 기능을 이해하고 검증하는 작업을 도와준다. 시뮬레이션은 작성된 명세 전반의 행동을 분석할 수도 있고 일 부분의 명세에 대한 행동만을 분석할 수도 있다. 분석자는 시뮬레이션을 시작하기 전에 어떠한 명세를 시뮬레이션할 것인가를 결정하여야 한다. 다음은 ASADAL이 제공하는 시뮬레이션 종류를 차례대로 설명한 것이다. 분석자는 다음에 설명될 시뮬레이션 종류를 선택하고 시뮬레이션 범위를 지정함으로써 시뮬레이션을 시작할 수 있다.

(1) TES 명세의 시뮬레이션

이 시뮬레이션은 TES 명세를 사용자 관심에 따라 여러 방법으로 운영함으로써 시스템의 내부 행동을 동적으로 보여 준다. 다음은 TES 명세를 운영하는 방법들이다.

- 시간 진행 - 시스템 Clock에 의해 시간이 가도록 하고 사용자의 사건 발생으로 시뮬레이션이 진행된다. 이 때, 시간에 따라 스케듈링(Scheduling)되고 있는 사건이나 행동은 해당되는 시간이 되면 자동적으로 발생되거나 수행된다.
- 단계 진행 - 사용자의 사건 발생으로 상태의 전이가 가능하면 한 단계의 상태

전이가 일어난다.

- 계속 진행 - 사용자의 사건 발생으로 상태의 전이가 가능한 단계까지 계속 상태가 전이된 후 안정화된 상태에서 멈춘다. 이 때, 시간은 흘러가지 않는다.

위의 방법들은 사용자의 의도에 따라 시뮬레이션 도중에 언제든지 서로 교환될 수 있으며, TES 명세에 사용되는 변수의 값도 사용자 임의대로 아무 때나 바꿀 수 있는데 이런 특징은 시뮬레이션 시간을 극적으로 절약해 준다.

(2) TES 명세와 PDFD 명세의 시뮬레이션

이 시뮬레이션은 한 PDFD에 명시된 기능자들의 활성, 비활성 상태를 그 PDFD와 결합된 TES를 운영함으로써 동적으로 보여 준다. TES에 의해 활성화된 기능자는 명세된 기능을 수행하게 되는데, 중간 단계의 기능자인 경우에는 다음 단계의 PDFD와 이와 결합된 TES를 수행시키게 된다. 이러한 과정은 PDFD 전체 구조를 따라 진행되게 된다. 만약 활성화된 기능자가 기본 기능자인 경우에는 그 기본 기능자의 기능 명세서를 해석하여 수행한다. 이 시뮬레이션을 통해서 분석자는 기능들이 적절한 순서로 수행되는지를 살펴볼 수 있다. 또한, TES 운영이 현 PDFD의 기능자들의 사건과 변수에 의해서 이루어지므로 TES의 정상적인 운영의 여부에 따라 기능자들의 오류를 검사할 수 있다.

(3) RTET 명세와 PDFD, TES 명세의 시뮬레이션

이 시뮬레이션은 RTET 명세에 나와 있는 시나리오에 따라 TES 명세를 운영하고 TES 명세에 따라 PDFD 명세가 수행된다. 이 시뮬레이션을 통해서 시스템 모델인 PDFD 명세와 TES 명세의 내부 행동이 RTET 명세의 외부 행동을 지원하는지 동적으로 검사한다. 이 시뮬레이션의 주된 관심은 RTET 명세에 나오는 시스템에 대한 자극 사

건들을 사용해서 TES 명세를 운영할 때 RTET 명세에 나오는 시스템의 반응 사건들이 TES 와 PDFD 명세로부터 유도되는지를 살펴보는 것이다.

나. 내부 개체와 외부 개체

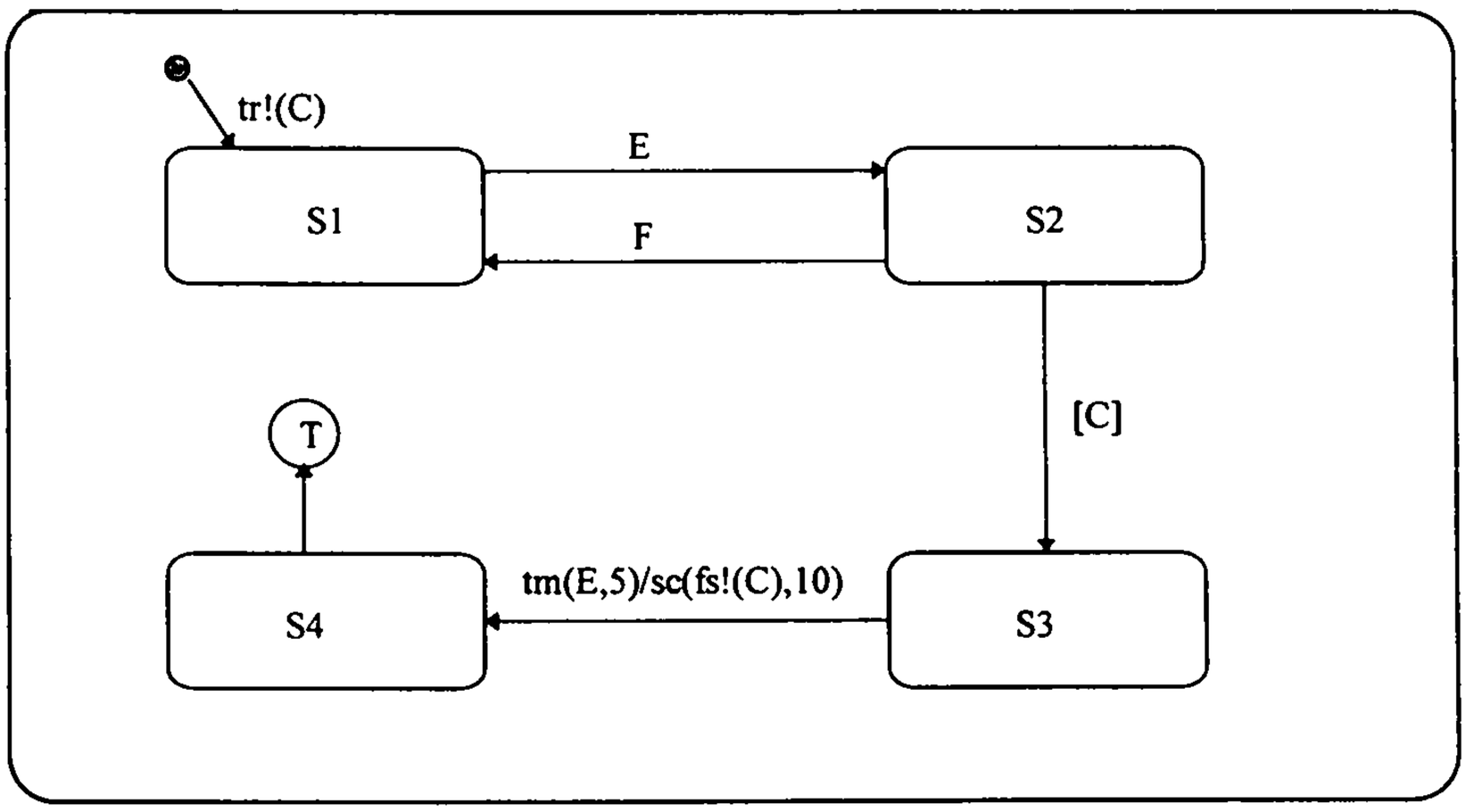
ASADAL 에서는 명세가 완전히 끝나지 않은 중간 과정의 개체(Element)나 시뮬레이션 범위 밖에 있는 개체를 외부 개체(External Element)라고 하고 범위 안에 있는 것을 내부 개체(Internal Element)라고 부른다. 내부 개체의 행동은 명세서에 작성되어 있으므로 명세서를 따르지만 외부 개체의 행동은 시뮬레이션 범위 밖이므로 명세되어 있지 않다. 그러므로 시뮬레이션 도중에 외부 개체가 생성하는 데이타는 사용자가 대화형으로 발생시키거나 외부 개체의 행동을 추계적(Stochastic)으로 나타낸 시뮬레이션 드라이버를 통해서 발생시킬 수 있다. 시뮬레이션 드라이버에 대한 내용은 제 4 장 제 3 절에서 자세히 다룬다.

예를 들면 그림 5 에서 시뮬레이션 범위를 이 다이어그램과 다이어그램 내의 기능자를 상세화한 자식 다이어그램(Child Diagram)까지 포함한다면 `Control Elevator Lamp'기능자, `Control Elevator'기능자, `Control Door'기능자, `Control Emergency'기능자, `Control Motor'기능자는 내부 개체이므로 이들이 발생시키는 데이타나 제어 신호는 시스템 명세에 따라 행동한다. 하지만 그림 5 의 다이어그램 밖에서 들어오는 데이타나 제어 신호는 사용자가 발생시키거나 시뮬레이션 드라이버에 의해 발생이 된다.

다. 시뮬레이션 시간과 진행 방법

시뮬레이션에 사용되는 시간은 실제 시간(Real Time)이 아닌 시뮬레이션 시간

(Simulation Time)이다. 즉 시뮬레이션을 수행하는 분석자가 시간이 흐르는 단위를 지정해 놓으면 시뮬레이션 도구는 정해진 시간 단위대로 시간을 증가시키게 된다. 시뮬레이션에서 시간을 증가시키는 방식에는 두 가지가 있다. 하나는 지정된 시간 만큼 시간을 증가시키는 방식이고 다른 하나는 다음 번에 스케듈링될 작업의 시간까지 시간을 증가시키는 방식이 있다. 이러한 시간의 증가 방식은 아이콘을 통해 사용자가 쉽게 이용할 수 있도록 시뮬레이션 도구에서 제공하고 있다.



림 9. 시뮬레이션 시간과 진행의 예제 TES 다이어그램

시뮬레이션 시 시간의 흐름과 시뮬레이션 진행(Step)과는 관계가 없다. 즉 시간이 흐르지 않는 동안 몇 단계의 시뮬레이션이 진행될 수도 있고 시뮬레이션이 한 단계에 머물러 있는 동안 시간이 흐를 수도 있다. 왜냐하면 상태 사이의 전이에는 시간이 걸리지 않고, 시스템이 한 상태에 머물러 있으면서 어떠한 작업을 수행하고 있다면 시간이 흐르기 때문이다.

그림 9는 시뮬레이션의 시간과 진행에 대한 개념을 설명하기 위한 예이다.

- E와 F가 외부 사건(External Event)이고 시스템이 초기에 S1 상태에 있다고 가정하고 다음의 시나리오를 따라보자.
- 초기에 디폴트로 S1 상태로 들어갈 때의 행동(Action)으로 조건 C는 참이 된다. 현재 시간은 t0 이다.
- 외부 사건 E가 발생되었다. 첫 단계(Step)에 S1 상태에서 S2 상태로 전이된다. 두 번째 단계에서 C가 참이기 때문에 S2 상태에서 S3 상태로 전이된다.
- 사건 tm(E,5)는 t0+5 까지 스케듈된다. t0+5 시간이 되면 세 번째 단계로 S3 상태에서 S4 상태로 전이된다.
- 행동 fs!(C)는 t0+15 까지 스케듈된다. 이때 마침 연결자(Termination Connector)로 전이가 일어난다.

이상의 시나리오에서 보는 바와 같이 첫 번째 단계에서 두 번째 단계까지 진행할 때는 시뮬레이션 시간은 증가되지 않는다. 하지만 시스템이 S3 상태에 있는 동안이나 S4 상태에 있는 동안에는 사건과 행동이 스케듈링되고 있는 관계로 시간이 흐른다는 것을 알 수 있다.

ASADAL 시뮬레이터는 시뮬레이션의 진행을 위해 다음과 같은 진행 명령어를 제공한다. 이러한 진행 명령어는 대화형 시뮬레이션에 주로 사용되는 명령어이다.

- 단계 진행 - 시간의 증가 없이 한 단계를 진행한다. 그림 9에서 사건 E가 발생한 후에 단계 진행을 하게 되면 S1 에서 S2 로 상태가 전이된다. 그런 후에 단계 진행을 한번 더하면 S2 에서 S3 로 상태가 전이된다. 하지만 단계 진행을 한번 더 하게 되면 아무런 변화가 없다. 왜냐하면 시간이 진행되지 않기 때문에

tm(E,5) 사건이 발생하지 않기 때문이다.

- 계속 진행 - 계속 진행은 안정된 상태가 지속될 때까지 단계 진행을 계속한다. 가령, 그림 9 에서 계속 진행을 하게 되면 S1 에서 S3 까지 상태가 전이된 후 기다리게 된다.

라. 시뮬레이션 모드

시뮬레이션은 두 가지 모드로 수행될 수 있다. 하나는 대화형 시뮬레이션이고 다른 하나는 배치형 시뮬레이션이다. 대화형 시뮬레이션이 사용자에 의해 조정된다면 배치형 시뮬레이션은 시뮬레이션 범위 내의 명세를 수행시키고 영향을 미치는 시뮬레이션 드라이버로 부터 작동된다.

(1) 대화형 모드 시뮬레이션

대화형 모드 시뮬레이션은 사용자가 외부 개체의 데이타를 발생시키고 시뮬레이션 진행 명령어와 시간 진행 명령어를 사용하여 시뮬레이션을 진행시킨다.

다음 시나리오는 그림 5 와 그림 7 에 나타난 명세를 대화형으로 시뮬레이션한 것이다.

- 시스템은 초기에 System Idle 상태에 있다.
- 사용자가 op_command(start) 제어 신호를 발생시킨다. start 제어 신호가 Elevator Controller 제어 기능자에 도달함에 따라 arv(start) 사건이 발생한다.
- arv(start)사건에 의해 시스템은 Normal_Op 상태로 전이되고 다시 Normal_Op 상태의 서브 상태인 Door Motor Idle 상태와 Lamp Idle 상태 그리고 Controlling Elevator 상태로 들어간다.
- Controlling Elevator 상태로 들어감에 따라 이 상태에 할당된 Control Elevator 기능자가 활성화된다.

- 이 때 사용자가 elevator_button(button3) 제어 신호를 발생한다. Control Elevator 기능자는 이 제어 신호를 받아들여 motor_req(up_req) 제어 신호를 발생시킨다.
- arv(motor_req)가 발생함에 따라 시스템 상태는 Door Motor Idle 상태에서 Motor Processing 상태로 전이된다. 이 때 Control Motor 기능자는 활성화되어 수행된다.
- Control Motor 기능자가 motor_req 를 받아들여 motor_command(move_up)을 발생시키면 외부 Motor 는 motor_response(started)를 Control Motor 기능자로 보낸다.
- 층 도달 감지 센서가 arrival 제어 신호를 발생시키면 Control Elevator 기능자는 이를 받아 arrived_floor 데이타를 Control Elevator Lamp 기능자로 보낸다. 이때 arv(arrived_floor) 사건이 발생한다.
- arv(arrived_floor) 사건에 의해 Lamp Idle 상태에서 Controlling Lamp 상태로 전이된다. 이 때 Control Elevator Lamp 기능자는 활성화되어 lamp_command 제어 신호와 lamp_no 데이타를 발생시킨 후 수행을 마친다. 이때 tmd(Control Elevator Lamp) 사건이 발생하여 Lamp Idle 상태로 전이된다.
- 층 도달 감지 센서가 arrival 제어 신호를 발생시킴에 따라 Control Elevator 기능자는 엘리베이터가 요청 층(Requested Floor)에 도달했을 때 motor_req(stop_req)를 발생시킨다. Control Motor 기능자는 이를 받아들인 후 수행 중인 작업을 끝마친다. 이 때 tmd(Control Motor)사건이 발생하고 Motor Processing 상태에서 Door Motor Idle 상태로 전이된다.
- 외부 Motor 가 motor_response(stopped)를 발생시킴에 따라 Control Elevator 기능자는 door_req(open_req) 제어 신호를 Control Door 기능자에게 발생시킨다.
- 이 때 시스템은 arv(door_req) 사건에 의해 Door Processing 상태로 전이되고 Control Door 기능자가 활성화된다.

이와 같은 대화형 모드 시뮬레이션은 명세가 간단한 경우에는 유용하게 사용되지만

명세의 범위가 넓고 많은 양의 데이타를 처리하여야 하는 경우에는 부적합면이 많다. 그러므로 배치 모드의 시뮬레이션이 필요하게 된다.

(2) 배치 모드 시뮬레이션

배치 모드의 시뮬레이션에서는 시뮬레이션 범위 밖에서 시뮬레이션 범위 안으로 들어오는 모든 데이타나 제어 신호의 발생을 시뮬레이션 드라이버가 담당한다. 또한 시뮬레이션 드라이버는 멈춤점을 설정함으로써 시뮬레이션 도중에 멈춤점의 트리거 조건이 만족되면 시뮬레이션을 중단하고 멈춤점에 정의된 행동들을 수행하게 된다. 그런 후에 다시 시뮬레이션을 재계하게 된다. 시뮬레이션 드라이버에 대한 명세 언어에 대해서는 제 4 장 제 3 절에서 언급하다.

마. 추계적 시뮬레이션

ASADAL 에서는 표 2 에서 보는 바와 같이 추계적 난수 함수(Stochastic Random Function)을 제공한다. 이러한 것들은 외부 시스템으로부터의 입력이 확률적으로 들어오는 시스템을 명세할 경우나 임의적인 행동 특성을 가진 시스템의 일부분을 시뮬레이션할 때 유용하게 사용된다.

2. ASADAL 의 시뮬레이션 언어

ASADAL 시뮬레이션 언어는 시스템의 시간적, 기능적 행동을 표현하고 시뮬레이션하는 것을 목적으로 한다. 시스템 모델을 수행시키기 위해서는 PDFD 의 기본 기능자의 행동을 수행 가능한 시뮬레이션 언어로 작성하여야 한다. 그리고 외부 개체의 데이타 발생이나 특정한 상황을 대처하기 위한 멈춤점(Breakpoint)의 설정 등의 역할을

담당하는 시뮬레이션 드라이버의 작성도 시뮬레이션 언어를 이용한다. 이 절에서는 시뮬레이션을 하기 위해 시뮬레이션 프로그램의 두 가지 경우인 기본 기능자의 명세와 시뮬레이션 드라이버의 명세에 대한 구조와 활용 예에 대해서 언급한다.

Continuous Random Variables	Discrete Random Variables
Random(i)	Rand_poisson(m)
Rand_exp(r)	Rand_uniform(a,b)
Rand_runiform(a,b)	Rand_binomal(n,p)
Rand_normal(u1,u2)	

표 2. Random Functions in ASADAL

가. 기본 기능자를 위한 프로그램

기본 기능자는 기능 명세의 최하위 단계에 위치한 시스템의 기본적인 기능을 나타내는 것으로 시뮬레이션 언어를 이용하여 명세함으로써 전체 시스템 모델이 수행 가능하게 된다. 기본 기능자는 한 개 이상의 행동(Behavior)을 가지지만 한 순간에는 오직 한 행동만이 수행 가능하다. 한 행동이 선택되어 수행되어지는 시기는 그 행동을 수행하기에 필요한 데이타가 큐에 들어 있을 때이다. 한 행동이 수행되어지는 동안 계속해서 큐에 데이타가 들어오면 큐에 데이타가 더이상 없을 때까지 계속해서 프로세싱을 한다.

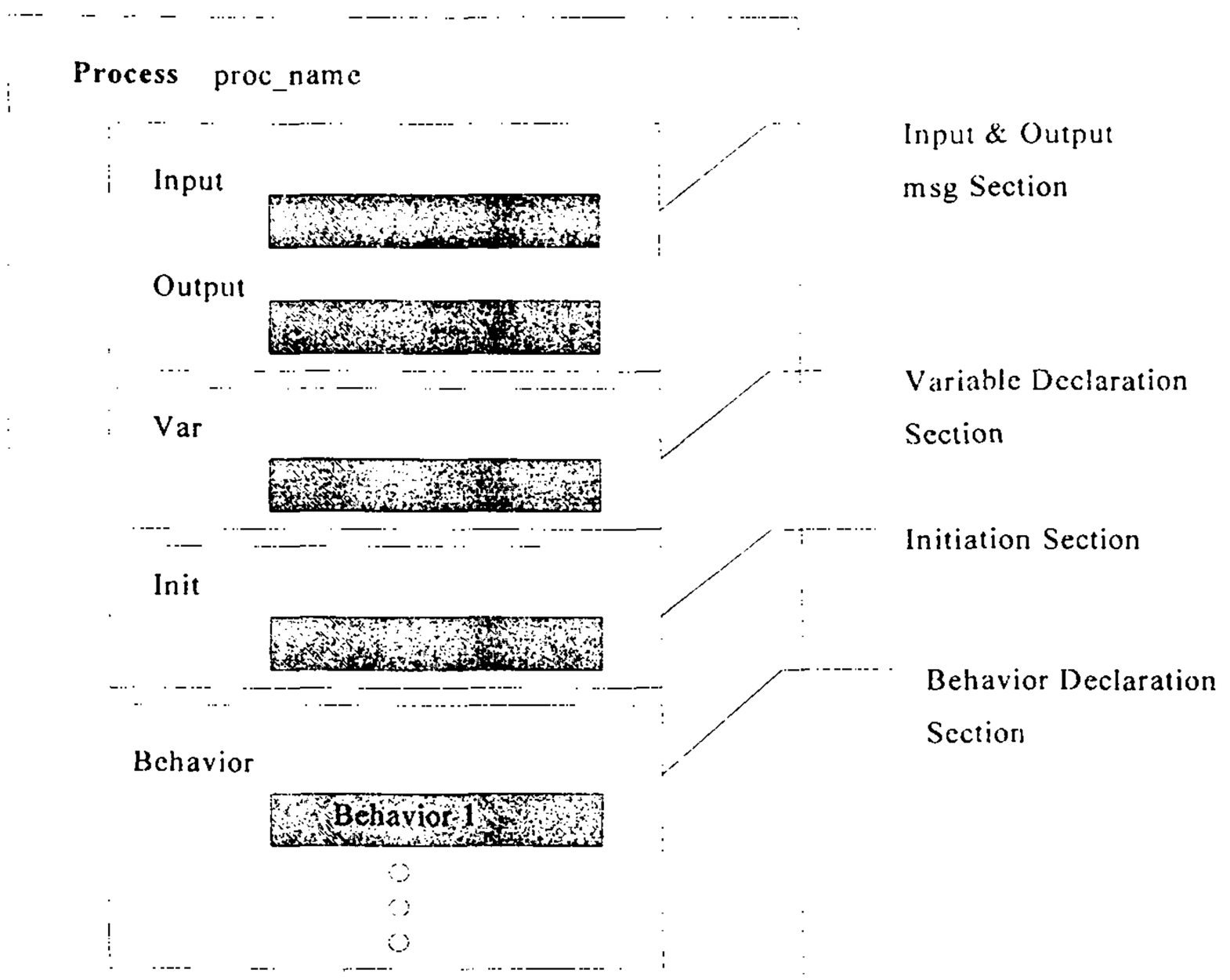


그림 10 프로그램 기본자를 위한 프로그램 구조

기본 기능자 내의 행동에 대해서 수행 시간을 지정할 수 있다. 이 수행 시간은 분석자가 실험적인 데이타를 근거로 표 2의 추계적 기본자(Stochastic Primitive)를 이용하여 지정할 수 있다. 일단 한 행동이 수행되기 위해 선택되어지면 그 행동의 지정된 수행 시간까지 기다린 후 정해진 행동을 취하게 된다.

그림 10은 기본 기능자의 행동 명세를 위한 시뮬레이션 프로그램의 구조이다. 다음은 이에 대한 각 부분별 설명이다.

(1) Input and Output message declaration

이 부분은 CASE 도구에 의해 자동적으로 생성되는 부분으로 메세지의 타입(Type)이 정의된 데이타 사전(Data Dictionary)를 근거로 한다. 기본 기능자로 들어오는 메세지는 "Input"이라는 키워드 다음에 선언되고, 나가는 메시지는 "Output", 들어 오다 나가는 메세지는"InOut"키워드 다음에 선언된다. 들어오는 메세지에 대해서는 크기가 무한한 큐가 할당된다. 즉 메세지가 들어올 때마다 큐에 차례대로 저장된다.

들어오고나 나가는 메세지의 선언은 다음 형태를 따른다.

[Input | Output | InOut]

<msg_name> : *<msg_type>*;

*<msg_type>*은 메세지의 타입을 나타내는 것으로 정수(Integer), 문자열(String), 논리(Boolean), 부동 소수(Float), 레코드(Record)등이 있다. 만약 *<msg_type>*이 레코드인 경우에는 다음의 형태를 따른다.

Record {

<msg_name> : *<msg_type>*;

.....

}

그림 11의 2-9 줄과 그림 12의 2-9줄은 메세지 선언부의 한 예이다.

(2) Variable Declaration

이 부분은 프로그램 내에 사용되는 지역 변수(Local Variable)을 선언한 곳이다. 변수의 타입은 정수, 문자열, 부동 소수, 논리, 레코드 중의 하나가 될 수 있다.

```
Var
        floor_number : Integer;
        CLOSED : Boolean;
```

(3) Initiation Section

이 부분은 이 프로그램이 시작될 때 처음 한번만 수행되는 문장들을 포함한다.

```
Init
        floor_number = 1;
        True!(CLOSED);
```

(4) Behavior Declaration

이 부분은 기본 기능자가 수행할 행동들에 대한 정의를 포함한다. 한 행동은 트리거 정의, 프로세싱 시간 지정, 수행될 행동 정의 순으로 선언된다. 이의 구체적인 형태는 다음과 같다.

```
On  [<trigger>]  Do
        [Processing_Time  <processing_time>]
        Begin
                <statement_list>
        End
```

한 행동의 활성화는 <trigger>에 의해서 결정된다. <trigger>는 매 수행 단계 (Execution Step)마다 만족 여부가 검사되는데 트리거가 만족되면 해당되는 행동이 활성화되어 수행된다. <trigger>는 선택적이며 만약 생략된 경우에는 이를 포함한 프로그

램이 시작되는 시점에 해당되는 행동이 자동적으로 활성화된다. 예를 들면, 그림 12의 12 - 19 줄은 이 프로그램이 활성화되는 시점에 수행된다.

<trigger>는 입력 메세지가 사용 가능하면 참(.True)가 되고 아니면 거짓(False)가 된다. 그러므로 Available(input_message) 기본자를 사용하여 표현된다. <trigger>는 한 개 이상의 Available(input_message)를 논리 연결자(and 나 or)로 결합하여 표현될 수 있다. <trigger>가 참이 되면 사용 가능한 메세지는 큐로부터 읽혀져서 행동 정의 부분에서 사용된다. 예를 들면 다음과 같다.

- On Available(arrived_floor) :
  arrived_floor 가 사용 가능할 때, 정의된 행동이 수행될 준비가 되어 있다는 것을 의미한다. arrived_floor 는 큐로 부터 읽혀져 와 행동 정의 부분에서 사용된다.
- On Available(button1) or Available(button2) :
  button1 이나 button2 둘 중의 하나가 사용 가능하면 관련된 행동이 수행될 수 있다는 것을 의미한다. 만약 두개가 동시에 사용 가능하면 둘 중에 하나가 비결정적(Non-deterministic)으로 선택된다.
- On Available(msg1) and Available(msg2) :
  msg1 과 msg2 가 모두 사용 가능할 때 관련된 행동이 수행될 수 있다는 것을 의미한다.

실시간 요구사항 검사를 위해 프로세싱 시간의 지정은 필수적이다. 각각의 행동마다 이를 수행하기 위해 걸리는 시간을 지정함으로써 어떠한 일을 수행하는데 걸리는 전체 시간을 알 수 있을 뿐만 아니라 전체 프로세싱 시간에 민감하게 반응하는 기능자를 찾아내는 민감성 검사(Sensitive Test)에도 유용하게 사용된다.

<processing_time>은 수행되는 시간을 나타내는 시간 값으로 추계적 기본자를 이용하여 표현될 수 있다. 시스템의 요구 분석 단계에서는 시스템의 수행 시간을 정확히 알 수 없으므로 기존의 데이타를 바탕으로 확률적인 값을 지정하는 것이 보다 유용하다.

행동 정의 부분은 "Begin" 키워드와 "End" 키워드 중간에 위치한다. 행동 정의는 표 3 의 문장들을 이용하여 기술된다.

Simple Statement	
Gen!(msg)	generate msg.
Write!(msg)	write msg to data_store.
Read!(msg)	read msg from data_store.
True!(cond)	make condition variable cond true.
False!(cond)	make condition variable cond false.
v : = exp	value of expression(exp) is assigned to message(v)

Program Flow Controlling Statement	
If C Then S1 Else S2	If condition C is true, execute S1, otherwise execute S2.
While C Do S1	While condition C is true, statement S1 is executed repeatedly.

표 3. statements for process specification.

그림 11 은 기본 기능자의 예를 나타낸 것이다. 이에 대해 자세히 살펴보면 다음과

같다.

```
Process  Input_Elevator_Request
   Input
      button1 : Boolean;
      button2 : Boolean;
      button3 : Boolean;
   InOut
      floor_info : Record {
                  cur_floor : Integer;
                  req_floor : Integer; }
      On  Arrival(button1)  Do
      Processing_Time  Rand_exp(0.5)
      Begin
            floor_info.req = 1;
            WRITE!(floor_info);
      End

      On  Arrival(button3)  Do
      Processing_Time  Rand_exp(0.5)
      Begin
            floor_info.req = 3;
            WRITE!(floor_info);
      End
```

그림 11. 그림 6 의 Input Elevator Request 기능자를 위한 기능 명세서

- Process Input_Elevator_Request

  프로그램 머리로서 Process 는 기능 명세서를 의미하고 Input_Elevator_Request 는 프로그램 이름을 의미한다. 즉 이 프로그램은 그림 6 의 Input Elevator Request 기능자의 기능 명세서를 의미한다.

- Input

  이 프로그램으로 들어오는 데이타를 선언하는 곳임을 나타내는 키워드.

- button1 : Boolean;

  button2 : Boolean;

  button3 : Boolean;

  button1, button2, button3 는 제어 신호로서 Boolean 타입을 가진다. button1, button2, 혹은 button3 가 이 프로그램으로 들어오면 큐에 쌓이다. button1 은 엘리베이터의 1 층으로의 요구를 의미하고 button2 와 button3 은 각각 2 층, 3 층으로의 요구를 의미한다.

- Output

  이 프로그램에서 나가는 데이타를 선언한 곳임을 나타내는 키워드. 이 키워드 다음에 선언된 데이타는 출력 데이타를 의미한다.

- floor_info : Record {

  cur_floor : Integer;

  req_floor : Integer; }

  floor_info 는 출력 데이타를 나타내며 타입은 레코드 타입이다. 즉 정수 타입의 cur_floor 와 req_floor 를 갖는 구조체이다. floor_info 의 의미는 층의 정보를 나타내는 데이타로서 현재 층을 나타내는 cur_floor 와 요청 층을 나타내는 req_floor 로 구성되어 있다.

- On Arrival(button1) Do

  행동을 활성화시키는 트리거 조건을 나타내는 문장이다. 즉 button1 이 큐에 있으면 트리거 된다.

- Processing_Time Rand_exp(o 5)

  행동의 수행 시간을 지정하는 문장이다. 이 문장에서는 수행 시간을 추계적인 함수를 이용하여 표현하였다. 즉 행동의 수행 시간을 평균 0.5 초의 지수 분포가 되도록 하였다.

- Begin

  행동을 정의하는 부분의 시작을 나타내는 키워드

- floor_info.req_floor = 1;

req_floor 에 1 을 셋팅하였다. 이것의 의미는 button1 이 일층으로의 엘리베이터 요청을 의미하므로 .요청 층을 나타내는 req_floor 에 1 의 값을 지정한 것이다.

```
1   Process   Determine Next Direction
2     Input
3        floor_info : Record {
4                     cur_floor : Integer;
5                     req_floor : Integer; }
6     Output
7        up_req : Boolean;
8        down_req : Boolean;
9        stop_req : Boolean;
10    On   Do
11       Processing_Time     Rand_exp(0.7)
12       Begin
13            READ!(floor_info);
14            If (floor_info.cur_floor > floor_info.req_floor)
15            Then
16                 Gen!(up_req);
17            Else If (floor_info.cur_floor < floor_info.req_floor)
18                 Gen!(down_req);
19   End
```

그림 12. 그림 6 의 Determine Next Direction 기능자를 위한 기능 명세서

- WRITE(floor_info);

  floor_info 의 정보를 데이타 저장소에 저장하는 것이다. floor_info 의 목적지가 데이타 저장소이므로 이 정보를 데이타 저장소에 쓰는 역할을 한다.

- On Arrival(button3) Do

  Processing_Time Rand_exp(0.5)

  Begin

  floor_info.req_floor = 3;

  WRITE!(floor_info);

  End

  button3 데이타가 도달하였을 때 트리거되어 Rand_exp(0.5)초 만큼 기다린 후 Begin 에서 End 사이의 문장을 수행한다

그림 12 는 기능 명세서의 또 다른 예를 나타낸 것이다. 다음은 이 기능 명세서를 자세히 설명한 것이다.

- *Process Determine Next Direction*

그림 6 의 *Determine Next Direction* 기능자에 대한 기능 명세서를 나타낸다.

- Input

  floor_info : Record {

  cur_floor : Integer;

  req_floor : Integer; }

  입력 데이타를 선언한 것으로 floor_info 는 이 프로그램으로 들어오는 레코드 타입의 데이타이다. 이 데이타는 데이타 저장소로부터 읽어 들어오는 데이타이므로 큐에 쌓이지는 않는다.

- Output

  up_req : Boolean;

down_req : Boolean;

stop_req : Boolean;

출력 데이타를 나타내는 것으로 up_req, down_req, stop_req 는 제어 신호를 나타낸다.

- On Do

  이 문장은 트리거 조건이 없다. 이 경우에는 이 프로그램이 활성화되는 시점에 트리거 된다.

- Processing_Time Rand_exp(0.7)

  프로세싱 시간이 Rand_exp(0.7)이 될 때까지 기다린다. 프로세싱 시간이 다되면 행동 정의 문장들을 수행하게 된다.

- Read!(floor_info);

  데이타 저장소로부터 floor_info 를 읽어 온다.

- If (floor_info.cur_floor > floor_info.req_floor)

  현재 층이 요청 층보다 높으면 다음을 수행한다.

- Gen!(up_req);

  up_req 를 발생시킨다.

- Else if (floor_info.cur_floor < floor_info.req_floor)

  현재 층이 요청 층보다 낮으면 다음을 수행한다.

- Gen!(down_req);

  down_req 를 발생시킨다.

3. 시뮬레이션 드라이버를 위한 프로그램

시뮬레이션 드라이버는 명세의 수행을 주관하는 텍스트 화일로서 시뮬레이션 언어를 사용하여 기술된다. 시뮬레이션 범위 내에 있는 모든 개체는 시뮬레이션 드라이버

프로그램 내에서 사용 가능하다. 시뮬레이션 드라이버 프로그램의 구조는 그림 13 과 같다.

- Variable Declaration and Initiation Section :

  기본 기능자를 위한 프로그램과 같은 형태를 지니다.

- External Behavior Section :

외부 개체의 행동은 시스템의 상태나 시뮬레이션 진행과는 상관없이 일어나는 경우가 보통이다. 이 부분에서는 이러한 외부 개체의 행동을 정의한다. 시뮬레이터는 여기에 정의된 행동을 따라 외부 개체의 데이타를 발생시킨다. 외부 개체의 행동은 다음과 같은 형태를 따라 정의된다.

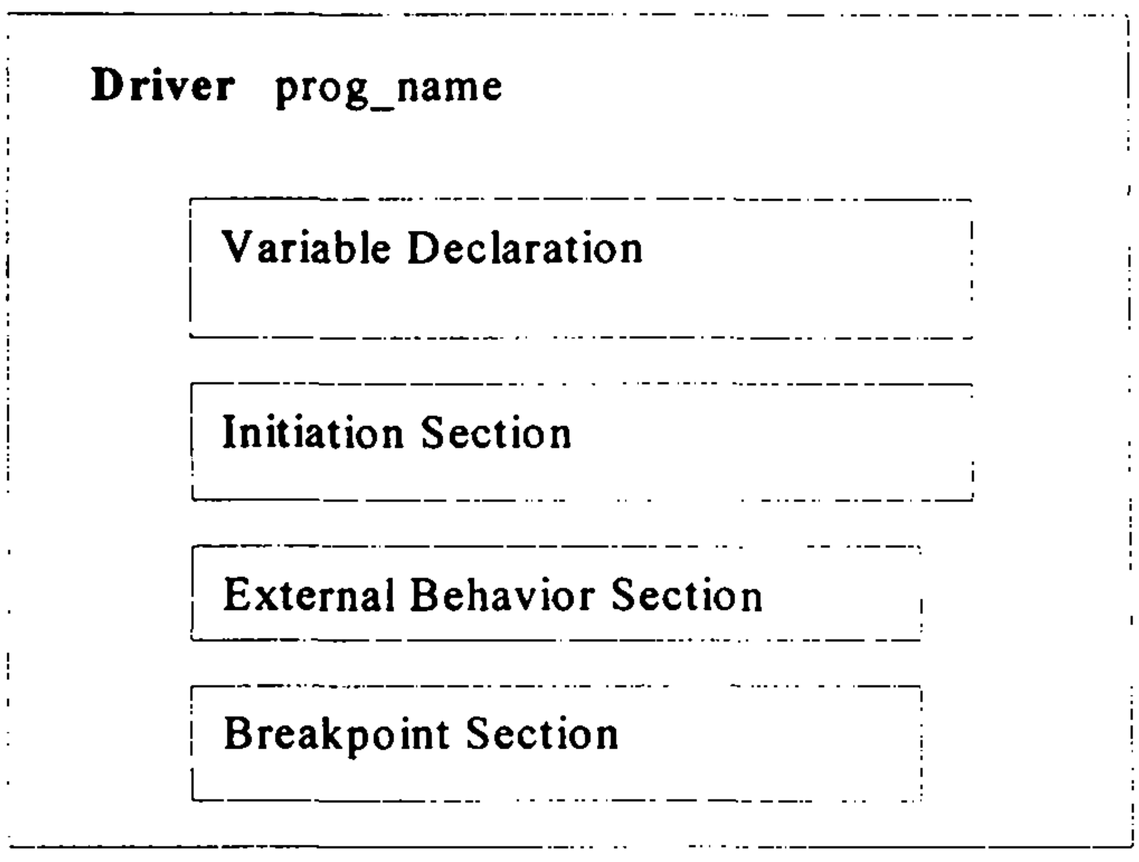


그림 13. Simulatino Driver 를 위한 프로그램 구조

Every <time_expression> :- <statement_list>

이것이 의미하는 바는 <statement_list>가 매 <time_expression>마다 수행되는 것을 말한다. <statement_list>는 기본 기능자에서 사용되는 것과 같은 것이다.

<time_expression>는 시간 값을 나타낸 것으로 추계적 기본자를 이용하여 표현할 수 있다. 예를 들면, 만약 사용자의 요청이 포아송(Poisson) 분포를 따라 들어올 때, 이를 모델링하면 다음과 같다.

Every Rand_poisson(m) :- Gen!(request) (m은 포아송 분포의 평균 값을 의미)

- Breakpoint Section

이 부분은 멈춤점(Breakpoint) 정의를 포함한다. 멈춤점은 시뮬레이션 도중 일어나는 특별한 경우를 다룰 때 유용하다. 가령, 시스템이 무한 루프를 돌고 있거나 비결정성(non-determinism)에 빠지는 경우, 이를 처리하고자 할 때 사용된다. 멈춤점은 매 수행 단계마다 검사되는데 멈춤점의 트리거 문장이 만족되면 모든 시뮬레이션 진행이 멈추고 관련된 문장이 수행된다. 멈춤점의 정의는 다음의 형태를 따른다.

Breakpoint <trigger_expression> Do
<statement_list>

<trigger_expression>은 사건 문장이나 조건 문장이 될 수 있다. 즉 트리거를 만족시키는 사건이 일어 났거나 조건이 만족된다면 해당되는 <statement_list>가 수행된다. <statement_list>는 기본 기능자 프로그램에 나타난 것과 동일한 것을 사용한다.

그림 14는 엘리베이터 제어 시스템을 위한 시뮬레이션 드라이버의 예이다. 다음

은 이 예제를 자세히 설명한 것이다.

- Driver Elevator

  Driver 는 시뮬레이션 드라이버를 나타내는 키워드이고 Elevator 는 시뮬레이션 드라이버의 이름이다.

- Variable Declaration

  지역 변수 선언부의 시작을 나타내는 키워드이다. 각 변수의 선언은 세미콜론으로 구분된다.

```
1  Driver  Elevator
2     Variable Declaration
3        button_no : Integer;
4     Init
5        button_no = 0;
6     Every  Rand_poisson(10) :-
7        button_no = Rand_Uniform(1,3);
8        If(button_no == 1) Then
9              Gen!(button1);
10       else if (button_no == 2)
11             Gen!(button2);
12       else if (button_no == 3)
13             Gen!(button3);
14
```

```
30    Breakpoint  in(normal_op)  Do
31       If (Rand_uniform(1,100) == 1) Then
32             Gen!(emergency);
33    Breakpoint  in(Emergency)  Do
34       If (Rand_uniform(1,10) == 1) Then
35             Gen!(restart);
```

그림 14. 시뮬레이션 드라이버를 위한 예제

- button_no : Integer;

button_no 는 엘리베이터의 요청 층을 나타내는 정수 값이다.

- Init

  이 프로그램이 시작될 때 처음 한번 수행되는 문장을 담은 부분을 나타내는 키워드이다.

- button_no = 0;

  button_no 를 0 으로 초기화하는 문장이다.

- Every Rand_poisson(10) :-

  외부 개체의 행동을 정의하는 부분의 트리거 부분이다. 즉 포아송 분포의 시간 간격으로 다음 문장들을 수행하게 된다.

- button_no = Rand_Uniform(1,3);

  1 에서 3 까지의 정수 숫자 중 하나를 동일한 확률로 골라 button_no 에 대입한다.

- If (button_no == 1) Then

  button_no 가 1 이 된 경우에 다음 문장을 수행한다.

- Gen!(button1);

  button1 제어 신호를 발생시킨다.

- else if (button_no ==2)

  button_no 가 2 인 경우에는 다음 문장을 수행한다.

- Gen!(button3);

  button3 제어 신호를 발생시킨다.

- else if (button_no == 3)

  button_no 가 3 인 경우에는 다음 문장을 수행한다.

- Gen!(button3)

  button3 제어 신호를 발생시킨다.

- Breakpoint in(normal_op) Do

  멈춤점의 트리거 부분으로써 시스템 상태가 normal_op 상태에 들어가면 이 멈춤

점이 트리거 된다.

- If (Rand_uniform(1,100) == 1) Then

  1 에서 100 까지의 숫자 중 임의로 골라 그 값이 1 이 되면 참이 되어 다음 문장을 수행하게 된다. 즉 0.01 의 확률로 다음 문장이 수행된다.

- Gen!(emergency);

  emergency 제어 신호를 발생시킨다.

- Breakpoint in(Emergency) Do

  시스템 상태가 Emergency 상태에 들어가면 이 멈춤점이 트리거 된다.

- If (Rand_Uniform(1,10) == 1) Then

  1 에서 10 까지의 숫자 중 임으로 골라 그 값이 1 이 되면 참이 되어 다음 문장을 수행하게 된다. 즉 0.1 의 확률로 다음 문장이 수행된다.

- Gen!(restart);

  restart 제어 신호를 발생시킨다.

5 절 ASADAL CASE 도구

본 절에서는 ASADAL 방법론을 지원하는 CASE 도구에 대해서 설명한다. ASADAL CASE 도구는 방법론의 효율적인 지원과 높은 일반성(Generic Property)를 지니기 위해서 그 기능면과 구조, 그리고 설계면에서 유용한 특징들을 제공하고 있다. 다음의 이어지는 절들은 이들 ASADAL 의 기능, 구조상의 특징들에 대한 설명이다.

1. ASADAL CASE 도구의 기능상 특징

대규모의 복잡한 실시간 시스템을 분석하기 위한 지원 도구로서 ASADAL CASE 도구는 크게 다음과 같은 기능상 특징을 가지고 있다.

- 사용자 요구사항 명세 언어인 RTET, 그리고 시스템 모델을 나타내는 TES, PDFD 를 쉽게 해주는 다이어그램 편집기를 제공한다.
- 작성된 다이어그램의 문법과 의미의 정확성을 분석하는 문법 검사기(Syntax Checker)와 의미 검사기(Semantic Checker)를 제공한다.
- 시스템의 행동과 기능에 대한 효과적인 이해와 검증을 위해서 RTET, PDFD, 그리고 TES 명세를 사용한 여러 각도의 시뮬레이션을 제공한다.

위에서 언급한 ASADAL CASE 도구의 각각의 기능들에 대한 설명은 다음과 같다.

ASADAL CASE 도구는 실시간 시스템을 분석하기 위해 제공하는 세가지 명세 언어를 수월하고 효율적으로 사용할 수 있도록 도와주는 각 명세 언어에 대한 다이어그램 편집기를 제공한다. ASADAL CASE 도구의 다이어그램 편집기는 사용자의 편의와 효율을 극대화하기 위해서 다음과 같은 특징을 가지고 있다.

- ASADAL 의 다이어그램 편집기는 지원하는 명세 언어의 모든 기본 구조를 ICON 으로 제공함으로써 다이어그램의 편집이 효과적으로 이루어지도록 한다.
- ASADAL 의 다이어그램 편집기는 풍부한 도움말 기능을 제공하여 적시에 적절한 도움말을 사용자가 언제나 구할 수 있도록 한다.
- ASADAL 의 다이어그램 편집기는 사용자가 편집을 수행할 때 발생하는 기본적인 문법적 오류를 지능적으로 방지해 준다. 예를 들어, 도착지가 없는 정보 흐름자는 그려지는 즉시 편집기에서 오류로 판정하여 사용자에게 알려 준다.

ASADAL 은 작성된 시스템 명세의 문법과 의미를 자동적으로 검사하는 기능을 제공한다. 앞의 문단에서 각 명세 언어로 다이어그램을 작성할 때 기본적인 문법 오류는 편집기를 통해서 즉시 판정되고 시정된다고 설명했다. 그 외의 문법 오류는 ASADAL 에서 별도로 제공하는 문법 검사기를 통해서 찾아지게 되는데 문법 검사기는 발견된

문법 오류에 대한 자세한 정보와 도움말을 사용자에게 제공한다. 명세의 의미 검사는 각 명세 언어로 작성된 다이어그램의 의미에 대해서 그 올바름을 검증하는 것이다.

ASADAL 은 작성된 명세들을 여러 각도로 시뮬레이션할 수 있는 기능을 제공함으로써 분석자가 시스템의 행동과 기능을 이해하고 검증하는 작업을 도와준다. 논리 기반의 정형적인 분석 방법은 시스템의 규모가 커 지고 행동이 복잡하면 매우 많은 시간과 기억소자를 사용한다. 그러므로 ASADAL 은 사용자의 관심에 따른 다양한 각도의 시뮬레이션 기능을 제공함으로써 분석적 이지는 못하나 비용을 적게 사용하는 의미 검증 방법을 제공한다. 이런 여러 각도의 시뮬레이션 기능은 실시간 시스템을 분석하는 데 필수적인 CASE 도구의 특징인데도 불구하고 많은 기존의 CASE 도구들은 이를 제공하지 않고 있다.

2. ASADAL CASE 도구의 구조

이 절에서는 ASADAL CASE 도구가 외부 환경 변화에 영향을 적게 받는 시스템이 되기 위해 외부 환경과 인터페이스 하는 계층을 내부와 분리한 세 계층 구조에 대해 설명한다.

ASADAL CASE 도구는 그림 15 와 같이 사용자 인터페이스, 자료 처리, 데이타 베이스 세 계층으로 구성되어 있다. 사용자 인터페이스 계층은 사용자에게 ASADAL 의 세 가지 명세 언어를 작성할 수 있는 다이어그램 편집기와 작성된 명세들간의 행동을 동적으로 보여 주는 시뮬레이션 인터페이스를 제공하여 준다. 자료 처리 계층은 사용자 인터페이스 계층으로 부터 받은 각종 자료를 처리하여 사용자에게 보여 주거나 데이타베이스에 저장하는 역할을 한다. 마지막으로 데이타베이스 계층은 명세에 나타난 정보

를 미리 정의된 스키마 정보에 따라 저장하고 상위 계층에서의 요구가 있을 때 이를 데이타 베이스에서 추출하여 전달해 주는 역할을 한다. 이러한 계층적인 구조는 독립적인 시스템 개발 환경을 제공해 줄 뿐만 아니라 새로운 환경 변화에 시스템이 쉽게 적응할 수 있는 장점이 있다. 다음 절에서는 이러한 특징을 가지는 ASADAL 의 내부 구조를 계층별로 설명한다.

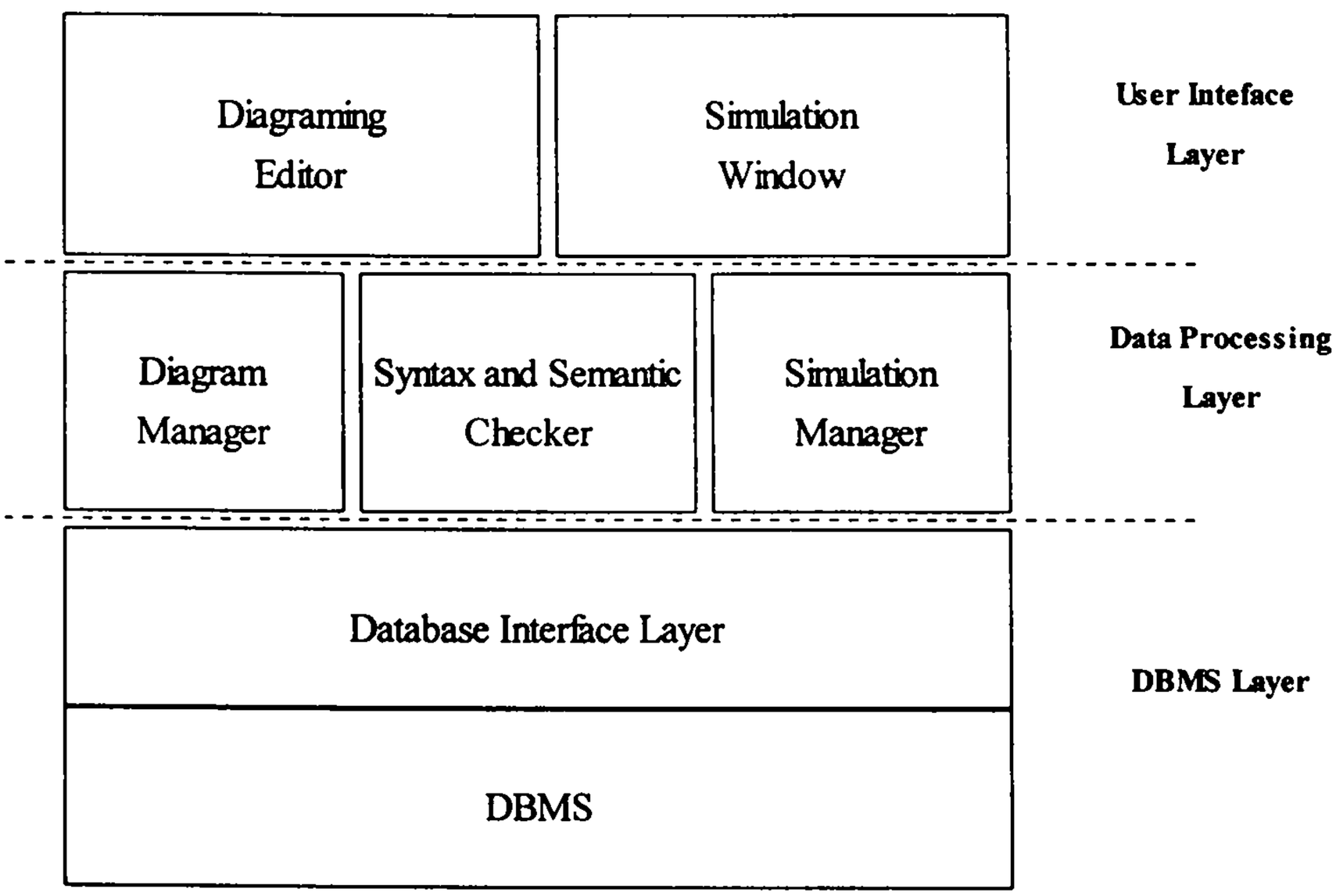


그림 15. ASADAL CASE 도구 시스템 구조도

가. 사용자 인터페이스 계층

사용자 인터페이스 계층은 사용자와 상호작용을 하는 인터페이스로서 사용자로부터 ASADAL 의 명세언어인 RTET, PDFD, TES 명세를 작성할 수 있는 환경을 제공해 주는

다이어그램 편집기와 작성된 명세의 동적인 행동을 사용자와 상호작용을 하면서 보여 주는 시뮬레이션 인터페이스가 있다.

사용자 인터페이스 계층은 사용자와 대화하는 다양한 윈도우를 생성하고 이 윈도우를 바탕으로 사용자의 입력을 받아 이를 자료 처리 계층에 전달하여 주고 자료 처리 계층에서 처리된 결과를 받아 화면에 보여 주는 역할을 한다.

사용자와 대화하는 다양한 윈도우를 간편히 생성시키기 위해서 사용자 인터페이스 계층에서는 윈도우를 구성하는 윈도우 구성요소를 상위 단계의 추상화 모듈로 제공한다. 이러한 상위 단계의 추상화 모듈을 이용하여 CASE 도구 제작자는 새로운 윈도우를 설계하고 구현하는데 많은 노력을 들일 필요가 없게 된다.

또한 사용자 인터페이스 계층에서는 특정 윈도우 시스템과 관련된 기능은 모두 이 계층에서 제공한다. 예를 들면 다이어그램 관리자에서 원, 사각형 혹은 직선 등의 모양을 가지는 다이어그램 기본자를 그리기 위해서는 윈도우 시스템에서 제공하는 Graphic Library 를 사용하여야 한다. 만약, 자료 처리 계층에 있는 다이어그램 관리자에서 다이어그램 기본자를 그리기 위해 특정 윈도우 시스템의 Graphic Library 를 직접 사용하게 되면 후에 윈도우 시스템이 바뀔 경우 사용자 인터페이스 계층 뿐만 아니라 자료 처리 계층도 바뀌어야 한다. 그러므로 특정 윈도우 시스템과 관련된 모든 기능은 사용자 인터페이스 계층에서 제공하도록 하고, 자료 처리 계층에서 이러한 기능이 필요할 때는 사용자 인터페이스 계층에서 제공하는 함수를 이용하도록 한다. 이렇게 시스템의 기능을 계층별로 분류하여 놓고 다른 계층의 기능이 필요한 경우에 그 계층에서 제공하는 인터페이스 함수를 사용함으로써 시스템의 환경 변화에 쉽게 대처할 수 있다.

나. 자료 처리 계층

자료 처리 계층은 사용자 계층에서 받은 입력을 처리하여 결과를 사용자 인터페이스 계층으로 보내 주거나 데이타 베이스 계층에 결과를 보내는 역할을 한다. 자료 처리 계층은 다음과 같이 크게 세가지 구성 요소로 구분된다.

- 다이어그램 관리자

  다이어그램 관리자는 다이어그램 편집기로 부터 들어오는 입력에 따라 다이어그램 기본자를 처리하는 기능을 한다. 즉, 방법론에서 지원하는 명세 언어를 그림적인 형태로 나타낼 수 있도록 다이어그램 기본자를 처리한다. 다이어그램 기본자를 처리하는 기능으로는 다이어그램 기본자의 생성, 삭제, 움직임, 크기 조절, 저장, 로드 등이 있다.

- 문법과 의미 검사기

  문법 검사기는 명세 언어의 그림적인 형태인 다이어그램의 문법적 정확성과 의미의 일관성을 검증해 주는 기능을 한다. 문법 검사기는 사용자가 원하는 시점에 검사가 가능하도록 하며, 데이타 베이스 계층으로 부터 명세와 관련된 정보를 읽어 와 이의 문법적 정확성과 의미적 일관성을 검사하여 사용자 인터페이스 계층에 이의 결과를 보내 준다.

- 시뮬레이션 관리자

  시뮬레이션 관리자는 사용자로 부터 선택된 명세에 대한 시뮬레이션을 해 준다. 시뮬레이션의 절차는 다음과 같다. 먼저, 사용자는 시뮬레이션을 수행하기 원하는 다이어그램들을 선택하고 시뮬레이션의 범위를 정한다. 그런 다음 사용자가 각종 환경 변수를 지정해 주고 사건을 발생시킴으로써 시뮬레이션을 진행해 나간다. 이러한 모든 절차는 시뮬레이션 관리자에 의해 제어된다.

다. 데이타 베이스 계층

이 계층은 명세의 정보를 저장하고 있는 계층으로 다시 두 부분으로 나뉜다.

- Database Interface Layer

  데이타 베이스 인터페이스 계층은 다이어그램에 그려진 그래픽 객체를 데이타 베이스에 저장할 때 사용되는 인터페이스 모듈로서, 일반적인 데이타 베이스 접근 함수를 제공함으로써, 실제로 사용되고 있는 데이타 베이스가 바뀌어도 상위 계층에는 영향을 미치지 않도록 고안된 계층이다.

- DBMS Layer

  이 계층은 실제 DBMS 에 해당하는 것으로 관계형 DB, 네트워크 DB, 객체지향 DB 등 모든 것을 다 사용할 수 있으나, 현재 ASADAL CASE 도구에서 사용하는 DBMS 는 UniSQL 로 객체지향 DB 이다.

3. ASADAL CASE 도구의 설계

ASADAL CASE 도구는 새로운 방법론의 추가 지원이나 기존 방법론의 수정 및 확장이 쉽게 이루어 질 수 있고 가능한 한 재사용할 수 있는 부분이 많게 설계되어 비슷한 종류의 새로운 시스템을 개발하는 데 있어 생산성을 향상시킬 수 있다. 본 절에서는 이러한 요구 사항을 만족시키기 위해서 ASADAL CASE 도구를 객체 지향 방식으로 설계하고 구현하는 방법에 대해 기술한다.

일반적으로 객체 지향 접근 방식은 추상화에 의한 복잡성의 해결에 많은 도움을 줄 뿐만 아니라, 확장성, 유지 보수성, 재사용성 등의 장점이 있다. 예를 들면, 그래픽 편

집기의 개발에 있어서 객체 지향 방식을 사용하면 시스템을 모델링하기가 쉽고 소프트웨어의 변화 가능성을 특정 클라스에만 한정시킬 수 있기 때문에 유지 보수성을 높이는 효과를 가져다 준다. 즉, 고유한 모양을 가지는 그래픽 기본 요소 각각을 하나의 객체로 대응시키고 이들 그래픽 기본 요소를 다루는 오퍼레이션을 객체의 메소드(Method)로 하여 객체의 정보를 객체 내에 감추도록 설계하면 된다. 또한, 공통된 행동이나 의미를 갖는 부분을 일반화하여 추상화된 상위 클라스를 만들고 고유한 영역을 차지하는 부분은 특정한 종속 클라스로 정의함으로써 후에 새로운 요소를 첨가할 때 공통인 부분은 상속을 이용하여 재사용하고 특정한 부분만 새로이 정의함으로써 확장성과 재사용성을 증대시킬 수 있다.

이러한 소프트웨어 시스템의 확장성, 유지보수성, 재사용성을 고려한 시스템의 설계 방법에 대해 사용자 인터페이스, 다이어그램 관리자, 문법과 의미 검사기, 시뮬레이션 시스템, 데이타 베이스 인터페이스 시스템의 순서로 설명한다.

가. 사용자 인터페이스 설계

이 절에서는 사용자 인터페이스를 설계할 때 고려한 윈도우 구성요소의 재사용과 특정 윈도우 시스템의 정보를 감추는 추상화 머신의 개념을 실질적으로 적용한 설계 방법에 대해서 기술한다.

- 윈도우 구성 요소를 재사용하기 위해 객체 지향 방식의 윈도우 시스템을 설계한다. 윈도우를 구성하는 구성요소 각각을 하나의 객체에 대응시켜 새로운 윈도우를 설계할 때, 이 윈도우 구성요소에 해당되는 객체를 조합하여 만든다. 이러한 접근 방법은 단순 반복적인 윈도우 설계 작업에 상위 단계의 추상화를 제공하여 주기 때문에 개념적으로 명쾌하고 실질적으로 설계하기에 간편한 설계 방식이다.
- 특정한 윈도우 시스템에 의존적인 설계를 하면 미래에 다른 윈도우 시스템으로

변경될 경우에 특정 윈도우의 인터페이스를 사용하는 모든 모듈을 다시 설계하여야 하는 단점이 있다. 그러므로 특정 윈도우 시스템의 정보를 감추는 추상화된 윈도우 시스템을 설계하는 것이 다양한 외부 환경 변화에 쉽게 대처할 수 있는 방법이다.

위에서 말한 설계 방안을 실질적으로 적용한 사례는 다음과 같다.

먼저, 재사용 가능한 윈도우 구성요소를 찾기 위해서 사용되는 사용자 인터페이스의 윈도우 구조를 파악하여 본다. 그림 16 은 사용자 인터페이스에서 사용되는 윈도우 구조의 한 예를 나타낸 것이다. 여기서 재사용 가능한 윈도우의 구성 요소로는 Menubar, Iconbar, Workarea, 등이 있다. 즉 쉘(Shell)위에 이와 같은 윈도우 구성 요소를 원하는 위치에 배치시킴으로써 간단히 윈도우를 설계할 수 있다.

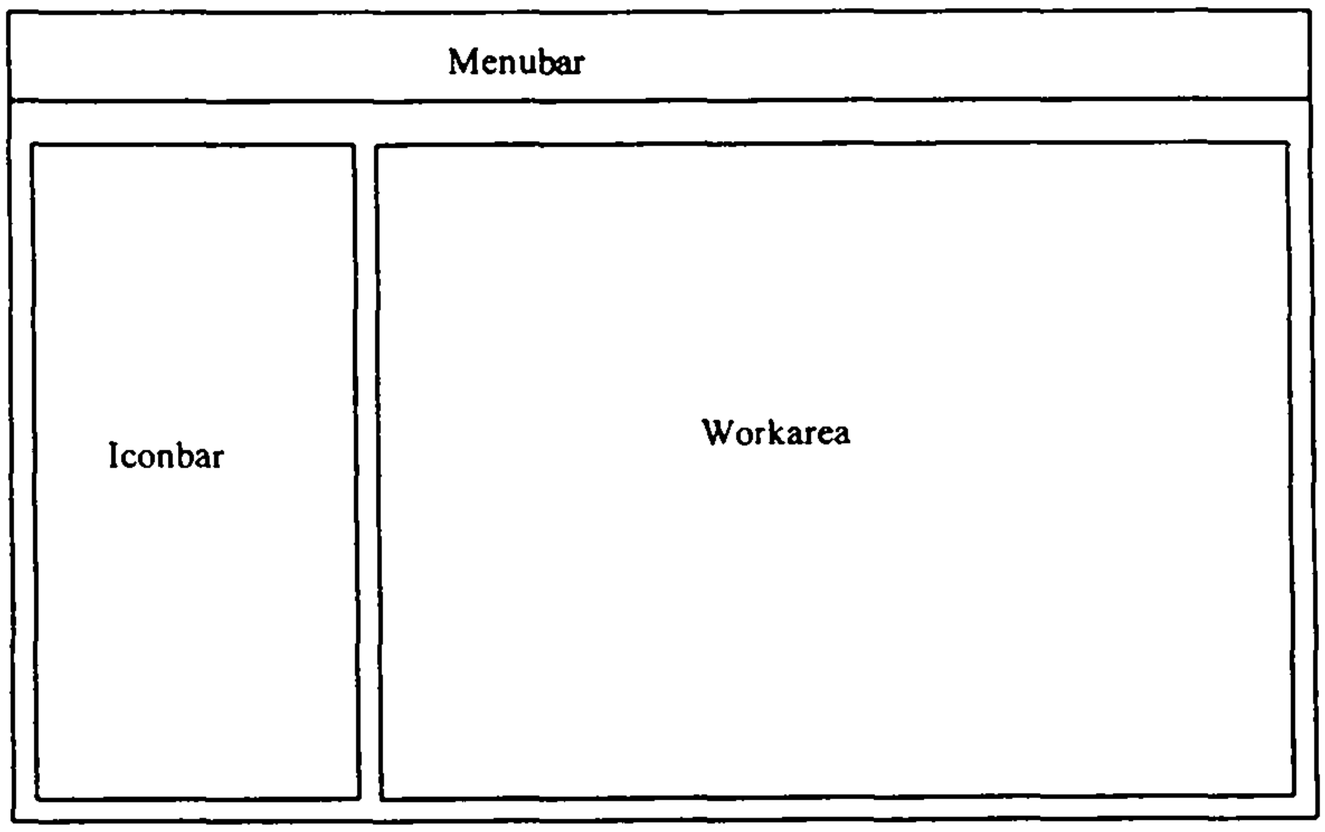


그림 16. 사용자 인터페이스 윈도우 구성의 한 예

윈도우 구조로 부터 재사용 가능한 요소로 파악된 윈도우의 각 구성요소를 하나의

클라스로 설계한다. 각 클라스는 윈도우 구성 요소를 화면에 나타내는 메소드와 그 윈도우 구성 요소가 가져야 하는 특정의 메소드를 가진다. 그림 17은 Menubar 윈도우 구성 요소를 클라스로 설계한 것이다.

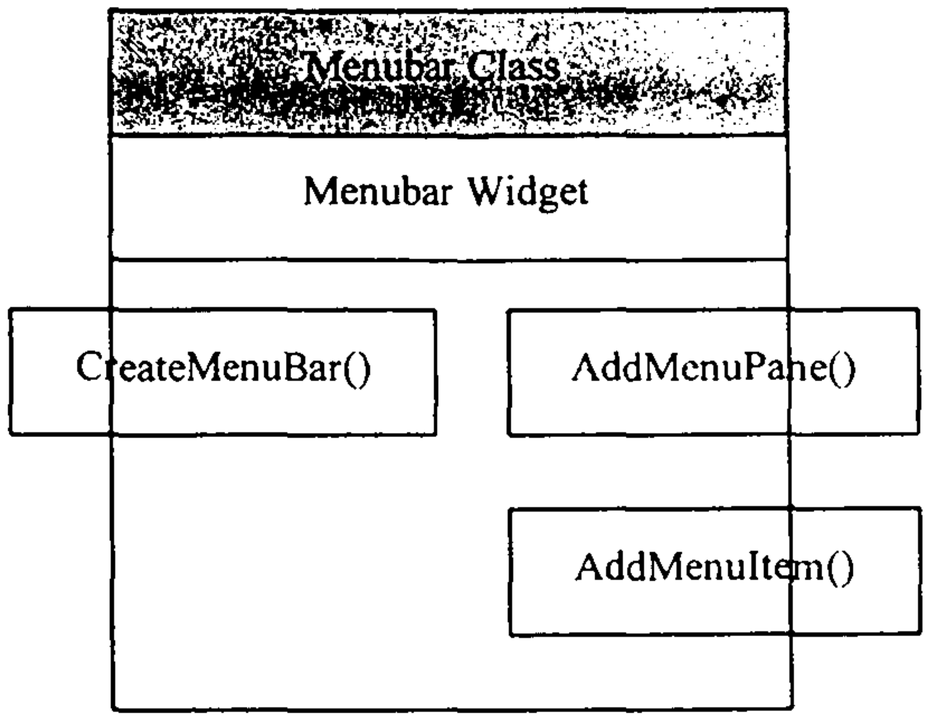


그림 17. Menubar Class 구조

이와 같이 윈도우 구성요소 각각을 클라스로 정의하여 추상화해 놓음으로써 윈도우 설계자는 이 클라스들을 이용하여 상위 단계의 윈도우 설계만으로 쉽게 윈도우를 생성시킬 수 있다. 뿐만 아니라 새로운 윈도우 시스템으로 바뀌는 경우에도 상위 단계의 인터페이스는 변하지 않고 클라스 내부의 구현 부분만 변화되기 때문에 이러한 상위 단계의 추상화된 클라스를 사용하는 다른 모듈은 변경할 필요가 없다.

나. 다이어그램 관리자의 설계

이 절에서는 다이어그램 편집기의 핵심적인 부분인 다이어그램 관리자가 명세 언어의 변화 등의 변화 가능성에 대처하기 위해 어떻게 설계되어져야 하는 가를 기술한다.

다이어그램 관리자는 방법론에서 제공하는 명세 언어의 의미와 이를 그래픽 칼하게 표현한 표기법에 따라 다이어그램 관리자의 행동이 달라지게 된다. 그러므로 방법론에서의 의미가 변하거나 표기법이 변하였을 때, 다이어그램을 구성하는 기본 요소인 다이어그램 기본자를 설계할 때, 다이어그램 기본자의 의미 정보와 그림 정보를 분리하여야 한다.

예를 들면, PDFD 에서 시스템의 기능을 나타내는 기능자는 자료 흐름자와 연결되어 자료의 생성자가 될 수도 있고 소비자가 될 수도 있다. 이러한 의미 정보는 다이어그램 기본자의 그림적인 형태를 나태 내는 그림 정보 - 기능자의 모양은 사각형 - 와 분리되어야 한다. 왜냐하면 명세 언어의 의미 정보가 바뀌는 경우에는 의미 정보를 가지고 있는 부분만 변경하면 되고 그림 정보가 바뀌는 경우에는 그림 정보를 가지고 있는 부분만 변경하면 되기 때문이다.

이러한 기본 철학을 가지고 다이어그램 관리자를 설계하면 그림 18 과 같은 클라스 구조의 다이어그램 관리자를 얻을 수 있다. 그림 18 에서 하나의 Diagram 클라스는 여러 개의 Diagram Element 클라스와 hold and manage 관계에 있고, Diagram 클라스와 Diagram Element 클라스 모두는 Controller 클라스에 의해 제어되는 관계에 있다. 그리고 하나의 Diagram Element 클라스는 그것의 그림 정보를 나타내는 Shape_Control 클라스를 가진다. Shape_Control 클라스는 기본적인 모양일 수도 있고 여러 가지 모양이 결합된 복잡한 모양일 수도 있으므로, 이 Shape_Control 클라스는 기본적인 모양을 나타내는.Shape 클라스를 가지게 된다.

그림 18 에서 나타난 Diagram Element 클라스와 Shape_Control 클라스는 다이어그램 기본자의 의미 정보와 그림 정보를 분리시키기 위해서 설계된 것이다. 즉, 다이어그램 기

본자의 의미 정보는 관계 있는 다이어그램 기본자들 간의 연결 정보를 나타내어지므로 이러한 연결 정보를 Diagram Element 클라스에 두고, 다이어그램 기본자의 그림 정보는 Shape_Control 클라스에서 가지고 있도록 설계된 것이다.

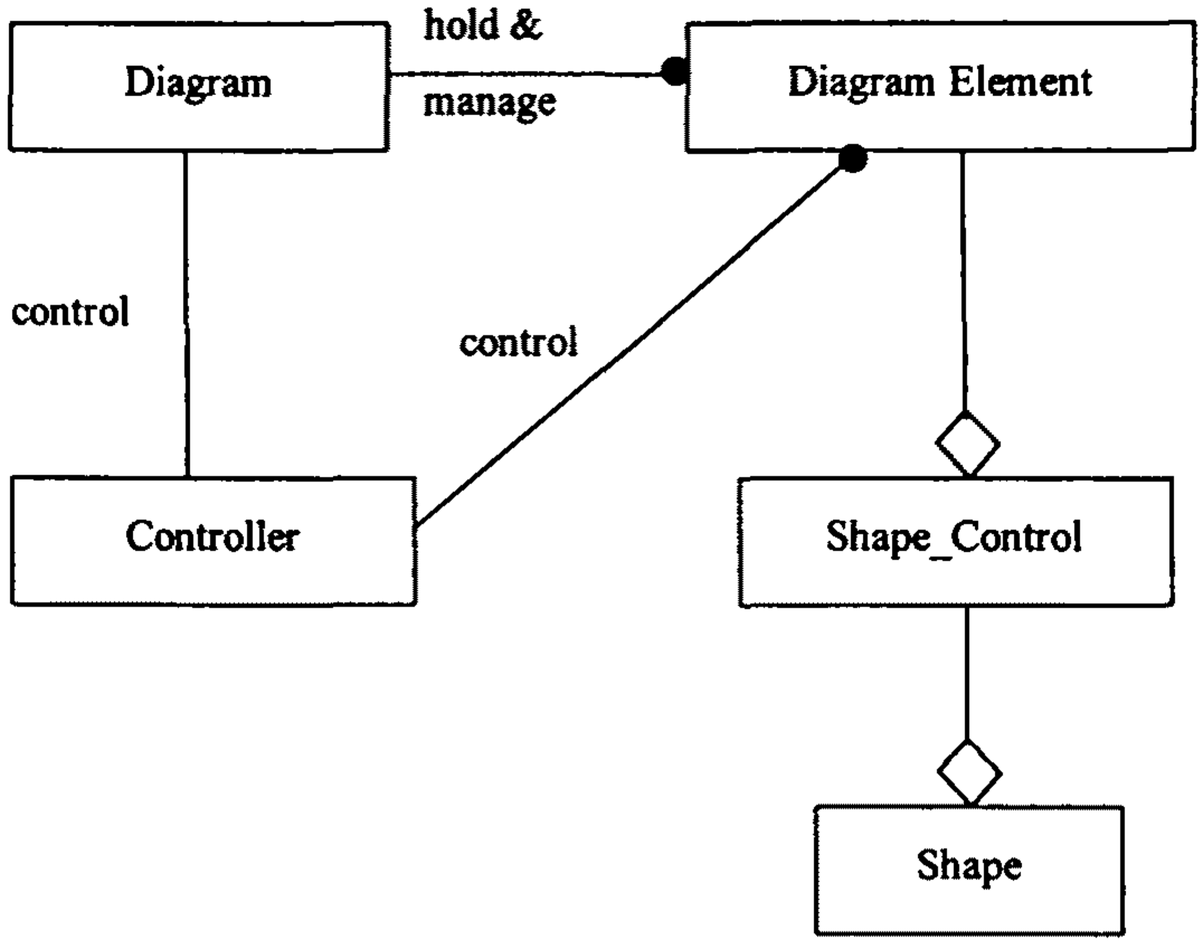


그림 18. 다이어그램 관리자의 일반적인 구조

다. 문법과 의미 검사기의 설계

이 절에서는 문법과 의미 검사기를 설계할 때 명세 언어의 변화에 따른 문법과 의미 검사기의 변화를 최소화하고 기존의 많은 모듈을 재사용할 수 있도록 설계하는 방법에 대해 기술한다.

문법 검사기의 설계 시 가장 중요한 것은 검사할 명세 언어 문법의 변화 가능성이다. 이는 CASE 도구가 한가지의 방법론만을 지원하는 것이 아니라 여러 가지 방법론

을 지원하며, 또한 각 방법론들은 여러 상이한 명세 언어를 가진다는 사실에 기인한다. 이런 명세 언어의 변화 가능성을 고려하여 문법 검사기를 설계하면 명세 언어가 바뀔 때마다 새로운 문법 검사기를 만들어야 하는 수고를 감소시킬 수 있다.

ASADAL 에서는 문법 검사기를 Context, Concept, Content 의 세 부분으로 나눔으로써 다양한 변화 가능성에 대처할 수 있게 하였다. Context 는 각 명세 언어의 문법을 Context Free Grammar 형태로 정의한 것이고 Concept 는 문법 검사기의 내부 구조를 정의한 것이다. 이러한 구조를 가지게 되면 만약 명세 언어의 문법이 바뀔 경우 명세 언어의 문법을 기술한 Context 를 변경시켜서 이 Context 를 바탕으로 새로운 문법 검사기를 사례화(Instanciation)할 수 있다.

라. 시뮬레이션 시스템의 설계

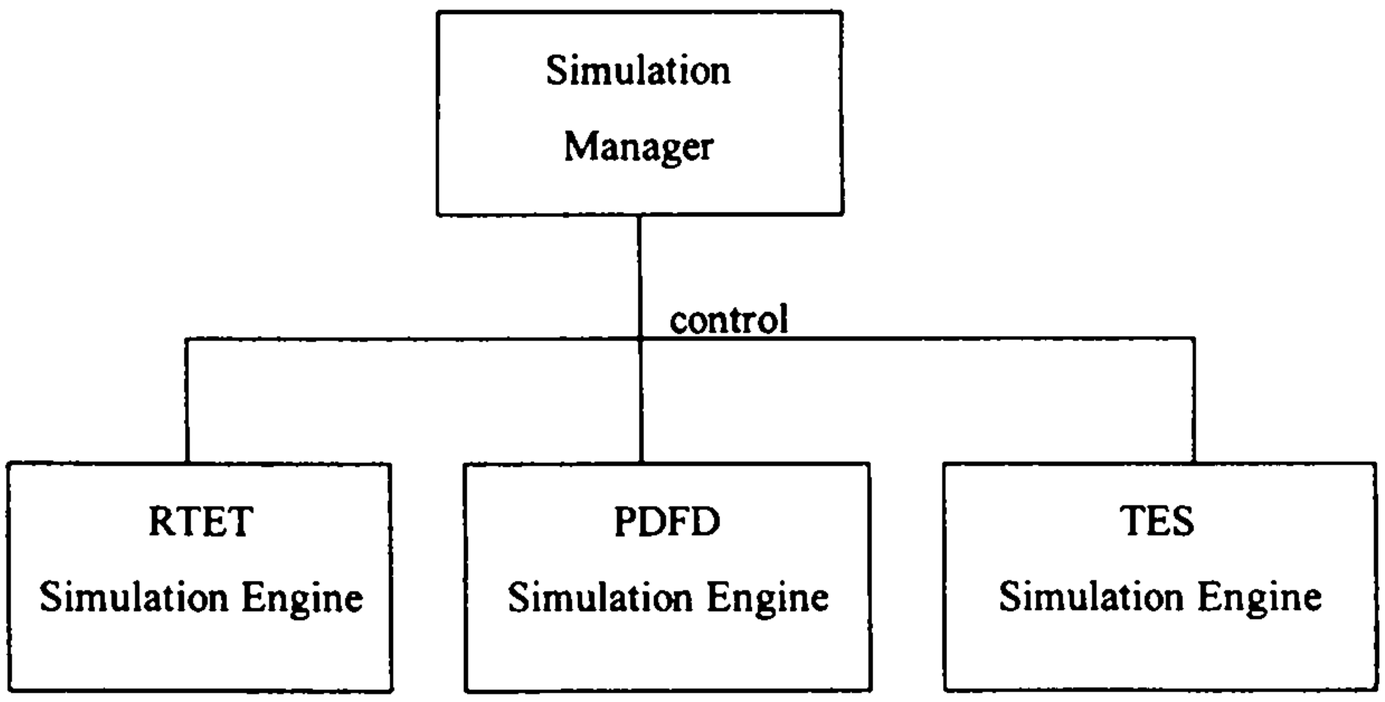


그림 19. 시뮬레이션 시스템의 구조

시뮬레이션은 명세에 따라 다르게 진행된다. 즉 RTET, PDFD, TES 의 세가지 명세를 시뮬레이션 하기 위해서는 이들 각각의 동적인 행동을 시뮬레이션할 시뮬레이션 엔진이 필요하게 된다. 또한, ASADAL 시뮬레이션은 시뮬레이션의 범위와 종류에 따라 TES 다이어그램의 시뮬레이션, PDFD 와 TES 의 시뮬레이션, RTET 과 결합된 시뮬레이션 등이 있다. 그러므로 이러한 다양한 시뮬레이션을 수행할 수 있는 시뮬레이션 시스템을 만들기 위해서 시뮬레이션 엔진을 제어하는 시뮬레이션 관리자를 두어야 한다. 이에 대한 구조는 그림 19 와 같다.

시뮬레이션 관리자의 역할은 시뮬레이션 범위를 정하고 시뮬레이션을 수행할 다이어그램을 선택하여 시뮬레이션을 위한 윈도우를 띄우고 각 시뮬레이션 엔진을 제어하여 시뮬레이션을 진행해 나가는 일이다. 그러므로 시뮬레이션 관리자를 설계하기 위해서는 시뮬레이션 관리자를 하나의 클라스로 두고 시뮬레이션 관리자의 역할을 클라스의 메소드로 설계하면 된다.

마찬가지로 각 다이어그램의 시뮬레이션 엔진을 설계할 때도, 각각의 시뮬레이션 엔진을 하나의 클라스로 두고 각 시뮬레이션 엔진의 역할을 그 클라스의 메소드로 정의함으로써 설계가 가능하다. 시뮬레이션 엔진의 역할은 다음과 같다.

- 데이타 베이스로 부터 명세를 읽어와 시뮬레이션을 수행하기에 적합한 시뮬레이션 구조를 만든다.
- 시뮬레이션 언어를 해석하여 시뮬레이션을 수행할 수 있어야 한다.
- 외부 환경 변수나 내부 환경 변수를 지정하거나 액세스할 수 있어야 한다.
- 시뮬레이션 진행 명령어에 따라 시뮬레이션을 진행해 나간다.

이와 같이 각 다이어그램에 대한 시뮬레이션 엔진을 만들고 이 시뮬레이션 엔진을

제어하는 시뮬레이션 관리자를 만듦으로써 새로운 형태의 시뮬레이션을 수행하기 위해서는 시뮬레이션 관리자에서 각 시뮬레이션 엔진을 제어하는 부분만 첨가시키면 된다.

마. 데이타 베이스 인터페이스 시스템의 설계

이 절에서는 일반적인 데이타 베이스 인터페이스를 제공하는 데이타 베이스 인터페이스 시스템의 설계에 대해서 기술한다.

데이타 베이스 인터페이스 시스템은 하위의 데이타 베이스 시스템을 감추고 일반적인 데이타 베이스 인터페이스 함수를 제공하는 시스템이다. 일반적인 데이타 베이스 인터페이스 함수를 제공함으로써 하위 데이타 베이스 시스템이 변경되는 경우에도 상위 단계에 있는 다른 모듈들은 변경할 필요가 없게 된다.

그러므로 먼저, 데이타 베이스 시스템이 가지는 일반적인 기능을 찾아 이를 일반적인 인터페이스로 제공하는 것이 필요하다. 다음은 데이타 베이스 시스템이 가지는 일반적인 기능을 나타낸 것이다.

- 데이타 베이스 연결 함수를 사용하여 데이타 베이스와 연결을 한다.
- 스카마 정의 함수를 이용하여 저장할 데이타의 스키마를 정의한다.
- 데이타 베이스의 데이타를 접근하는 함수는 크게 데이타를 읽어오는 함수와 데이타를 저장하는 함수, 해당되는 데이타를 삭제하는 함수, 해당되는 데이타를 변경하는 함수 등이 있다. 이러한 함수는 이미 정의된 스키마에 의거하여 수행되어 진다.
- 데이타 접근 함수를 사용하여 변경된 내용을 데이타 베이스에 실질적으로 적용시키기 위해서는 트랜잭션 승인 함수를 사용하고 이를 취소하기 위해서는 트랙잭션 취소 함수를 사용한다.

그러므로 이러한 일반적인 데이타 베이스 기능을 수행하기 위한 일반적인 인터페이스는 다음과 같은 것이다.

- 데이타 베이스 연결 함수 - Connect, Shutdown.
- 스키마 정의 함수 - CreateSchema, DeleteSchema, UpdataSchema.
- 데이타 접근 함수 - PutData, GetData, DeleteData, UpdateData.
- 트랜잭션의 승인 및 취소 함수 - Commit, Rollback.

이와 같은 일반적인 데이타 베이스 인터페이스 함수를 제공하는 시스템의 설계는 이상의 기능을 메소드로 제공하는 클라스를 설계함으로써 가능하다. 즉 데이타 베이스를 연결하는데 사용되는 메소드, 스키마를 정의하는데 사용되는 메소드, 데이타를 접근하기에 사용되는 메소드, 트랜잭션의 승인 및 취소를 위한 메소드들을 데이타 베이스 인터페이스 클라스의 메소드로 정의한다. 이렇게 특정 데이타 베이스와 관련된 부분을 이 클라스에 한정시킴으로써 외부 데이타 베이스 시스템이 바뀌는 경우에 이 클라스의 메소드 구현 부분만 변경하도록 설계되었다.

6 절 결론

본 보고서에서는 사용자의 요구사항을 받아들여 시스템 모델을 만들고 이 시스템 모델이 사용자의 요구사항을 만족하는 지를 분석하고 검증하는 방법으로 모델을 수행시키는 방법을 제안하였다.

ASADAL 은 다음과 같은 장점을 가지고 있다.

- 시스템 모델의 기능적 측면과 행동적 측면을 명확히 분리함으로써 모델을 만들

기가 쉽고 분석하기가 쉽다.

- 실시간 시스템의 다양한 시간적, 기능적 행동을 쉽게 표현할 수 있는 시뮬레이션 언어를 제공하고 이를 시뮬레이션 해 봄으로써 시스템의 다양한 분석이 용이하다.
- 시스템 모델이 사용자의 요구사항을 만족하는 지를 개발 초기에 검사함으로써 시스템의 개발 비용을 감소시킨다.
- 시스템의 행동을 관찰함으로써 최종 산출물의 모습을 보고 느낄 수 있다.

ASADAL 방법론은 CASE 도구로 개발, 확장 중에 있다. CASE 도구는 사용자의 요구사항을 받아들여 시스템의 모델을 쉽게 만들 수 있도록 편리한 사용자 인터페이스를 제공하는 그래픽 칼한 명세 작성기를 제공한다. 작성된 명세는 문법적 정확성과 의미적 일관성을 검사하는 문법.의미 검사기를 통해 명세의 기본적인 명세 오류를 찾아내어 준다. 문법적으로나 의미적으로 완전한 명세는 ASADAL 시뮬레이터를 통해 모델을 수행시켜 볼 수 있다. 다양한 시뮬레이션을 통해 모아진 시뮬레이션 데이타를 바탕으로 모델의 다양한 분석뿐만 아니라 실시간 시뮬레이션도 가능하다.

ASADAL 방법론과 CASE 도구는 복잡한 행동 양식을 가지는 실시간 시스템의 분석에 용이한 특징을 가지고 있다. Safety Home 시스템, Cruise Control System, Avionics System 등의 사례연구를 통해 ASADAL 방법론의 유용성을 검사 받았다. 특히 시스템 모델이 사용자의 요구 사항을 만족하는 지를 모델을 수행시켜 봄으로써 초기에 알아볼 수 있으므로 모델의 오류를 초기에 고칠 수 있어 전체 시스템 개발 비용을 감소시킬 수 있다.

제 3 장 다중 우선순위 스케쥴러 및 병렬 로봇 시뮬레이터에 관한 연구

1 절 서 론

빠른 응답시간과 예측가능성은 실시간 시스템의 기본 조건들이다. 특히 실시간 시스템은 한 event 가 발생했을때, 그것을 매우 빨리 알아차리고 deterministic 한 행동을 취할 수 있어야만 한다. 여기서 deterministic 하다는 의미는 그 행동의 기능과 시간이 잘 정의되어 있다는 것을 의미한다. 이런 실시간 시스템을 지원하는 운영체제는 하드웨어 인터럽트와 소프트웨어 인터럽트 모두에 대하여 잘 대응할 수 있어야 한다. 운영 체제 그 자체도 또한 인터럽트당할 수 있어야만 하고 그 인터럽트 후에 재진입이 가능해야 한다. 특히, high-priority process 가 실행하려고 할 때에는 운영 체제에서는 최소한의 overhead 를 보장해야 한다. 프로세싱의 예측가능성과 빠른 선점을 제공하려면 응용프로세스들의 생성, 소멸, 스케쥴링을 담당하는 스케쥴러측면에서 다중 우선순위 스케쥴링을 지원해야 한다. 보다 신속한 선점, 최소한의 overhead 와 예측가능한 실행시간을 구현하기 위해 많은 스케쥴링 알고리즘들이 제안되고 있다.

또한 실시간 시스템은 프로세스간의 통신, 데이타 획득, 입출력 지원 등의 기능을 제공할때 빠른 응답시간을 필요로 한다. 일반적으로 실시간 프로그램들은 많은 입출력 인터페이스들을 포함하기 때문에 이들의 빠른 응답시간은 성능향상에 도움을 줄 뿐 아니라 응용프로그램들이 주어진 요구시간 내에 작업을 처리하는데 많은 영향을 미친다. 이 외에도 사용자가 우선순위, 스케쥴링 정책 등을 선택할 수 있게 함으로써 CPU 에 대한 제어기능을 제공하며 실시간 타이머등의 기능을 사용자에게 제공하여 사용자가 실행시간을 예측할 수 있게 해야 한다. 분산, 병렬 시스템은 빠

른 응답시간을 제공하기 위한 적절한 방법으로 본 연구에서는 트랜스퓨터를 기반으로 하는 병렬 컴퓨터 TiME 을 사용한다.

TiME 의 프로세서인 트랜스퓨터는 하드웨어 스케쥴러를 내장하고 있어서 프로세스 관리능력이 뛰어난 반면 하드웨어적으로 두 단계의 우선순위만 지원하고 있어서 실시간 스케쥴링에 적합하지 않다. 본 연구에서는 트랜스퓨터 상에서 실시간 시스템을 구현하기 위해서 다중 우선순위 스케쥴러를 소프트웨어적으로 구현했다. 다중 우선순위 스케쥴러는 주기적으로 호출되지 않고 프로세스 큐가 바뀔 때에만 호출되게 함으로써 스케쥴러의 overhead 를 줄였다. 스케쥴러 자체는 프로세스 큐가 바뀌는 것을 알지 못하므로 사건 관리기가 스케쥴러를 호출하게 하였다.

본 연구에서는 병렬 시뮬레이터를 구현하기 위한 기초작업으로 로봇 시뮬레이터를 구현하였다. 로보트 시뮬레이터는 병렬 컴퓨터 TiME 상에서 여러개의 로봇 팔이 동시에 움직이는 것을 시뮬레이션 한다. 시뮬레이터를 병렬 시스템에서 구현할 때는 시뮬레이션 환경이 복잡해지더라도 이들 환경을 처리하는 루틴들이 여러 프로세서에 나누어져서 실행되므로 실행시간이 크게 증가하지 않는다는 장점이 있다.

구현된 로봇 시뮬레이터는 각 로봇 팔에 대해 2 개씩의 프로세서를 할당함으로써 32 개의 프로세서가 있는 TiME 환경에서 최대 16 개의 객체를 동시에 시뮬레이션할 수 있다.

2 절 병렬 시스템에서의 다중 로봇 시뮬레이션

1. 개 요

그래픽 로봇 시뮬레이션에서 많은 kinematics 계산과 그래픽 계산을 위한 많은 변환들은 계산의 복잡성을 초래한다. 병렬컴퓨터를 이용할 때 얻어지는 계산능력의

증대는 다음과 같은 방법으로 이용될 수 있을 것이다.

- 시뮬레이션 시스템의 기능을 강화 시킬 수 있다. 즉 향상된 계산능력을 이용하여 시스템 내에 동력학적 계산과 충돌 감지 기능을 포함시킬 수 있다.
- 많은 수의 로봇들로 이루어진 더 복잡한 환경들에 대한 시뮬레이션이 가능하다.

특히 점점 복잡해지고 있는 제조업 현장에서의 로봇 응용에 대한 시뮬레이션을 위해 위의 두번째 측면은 더욱 중요해지고 있다. 실제로 로봇 시뮬레이션을 병렬 시스템에서 구현한 Hubertus[1]의 예가 있으며, 본 연구는 Hubertus 의 논문을 근간으로 하여 TiME 에 맞게 재구성 한다. 이 절에서는 시뮬레이션에서 병렬성을 이용하기 위한 타스크의 분할과 각각의 구현에 대해 알아보기로 한다.

2. 시뮬레이션 타스크 분할

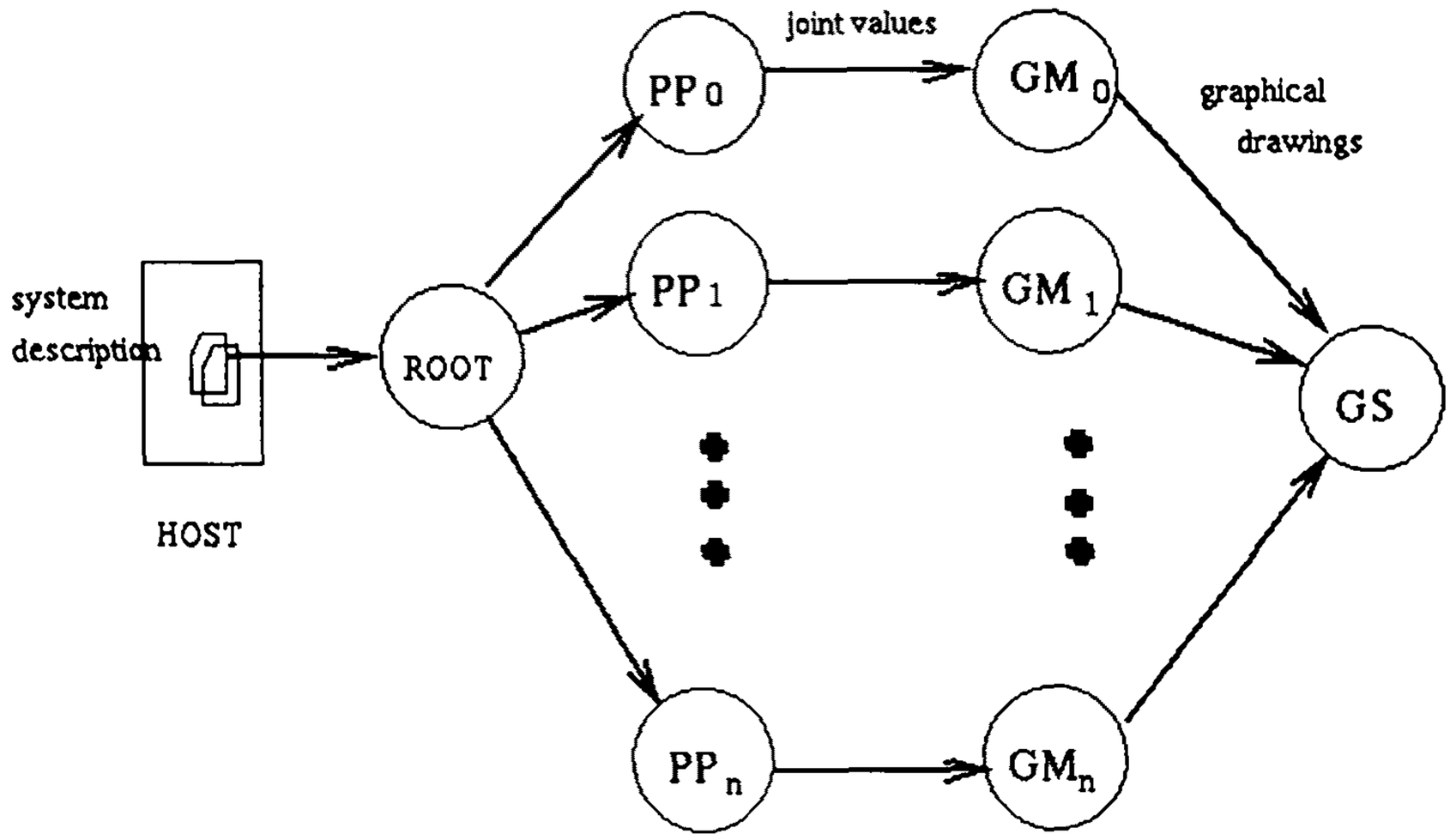
타스크 분할의 목표는 각 타스크의 병렬성을 이용하여 성능의 향상을 기하는 것이다. 하나의 로봇의 시뮬레이션을 위해 다음과 같은 기능들이 필요하다.

- Kinematics (Path-Generation) : 이것의 기능은 단순한 직선의 움직임 혹은 점과 점 사이의 움직임을 interpolation 을 통하여 각 로봇에 따라 주어지는 제한요소들을 만족시키는 joint 변수들의 연속으로 변환 시키는 것이다. Kinematics 는 interpolation 과 inverse kinematics subtask 들로 나눌 수 있다.
- 그래픽-Computaion : 그래픽 시뮬레이션을 위해 각 로봇의 링크(link)들은 원통형이나 box 의 형태로 그려지게 된다. 각 링크 들에는 Denavit-Hartenberg 표기법에 의해 정의된 좌표계가 설정되어 있고, 이 좌표계에 대한 좌표로 각 링크들은 묘사된다. 로봇의 자세가 변할 때마다 각 링크의 위치가 수정되어야 한다. 이 수정은 일련의 행렬(matrix)과 벡터(vector)간, 행렬과 행렬간 계산을 통해 이루어 진다. 또한 view-coordinate 시스템에 대한 좌표로의 변환을 위한 특별한 그래픽을 위한 변환이 수행되어야 한다.

일단 그래픽을 위한 계산이 완료되면 각 기본 그래픽 개체 (graphic primitive)에 대한 데이타는 그래픽 서버로 보내지며 그래픽 서버에서는 draw_polygon()등을 이용하여 직접 윈도우상에 그림을 그리게 된다. 모든 로봇에 대한 drawing 이 완료되면 윈도우는 refresh 되어 새로 변화된 로봇들에 대한 그림을 화면상에 출력한다. 이러한 방법으로 애니메이션을 수행하게 된다. 여기서는 병렬 시스템상에서 로봇 시뮬레이션을 구현하기 위해 세 단계로 타스크를 분할하여 전체적인 로봇 시뮬레이션을 수행하도록 하였다. 한 개의 로봇에는 위에서 언급된 시뮬레이션 kinematic, 그래픽 계산 등의 기능이 요구되는데 이들 타스크들을 분리하여 하나의 프로세서에서 kinematic 계산, 하나의 프로세서에서 그래픽 계산을 수행하도록 하였다. 많은 수의 로봇이 시뮬레이트되는 경우는 각각 로봇 마다 kinematic, 그래픽 계산을 위한 프로세서들이 2 개씩 할당되어 수행된다. 또한 시뮬레이션을 수행하고 host 와 interaction 을 취하기 위한 root, 각 로봇의 시뮬레이션 과정에서 계산된 그래픽 데이타 들이 보내져서 화면상에 디스플레이하는 그래픽 서버로 이루어 진다. 현재 시뮬레이션에서 타스크의 분할과 타스크간 통신은 **그림 1**과 같이 이루어 진다.

3. 시스템 서술

시뮬레이터를 이용하기 위해 사용자는 로봇 팔의 기하학적 형태, kinematics, 타스크의 할당 등을 정의하는 파일을 작성하여야 한다. 커널 초기화가 이루어 진후 root node 에서 실행되는 loader 에 의해 system description file 의 내용이 읽혀진다. Transputer network 에서 타스크 할당을 지정하고 로봇의 형태와 kinematics 의 지정을 위한 syntax 는 다음과 같다.



PP_1 = Path Planner for Robot 1
GM_1 = Graphics Master for Robot 1
GS = General Purpose Graphics Server
*.cad = Robot System specification file

그림 1: 여러개의 로봇 시뮬레이션을 위한 2 단계의 시뮬레이션

우선 각 로봇들은 로봇 내의 링크 수와 그 로봇의 inverse kinematic 함수를 지정하기 위한 inverse kinematic 함수 table index 와 함께 정의 되어야 한다. 다음은 로봇 정의 syntax 이다.

```
ROBOTS
    <robot name>   < # of links in the robot> < inverse kinematic index> <comma>
    <robot name>   < # of links in the robot> < inverse kinematic index>
END
```

다음은 5 개의 link 를 갖고 manipulator_5_R_1_P() routine 을 inverse

kinematic routine 으로 이용하는 robot1 과 6 개의 링크를 갖고

fast_printer_device() routine 을 manipulator 로 이용하는 robot2 에 대한 정의 예이다.

```
ROBOTS   robot1 5 0 , robot2 6 1   END
```

다음으로 로봇을 화면상에 그리기 위한 그래픽 개체에 대한 정의를 수행해야 한다. 현재 로봇 링크의 그래픽을 위해 이용되는 형태는 원통형과 상자형 두 가지가 지원된다. 다음은 객체의 하나의 상자와 하나의 cylinder 로 이루어진 객체에 대한 정의 예이다.

```
OBJECT   arm1   coral
  BOX          2.0   2.0   6.0    0.0   0.0   -7.0   0.0   0.0   0.0
  POLYEDER     1.0   1.0   12     0.0   0.0   -1.0   0.0   0.0   0.0
       END
```

arm1 은 객체의 이름, coral 은 객체의 색깔을 나타낸다. 상자에 대한 정의는 상자의 길이(2.0), 폭(2.0), 높이(6.0), translation 좌표(x,y,z)=(0.0, 0.0, -7.0), 좌표계의 회전을 나타내는 오일러 각도 (ϕ, θ, φ) = (0.0, 0.0, 0.0) 등으로 이루어 진다. Cylinder 에 대한 정의는 keyword 인 POLYEDER, 지름(1.0), 높이(1.0), cylinder 를 선을 이용하여 그리기 위한 edge 의 갯수(12), translation 좌표 (x,y,z)=(0.0, 0.0, -1.0), 좌표계의 회전을 나타내는 오일러 각도 (ϕ, θ, φ) = (0.0, 0.0, 0.0) 등으로 이루어 진다. 객체의 기본요소(box, polyeder) 들은 오일러 좌표 (x, y, z, ϕ, θ, φ)에 의하여 정의된 좌표계에따라 그려진다. 오일러 각도 정의의 단위는 도이다. 그 이외 수치의 단위는 프로그램 내에서 디스플레이 윈도우를 생성하고 윈도우 상의 pixel 을 전역 좌표로 변환하는 과정에서 결정된다. 현재 디스플레이 윈도우 크기는 1000 pixel × 1000pixel 로 이루어져 있다. 또한 1000 × 1000 윈도우는 x 축으로 -23 에서 23, y 축으로 -23 에서 23 을 갖는 전역

좌표 윈도우로 변환 된다. 객체의 정의를 위해 사용된 값은 전역 좌표 상의 좌표값이다.

로봇의 kinematics 는 링크 정의의 연속으로 이루어 진다. 링크 정의의 문법은 다음과 같다.

```
LINKS   <robot name>
   BASE          <object name>      < world placement >
         PRISMATIC   <object name>      <DH_parameter>
                                        <joint_limitations>


   END
```

robot name 은 로봇 정의 부분에서 정의된 것 중의 하나이어야 한다. 링크 정의 부분은 링크의 joint type(prismatic or revolute), 링크로 이용되는 graphical object name, 이전 링크와의 관계를 기술하는 Denavit-Hartenberg 변수들의 값, joint value 의 최대, 최소값, joint 값에 의한 moving 을 수행할 때 moving 속도와 가속도의 값 등을 기술함으로써 이루어진다. 로봇들이 놓이게 되는 table 등과 같이 움직이지 않는 물체들에 대한 기술은 다음과 같은 syntax 에 의하여 이루어 진다.

```
WORLD <object name>   <translation   and rotation of attached frame> END
```

움직이지 않는 물체에 대한 정의에서는 물체에 설정된 좌표계의 회전과 변위를 나타내는 $(x, y, z, \phi, \theta, \varphi)$ 등만을 기술하면 된다.

시뮬레이션을 위해 하나의 로봇에 필요한 것은 kimatic 계산을 위한 pathplanner 와 그래픽 계산을 위한 그래픽 master 이다. 다음은 transputer network 상의 프로세서를 로봇의 pathplanner 와 그래픽 master 로 지정하는 syntax 이다.

```
SYSTEM
  <robot name>   <path planner> < graphic master>
        WORLD                                  <graphic master>
  END
```

path planner 와 그래픽 master 의 지정은 path planner(또는 그래픽 master)로 사용될 node 의 transputer network 상에서의 row, column 값을 지정한다. 다음은 다섯 개의 로봇 puma1, puma2, puma3, puma4, table 과 static object 로 이루어진 시스템에서 각 타스크를 위해 proess 를 할당한 예이다.

```
SYSTEM
        puma1   0 0   1 0
        puma2   0 1   1 1
        puma3   0 2   1 2
        puma4   0 3   1 3
        table     2 0   3 0
        WORLD        3 1
        END
```

위의 할당결과 TiME 에 생성되는 각 타스크는 **그림 2**와 같다.

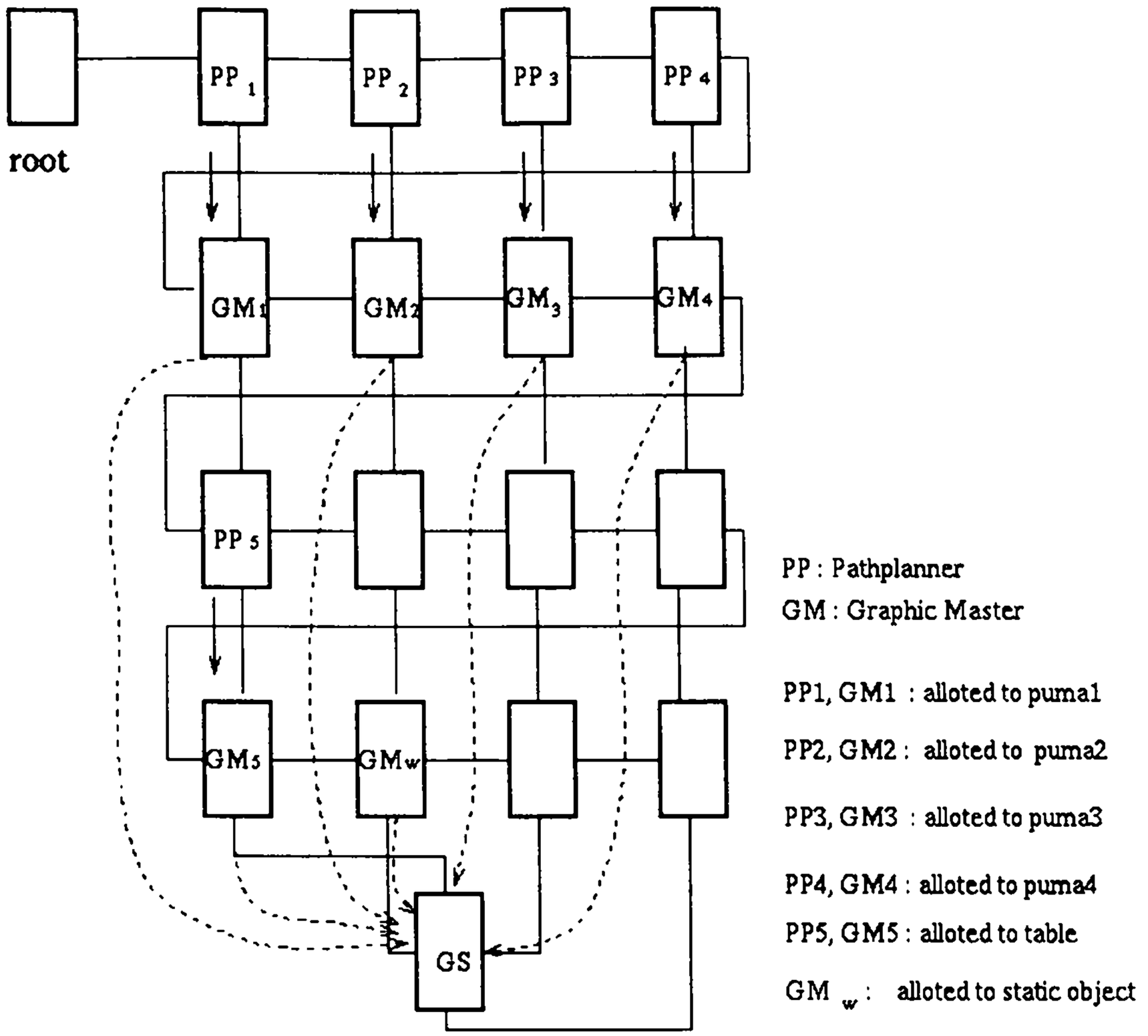


그림 2: 5 개의 로봇와 1 개의 고정물체가 있는 경우에 타스크들을 4 × 4 의 TiME 구조에 할당한 형태

4. 시스템 구성

시스템이 실제 시뮬레이션을 시작하기 전에 시뮬레이터는 두 단계의 과정(Loader,Distributer)을 거쳐 시스템을 구성하게 된다. Root node 에서 실행되는 main routine 에서는 routine 의 마지막 부분에서 초기화를 수행하는 함수를 호출 한 후, 로봇 시뮬레이션을 위해 root node 에서 실행되어야 할 타스크들을 생성하고 시뮬레이터 설정 등을 수행하게 하는 root_main() proedure 를 호출 하도록 되어있다. root_main() procedure 에서 호출되는 procedure 로는 Loader(), InitMoveHdl(), Distributer(),

InitApplications(), Banner(), InitUserIO() 등이고 각각의 기능은 다음과 같다.

가. Loader()

각 로봇에 대한 정보를 읽어서 database 에 저장한다. 로봇 시뮬레이터를 실행시키게 되면 커널에 대한 초기화가 수행된 후 robot specification file 이름을 묻는 message 가 출력된다. Robot specification file 이름을 입력하면 파일을 open 하여 정보를 읽어 나가며 syntax check 도 수행한다. 읽혀진 정보는 Robot 자료 구조를 갖는 robot_defs[ ] 라는 table 에 저장된다.

나. InitMoveHdl()

-- moving operation handling 타스크인 MoveHdl_Code()를 타스크로 생성시킨다.

다. Distributer()

-- Distributer() 에서는 두 개의 procedure, System_Configurer() 와 Robot_Distribution()를 호출 한다. System_configurer 는 system specification file 의 SYSTEM 정의 부분에서 타스크가 할당된 프로세서에 타스크를 생성시키는 message 를 보낸다. 이것은 커널의 Configurer TOPS_CREATE_RUN_TASKS message 를 전송하여 이루어진다. Robot_Distribution()에서는 Loader()에 의해 읽혀진 각 로봇에 대한 정보 중 kinematic 계산에 필요한 정보, 즉 specification file 에서 정의된 Denabit-Hartenberg parameters, 링크에 설정된 frame, joint value limit 등을 pathplanner 에 넘겨준다 (object.h 의 pp_Robot_Descr 자료 구조 참조) 또한 그래픽 master 에서 필요한 정보, 즉 specification file 에서 정의된 것중 객체의 geometry 와 각 링크에 설정된 frame 에 대한 정보 등을 그래픽 master 에 넘겨준다(object.h 의 GM_Robot_Descr 자료 구조 참조).

라. InitUserIO()

-- 시뮬레이션을 위한 configuration 이 끝나고 본격적으로 user 와 interaction 을 통하여 시뮬레이션을 수행할 수 있도록 menu 를 제공하고 user 로부터 입력을 받는 타스크를 생성한다. Configuration 이 끝나면 UserIO_Code()가 InitUserIO()에 의해 타스크

로 생성되게 되는데 이것은 user 로부터 <space bar> < return> 입력을 기다리고 있다. 일단 <space bar> < return> 입력을 받게 되면 시뮬레이터에서 제공되는 메뉴가 출력되고 user 와의 interaction 을 수행하게 된다. UserIO_Code() task 는 User 와 interface 역할을 하는 타스크 이다.

5. Path Planner

Path planner 는 로봇의 움직임과 관련된 계산을 수행하기 위한 타스크를 말한다.

로봇 시뮬레이터에서 사용자에게 제공되는 메뉴 중 move 와 관련된 menu 는 **그림 3** 과 같다. Pathplanner 에서는 interpolation 과 inverse kinematic 계산을 수행한다. Path planner 는 loader 에서 저장된 로봇에 대한 정보중 kinematic 계산에 필요로 되는 DH(Denavit-Hartenberg) 변수와 joint 의 최대, 최소값 등을 받아 저장한 후 kinematic 계산을 수행하게 된다. Move 의 submenu 각각의 기능과 구현은 다음과 같다.

0) cart move
1) cart set
2) cart_move_rr
3) cart set rr
4) joint move
5) joint set
6) set single axis
7) locate robot

그림 3:이동시의 선택사항들

가. cart move

사용자로부터 end_effector 에 설정된 frame 의 position 과 rotation 값을 입력받아, 해당 위치로 이동하는 사이에 interpolation 을 통하여 로봇이 움직이는 동작을 나타내고 있다. 이 때 사용자로부터 속도 또한 입력으로 받는데, 속도에는 어떤 물리적 의미가 내포된 것이 아니고 interpolation 간격을 얼마나 세밀하게 하는가를 결정하는 요소로 작용한다. 속도가 클 수록 interpolation 간격이 커져서 더 빠른 움직임을 얻을 수 있다. Interpolation 은 현재의 위치와 목적위치의 차를 interpolation 수로 나눈 간격을 매번 더하여 목적지에 도달하게 하는 간단한 방법을 사용하고 있다. 두 개이상의 로봇이 움직이다 충돌이 일어나는 경우에 대한 collision detection 기능은 구현되어 있지 않은 상태이다.

End_effector 의 frame(translation and rotation)이 주어질 때, 그 위치에 도달하기 위한 각 로봇내 각 링크들의 관계를 나타내는 joint value 를 구하는 inverse kinematic routine 들이 각 interpolation 에서 이용된다. 현재 로봇의 구조로 도달할 수 없는 위치가 목적지로 주어질 때에는 error 를 표시한 후 moving 이 중단된다. Inverse kinematic routine 에 입력으로 주어지는 것은 로봇의 목적지 frame 값과 PP(Pathplanner)에 전송되어 온 로봇의 kinematic 정보들이고 출력은 현재 로봇의 형태를 나타내는 각 joint 값이다. 로봇의 움직임은 각 interpolation step 마다 변형된 로봇의 형태인 joint value 들을 그래픽 master 에 넘겨 줌으로서 이루어진다. GM(그래픽 Master)에서는 변형된 로봇의 모양에 따라 각 링크에 설정된 변형된 frame 값을 구하여 그래픽 데이타를 생성한 뒤 그래픽 서버에 보내준다. 로봇에 대한 그래픽 데이타가 모두 GS(그래픽 서버)로 보내진 후 GM 은 다음 interpolation step 의 시작 signal 을 기다리고 있는 PP 에 operaion signal 을 보내서 다음 interpolation 이 시작되도록 한다. Interpolation 이 종료되면 end_effector frame 의 최종 값과 error interpolation step 수, error code 등이 출력된다.

나. cart set

cart set menu 는 목적지에 frame 값이 cratesian 좌표를 입력으로 받는다. 그러나 interpolation 을 수행하지 않고 한번에 목적지로 이동된 로봇의 형태를 그려준다. 목적지가 도달할 수 없는 위치이면 error 가 출력된다. 목적지에 도달된 후 역시 end_effector frame 값이 출력된다.

다. cart move rr, cart set rr

cart move 와 cart set 에서 사용자로부터 주어진 frame 은 world frame 에 대한 값이다. 그러나 cart move rr 과 cart set rr menu 에서 사용자로부터 받은 입력은 base object 에 설정된 frame 에 대한 end_effector frame 값으로 계산된다.

라. joint move

cart move 는 end_effector frame 값을 입력으로 받았으나 joint move 에서는 destination 에서의 각 joint 의 값을 사용자로부터 입력받는다. Joint value 를 통한 interpolation 을 수행하여 움직이는 로봇을 나타낸다. Interpolation 을 통하여 각 interoplation step 에서 joint 값이 계산되므로 inverse kinematic 함수를 이용할 필요가 없다.

마. joint set

입력받은 joint 값들로 한번에 변형한다. 새로운 joint 값들을 GM 에 넘겨주면 GM 에서는 변화된 joint 값에 따라 그래픽 데이타를 생성한다. 사용자에 의해 입력된 joint 값들이 로봇 구조에 맞지 않는 joint value 들의 집합이라면 error code 가 출력된다.

바. set single axis

joint 중 하나의 joint 값만을 변환하여 로봇을 움직인다.

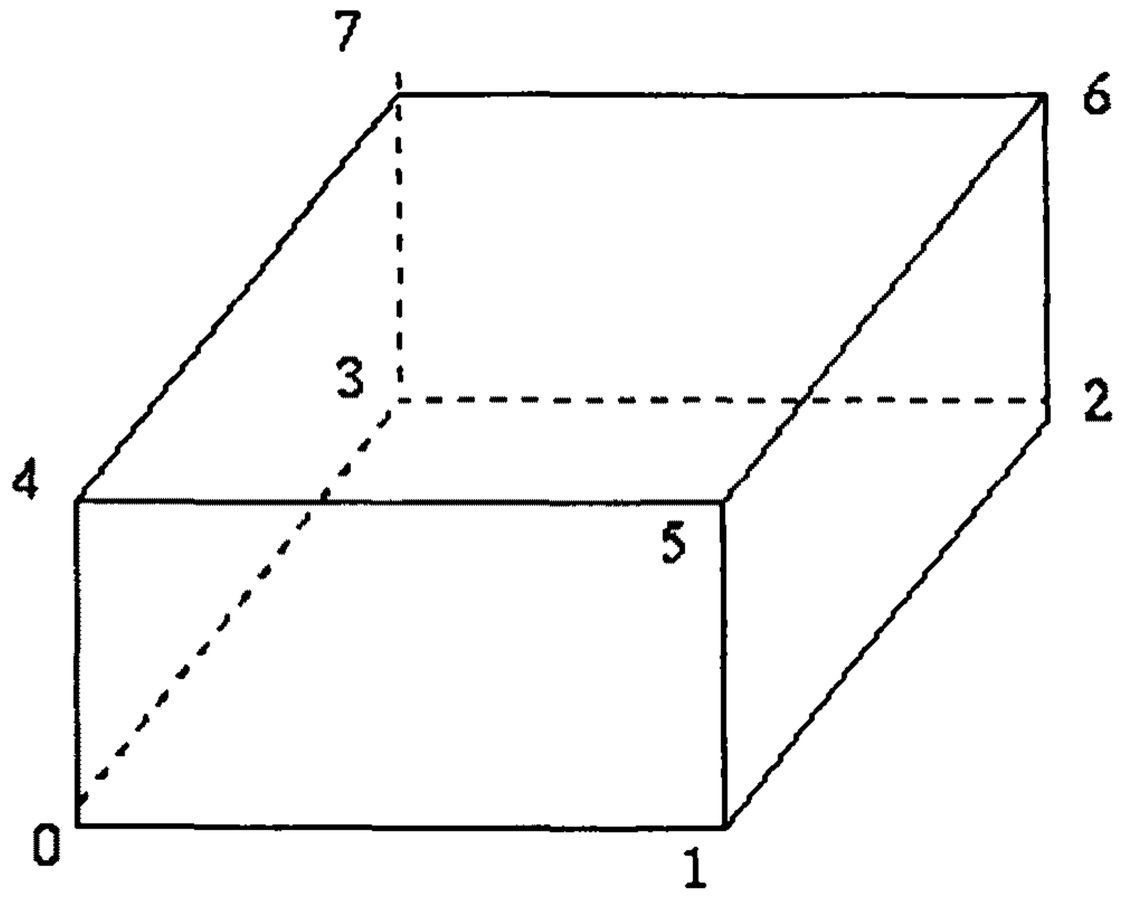
사. locate robot

로봇의 위치를 변환시킨다. Locate robot menu 를 선택하면 submenu 가 나타난다. Submenu 에는 0(global), 1(local)의 두 가지가 포함된다. global 은 로봇이 움직여질 위

치에 대한 좌표로써 world frame에 대한 좌표로 입력을 주게 되고, local의 경우 base frame에 대한 좌표로 입력하게 된다.

6. GM(Graphic Master)

로봇의 링크로 이용되는 객체는 system specification에서 정의한다. 객체의 모양은 육면체와 원통형 두 가지 모양으로 정의되고 각각의 형태는 길이, 폭, 높이, 지름, 높이, 옆면을 그리는 선의 수 등으로 나타내어진다. 또한 각 객체에 설정된 frame도 설정된다. 이러한 값들은 로봇의 drawing에 필요한 데이타 들로써 Loader()에 의해 읽혀진 후, Distributer()에 의해 GM으로 전달되고 각 GM은 자신이 할당된 로봇의 drawing에 필요한 각종 데이타를 갖게된다. 위와 같은 객체에 대한 데이타로부터 그래픽 routine들에 의해 바로 이용될 수 있는 데이타의 생성은 GM에서 이루어 진다. Box에 대한 정의로부터 box를 그릴 수 있는 데이타의 생성은 create_box_graphics()에 의해 이루어 진다. 또한 cylinder에 대한 정의로부터 cylinder를 그릴 수 있는 데이타의 생성은 create_polyeder_graphics()에 의해 이루어 진다.



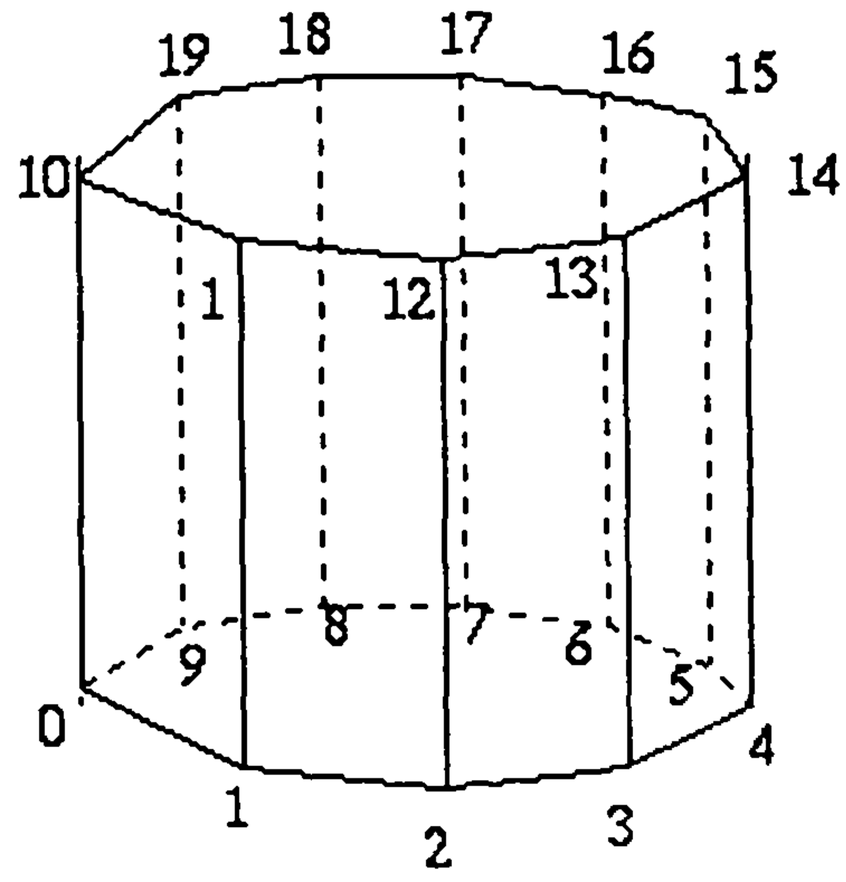


그림 4: generate_graphics()함수에 의해 생성되는 그래픽 물체들

로봇의 링크는 box 와 polyeder 등의 기본 그래픽 개체들로 이루어져 있다. 이러한 기본요소로부터 그래픽 데이타를 생성하는 routine 은 generate_graphics()이다. generate_graphics()에서 box primitive 는 create_box_graphics()에 의해 생성되고

cylinder 는 create_polyeder_graphics()에 의해 생성된다. **그림 4**는 위의 두 데이타 생성 routine 에 의해 생성된 box 와, polyeder 를 나타내고 있다. 생성된 데이타의 형태는 다음과 같다.

가. Box object 로부터 생성된 그래픽 데이타

육면체를 그리기 위해 8 개의 꼭지점을 구하고 각각 꼭지점을 선으로 육면체를 그릴 수 있다. create_box_graphics()로부터 생성되는 그래픽 데이타는 8 개의 꼭지점이 저장된 벡터 array 와 각 array 를 index 하여 선분으로 잊게하는 integer array 로 구성된다. integer array 가 갖는 값은 (0, 1, 2, 3, 0, 4, 5, 6, 7, 4, -1, 7, 3, -1, 5, 1, -1, 6, 2, -1, -2)이다. -1 이 array element 의 값인 경우에는 선분으로 연결되지 않는 것을 의미한다. 즉 4, -1, 7 의 경우 4 번째 벡터와 7 번째 벡터를 잇는 직선을 그리지 않는다는 것을 의미한다. -2 는 index array 의 마지막 element 임을 나타내는 값이다.

나. Polyeder object 로부터 생성된 그래픽 데이타

create_polyeder_graphics()로부터 생성된 그래픽 데이타도 box object 데이타와 같은 구조를 가지나 index array 의 값에는 차이가 발생한다.

PP 로부터 로봇의 형태와 관련하여 GM 에 넘겨지는 데이타는 각 joint 의 값들이다. 이 joint 값들로부터 실질적인 변화된 로봇에 대한 그래픽 데이타의 생성은 update_robot() routine 에 의해 이루어진다. update_robot()에서는 PP 로부터 받은 joint 값들중 현재 joint 의 값과 다른 값을 갖는 joint 에 대해 build_DH() routine 을 call 하여 joint 에 할당된 frame 을 구하고 새로운 frame 에 따라 그래픽 데이타의 좌표들을 다시 계산한다. 로봇 내의 모든 링크에 대한 그래픽 데이타의 생성이 완료된 후 그래픽 서버에 데이타를 보낸다.

7. GS(그래픽 서버)

GS 는 시뮬레이션 윈도우의 생성, GM 으로부터 전달되어 온 그래픽 데이타를 이용하여 시뮬레이션 윈도우 상에 직접 그림을 그리는 일 등을 담당한다. 시뮬레이션 윈도우로 생성된 윈도우는 윈도우에 drawing 을 수행하여도 refresh 되기전에는 화면상에 나타나지 않게한다. 모든 로봇 들이 한 번씩 그려진 후 윈도우를 refresh 하여 화면에 출력하게 된다. 이러한 방법으로 애니메이션이 구현되었다. 원래 로봇 시뮬레이터의 그래픽 서버에서는 low level 그래픽 routine 을 작성하여 그래픽 서버를 구현하였으나 TiME 에서는 ttg3 를 위한 그래픽 library 인 ttgs.lib 가 제공되므로 ttgs library 에서 제공되는 library 를 이용하여 그래픽 서버를 수정하였다. 수정된 부분에는 in-line comment 로 표시해 두었다.

3 절 Robotics 의 소개

Manipulator(i.e. Robot) kinematics 는 manipulator 팔이 움직이는 기하학적인 형태를 연구하는 분야이다. 즉, 로봇에 의해 수행되는 모든 일은 manipulator 팔의 각 링크의 움직임에 의하여 이루어지므로 kinematics 는 manipulator 의 설계와 제어에 필수적인 도구이다. 이 절에서는 로봇 팔의 각 링크의 움직임을 표현하는데 필수적인 수학적 도구들을 살펴본다. 이 것은 로봇 시뮬레이션의 구현원리를 이해하는데 필수적이다.

1. Mathematical Preliminary

로봇 팔의 각 링크(link)들은 rigid body 의 체계로 모형화될 수 있다. 각 rigid body 의 위치는 변위와 방향에 의해 표시될 수 있다. **그림 5** 에서 O-xyz 를 지면에 대한 좌표축이라하고 O'는 rigid body 내의 한 점이라 하면, rigid body 의 변위는 O-xyz 좌표축에 대한 O'의 좌표로 나타낼 수 있다.

$$X_0 = \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} \tag{1}$$

$\mathbf{X}_0$는 3×1 column 벡터이다. Rigid body 의 방향을 나타내기 위해 **그림 5** 에 나타난 것처럼 세 좌표축, x_b, y_b, z_b 이 rigid body 내에 설정된다.

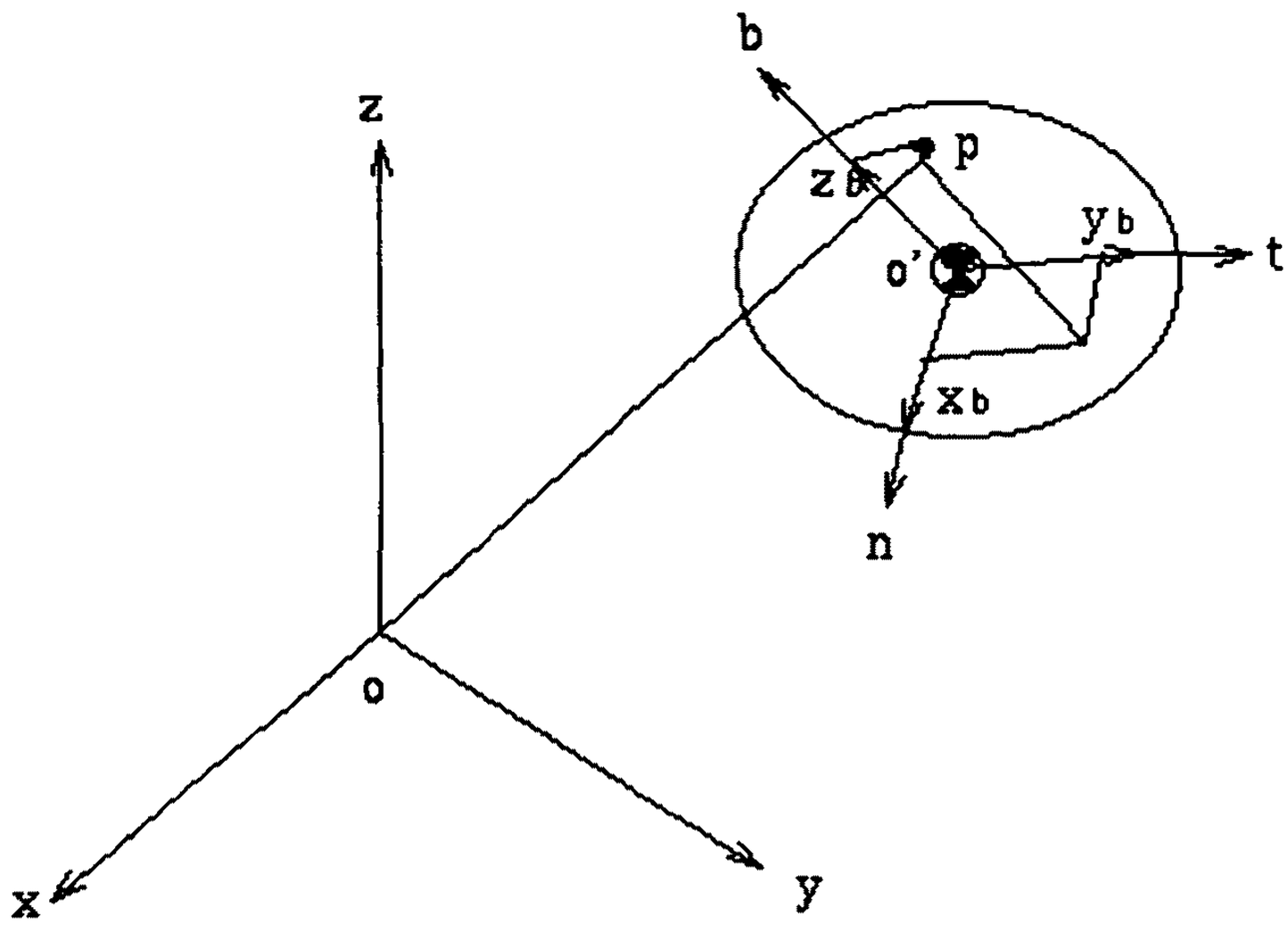


그림 5: 좌표 변환

이 좌표축들이 rigid body 와 함께 움직이는 또 다른 좌표계, $O - x_b y_b z_b$ 를 형성한다. Rigid body 의 방향은 이 좌표축들의 방향에 의해 나타내 진다. **n, t** 와 **b** 가 좌표축 x_b, y_b, z_b 의 방향을 나타내는 단위 벡터들이라면, 이것을 모아 3×3 행렬 **R** 로 나타내고 이것은 고정좌표계 O-xyz 에 대한 rigid body 의 방향을 나타낸다.

$$R = [n,t,b] \tag{2}$$

P 를 공간상의 임의의 점이라고 하면 P 고정좌표계 O-xyz 에 대한 좌표로 다음과 같이 나타낸다.

$$X = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{3}$$

P 는 rigid body 에 고정된 좌표계인 $O' - x_b y_b z_b$ 에 대해 다음과 같이 표현될 수 있다.

$$X^b = \begin{pmatrix} u \\ v \\ w \end{pmatrix} \tag{4}$$

superscript ^{b} 는 그 벡터가 body 좌표계에 대한 좌표임을 나타낸다.

두 좌표계 사이의 관계를 살펴보자. 이 관계는 고정좌표계와 body 좌표계 사이의 좌표변환을 정의한다. 위에서 살펴본 rigid body 의 변위를 나타내는 3×1 벡터 x_O 와 방향을 나타내는 3×3 행렬 **R** 이 좌표변환을 위해 쓰여진다. **그림 5** 에서 볼 수 있는 것처럼 점 p 는 O', A 그리고 B 를 통하여 접근할 수 있다. 이것은 수학적으로

$$\overrightarrow{OP} = \overrightarrow{OO'} + \overrightarrow{O'A} + \overrightarrow{AB} + \overrightarrow{BP}$$

$$OP = \mathbf{x}, OO' = \mathbf{x}_0 \tag{5}$$

벡터 $\overrightarrow{O'A}, \overrightarrow{AB}, \overrightarrow{BP}$ 는 각각 단위 벡터 **n, t, b** 와 평행하다. 그리고 각각의 길이는 u, v, w 이다. 그러므로 (5) 를 다음과 같이 다시 쓸 수 있다.

$$\mathbf{x} = \mathbf{x}_0 + u\mathbf{n} + v\mathbf{t} + w\mathbf{b} \tag{6}$$

(1) 과 (4) 로부터

$$\mathbf{x} = \mathbf{x}_0 + \mathbf{R}\mathbf{x}^b \tag{7}$$

(7)은 body 좌표 x^b 에서 고정좌표 x로의 변환을 나타낸다. 이 좌표 변환은 rigid body의 변위 x_0와 방향 행렬 **R**에 의해 주어진다. 즉, 고정좌표계에 대한 body 좌표계로 주어진다.

(7)을 **R**의 transpose 행렬 $\mathbf{R}^T$로 곱하면

$$\mathbf{R}^T\mathbf{x} = \mathbf{R}^T\mathbf{x}_0 + \mathbf{R}^T\mathbf{R}\mathbf{x}^b \tag{8}$$

행렬 **R**은 orthonormal 행렬이므로 $\mathbf{R}^T\mathbf{R}$은 단위 벡터가 된다. 그러므로(8)은

$$\mathbf{x}^b = -\mathbf{R}^T\mathbf{x}_0 + \mathbf{R}^T\mathbf{x} \tag{9}$$

로 나타낼 수 있다. (9)는 고정좌표에서 body 좌표로의 변환을 나타낸다. 즉(7)의 역변환이다. 역변환은 단순히 행렬 **R**의 transpose를 이용하여 이루어 진다.

가. 오일러 각도

앞 절에서 rigid body의 방향을 나타내기 위하여 3 × 3 행렬 **R**을 이용하였다. 그러나 행렬의 각 원소들은 독립 변수가 아니다. 행렬의 원소 수는 9개 이지만 서로 orthogonality 조건과 단위 길이 조건을 만족하여야 한다(condition of orthonormal matrix). 전체적으로 6개의 조건이 있으므로 9개의 원소 중 3개 만이 독립 변수이다. 이 절에서는 rigid body의 방향을 3개의 독립 변수로 나타내는 표현법에 대해 알아보기로 한다.

그림 6 에서 좌표계 O-xyz의 회전을 보면, 처음 좌표계는 z축을 중심으로 ϕ만큼 회전한다(**그림 6** (a)). 다음 새로운 좌표계 O-$x'y'z'$가 x' 축을 중심으로 θ만큼 회전한다(**그림 6** (a)). 마지막으로 O-$x'y''z''$가 z'' 축을 중심으로 φ만큼 회전한다. 세 번의 회전을 통해 생성된 좌표계 $O-x_by_bz_b$를 **그림 6** 에서 볼 수 있다. 이 세 각도, ϕ, θ, φ는 좌표계의 방향을 정할 수 있고, 오일러 각도라고 한다. 오일러 각도는 각각 임의

대로 변할 수 있으므로 독립변수이다.

위의 오일러 각도가 나타내는 세번의 연속적인 회전을 수행하는 회전 메트릭스를 계산하여 보자. 첫 번째 ϕ만큼 회전하는 변환 메트릭스는 $\mathbf{x}' = [x', y', z']^T$를 $\mathbf{x} = [x, y, z]^T$로 변환하는 3×3 회전 메트릭스 $\mathbf{R}_z(\phi)$이다. 즉,

$$\mathbf{x} = \mathbf{R}_z(\phi)\mathbf{x}'\mathbf{R}_z(\phi) = \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{10}$$

같은 방법으로 $\mathbf{x}'' = [x', y'', z'']^T$를 $\mathbf{x}'$를 중심으로 θ만큼 회전변환하는 메트릭스는 다음과 같이 주어 진다.

$$\mathbf{x}' = \mathbf{R}_{x'}(\theta)\mathbf{x}''\mathbf{R}_{x'}(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \tag{11}$$

마지막으로 φ만큼 회전하는 경우는,

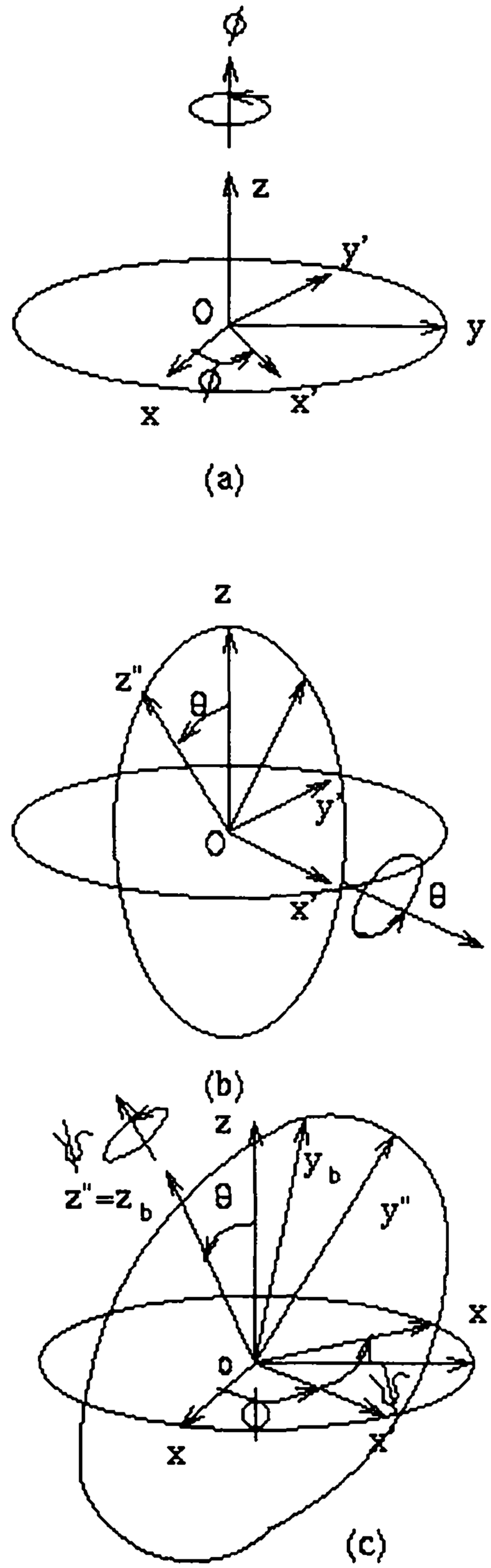


그림 6: 오일러 각도를 정의하기위해 사용된 3 연속 회전

$$\mathbf{x}'' = \mathbf{R}_{z''}(\varphi)\mathbf{x}^{b}\mathbf{R}_{z''}(\varphi) = \begin{bmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{12}$$

이다. 세 번의 변환을 묶으면

$$\mathbf{x} = \mathbf{R}_{z}(\phi)\mathbf{R}_{x'}(\theta)\mathbf{R}_{z''}(\varphi)\mathbf{R}(\phi,\theta,\varphi) = \mathbf{R}_{z}(\phi)\mathbf{R}_{x'}(\theta)\mathbf{R}_{z''}(\varphi) \tag{13}$$

메트릭스 $\mathbf{R}(\phi,\theta,\varphi)$는 좌표계 O-xyz 에서 $O-x_b y_b z_b$ 로의 변환을 의미한다.

2. Kinematic Modeling of Manipulator Arms

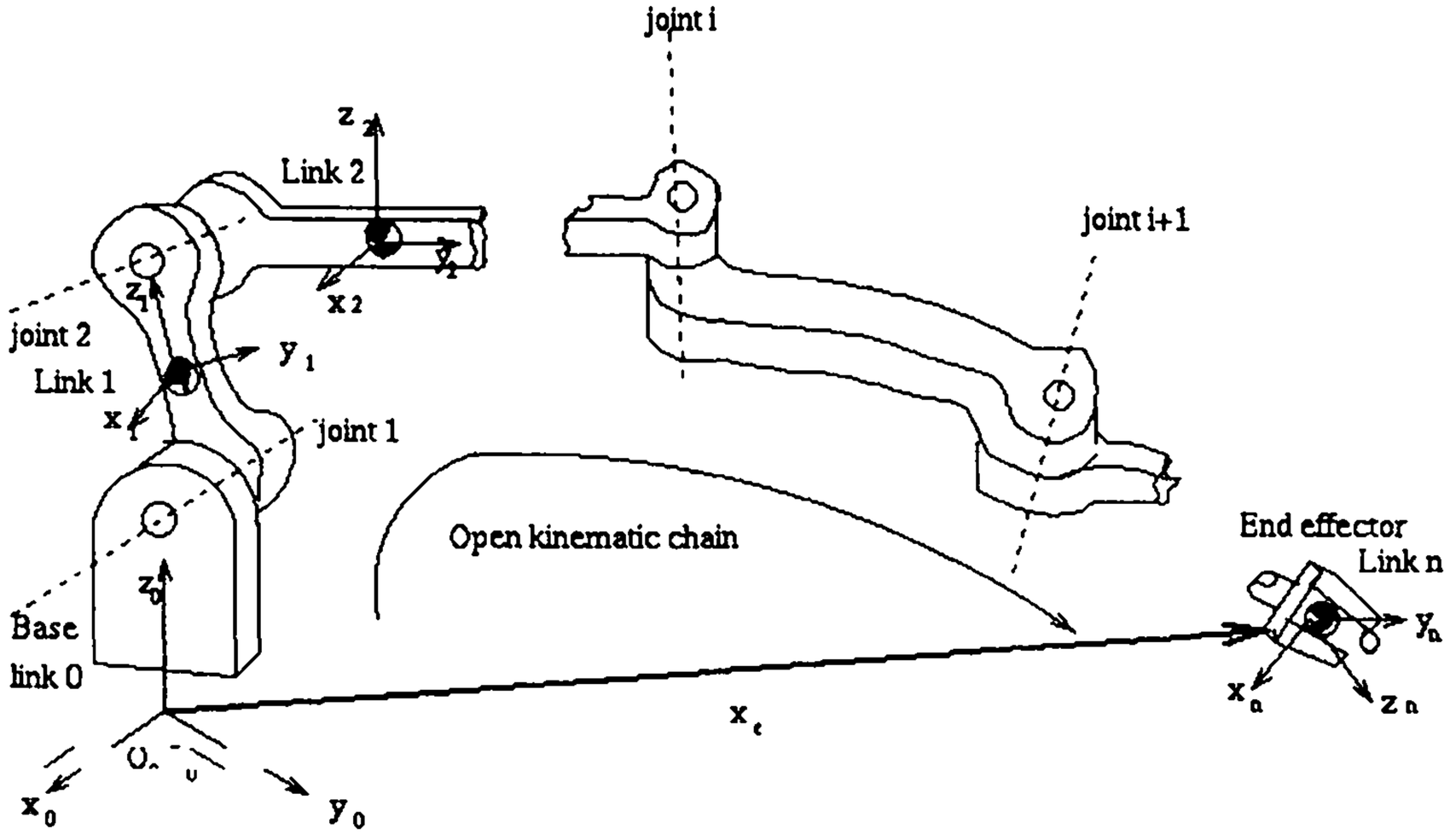


그림 7: open kinematic chain

가. Open Kinematic Chains

앞에서 다루었던 수학적 도구들을 이용하여 이 절에서는 로봇 팔의 kinematic

modeling 을 다루기로 한다. 특히 로봇 팔의 각 링크의 변위와 방향을 나타내기 위해 homogeneous transformation 메트릭스를 사용한다.

로봇 팔은 기본적으로 rigid body 의 연속체라고 할 수 있다. **그림 7** 은 rigid body 들의 연결로 이루어진 로봇 팔을 형상화 한 것이다. 이와 같은 open loop 구조를 open kinematic chain 이라 한다. 대부분의 산업용 로봇의 구조가 open kinematic chain 혹은 그와 비슷한 형태이다. **그림 7** 에서 볼 수 있드시 open kinematic chain 의 각 링크들은 순차적으로 0 부터 n 까지의 번호가 부여된다. 지면에 고착된 링크는 편의상 0 번으로 가장 먼 곳에 위치한 링크는 n 번으로 지정된다. 로봇 팔이 어떤 작업을 수행할 때 마지막 링크에 붙어 있는 end_effector 의 동작에 의해 작업이 이루어지므로 우리의 주된 관심사는 마지막 링크의 움직임을 분석하는 것이다. End_effector 의 변위와 방향을 나타내기 위해, 마지막 링크에 좌표계 $O-x_n y_n z_n$ 를 설정한다. 좌표계의 위치는 다른 좌표계인 $O_0-x_0 y_0 z_0$ (전역 좌표)에 대한 값으로 나타낸다. End_effector 의 동작은 base link 와 end_effector 사이에 있는 중간 링크들의 동작에 의해 발생한다. 그러므로 end_effector 의 위치는 base link 로부터 마지막 링크까지의 변위와 방향을 추적하여 알 수 있다. 이런 이유로 각각의 링크에 좌표계를 설정한다. 말하자면 i 번째 링크에 좌표계 $O_i-x_i y_i z_i$ 를 설정한다. 좌표계 $O_i-x_i y_i z_i$ 의 변위와 방향은 바로 이전 좌표계인 $O_{i-1}-x_{i-1} y_{i-1} z_{i-1}$ 에 대한 값으로서 4 × 4 메트릭스로 나타낸다. 이것은 두 좌표계 사이의 homogeneous transformation 을 나타낸다. 그러므로 end_effector 의, 변위와 방향은 마지막 좌표계로부터 base 좌표계까지의 연속적인 homogeneous transformation 에 의해 이루어 진다.

인접한 두 링크의 상대적인 움직임은 두 링크들을 연결하는 joint 의 움직임에 의해 유발된다. **그림 7** 에서 볼 수 있듯시 n+1 개의 링크로 이루어진 로봇 팔은 n 개의 joint 를 갖는다. i-1 번째 링크와 i 번째 링크 사이의 joint 를 joint i 라 부르기로 한다. End_effector 의 변위와 방향은 n 개의 joint 들의 변위에 의해 결정된다.

나. The Denavit-Hartenberg Notation

이 절에서는 open kinematic chain 에 포함된 두 인접 링크들의 운동역학적인 관계에 대해 알아본다. Denavit-Hartenberg 표현법은 이 운동역학적 관계를 묘사하는 체계적인 방법으로 이용되고 있다. 이 방법은 rigid body 의 변위와 방향을 4 × 4 메트릭스에 의해 나타내는데 기초하며, 운동역학적 관계를 최소한의 변수들을 이용하여 완전히 표현할 수 있다.

그림 8은 인접한 한 쌍의 링크(link i-1 과 i)들과 각각의 링크에 관련된 joint(joint i-1, i, i+1)을 보여준다. 직선 H_iO_i 는 joint 축 i 와 i+1 에 대한 공통 수선이다. 두 링크들 사이의 관계는 각 링크에 설정된 좌표계의 상대적 변위와 방향에 의해 설명된다. Denavit-Hartenberg 표현법에서 i 번째 좌표계의 원점, O_i 는 joint 축 i+1 과 joint 축 i 와 i+1 의 공통수선이 교차하는 지점에 위치한다. i 번째 링크의 좌표계는 i 번째가 아닌 i+1 번째 joint 축에 위치하게 된다. x_i 축은 공통수선의 연장선 상에 있고, z_i 축은 i+1 joint 축위에 있다. 마지막으로 y_i 축은 좌표계 $O-x_by_bz_b$ 가 right-hand coordinate 체계가 되도록 설정된다.

두 좌표계의 상대적 위치는 다음의 네 변수들에 의해 완전하게 결정된다.

- a_i 공통수선의 길이
- d_i 원점 O_{i-1} 과 점 H_i 사이의 길이
- α_i joint 축과 z_i 축 사이의 오른손 방향 각도
- θ_i z_{i-1} 축을 중심으로 한 x_{i-1} 축과 공통수선 H_iO_i 사이의 오른손 방향 각도.

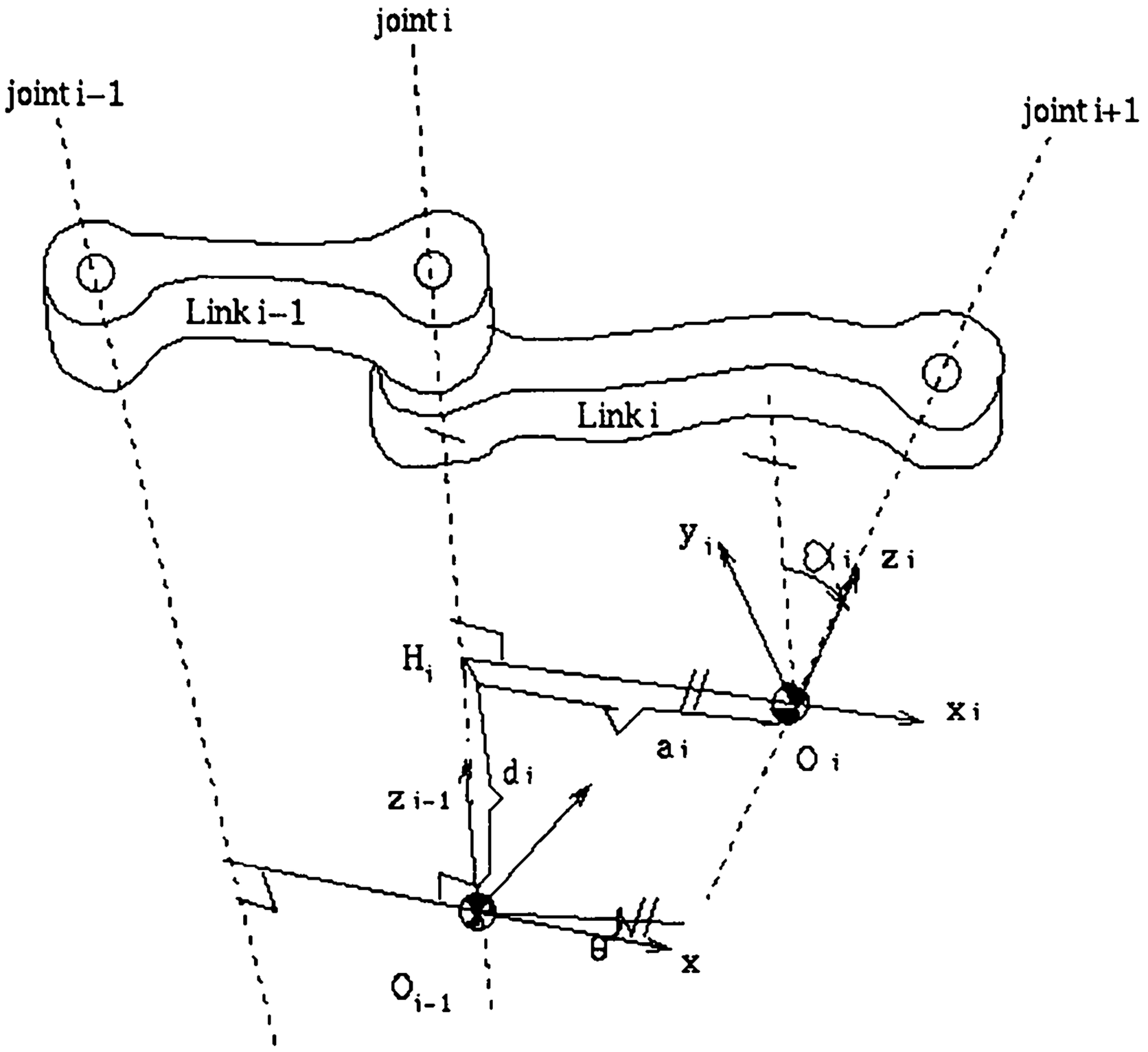


그림 8: Denavit-Jartenberg notation

변수 a_i와 α_i는 링크의 기하학적인 형태에 의해 결정되어 지는 constant 들 이다. a_i는 링크의 길이를 나타내고 α_i는 두 joint 좌표축들 간의 뒤틀림의 정도를 나타낸다. Joint 가 움직일 때 다른 두 개의 변수들인 d_i와 θ_i 중 하나가 변한다.

로봇 팔에 쓰이는 joint 장치로는 joint 축을 중심으로 인접한 링크들이 상대방에 대하여 회전하는 revolute joint 와 joint 축을 따라 인접한 링크들이 선형 이동을 하는 prismatic joint 등 두 가지가 있다. Revolute joint 의 경우 d_i는 일정하고 α_i 는 joint 의 변위를 나타내며 변하는 값이다. Prismatic joint 의 경우 α_i는 일정하고 d_i 는

joint 의 변위를 나타내며 변하는 값이다.

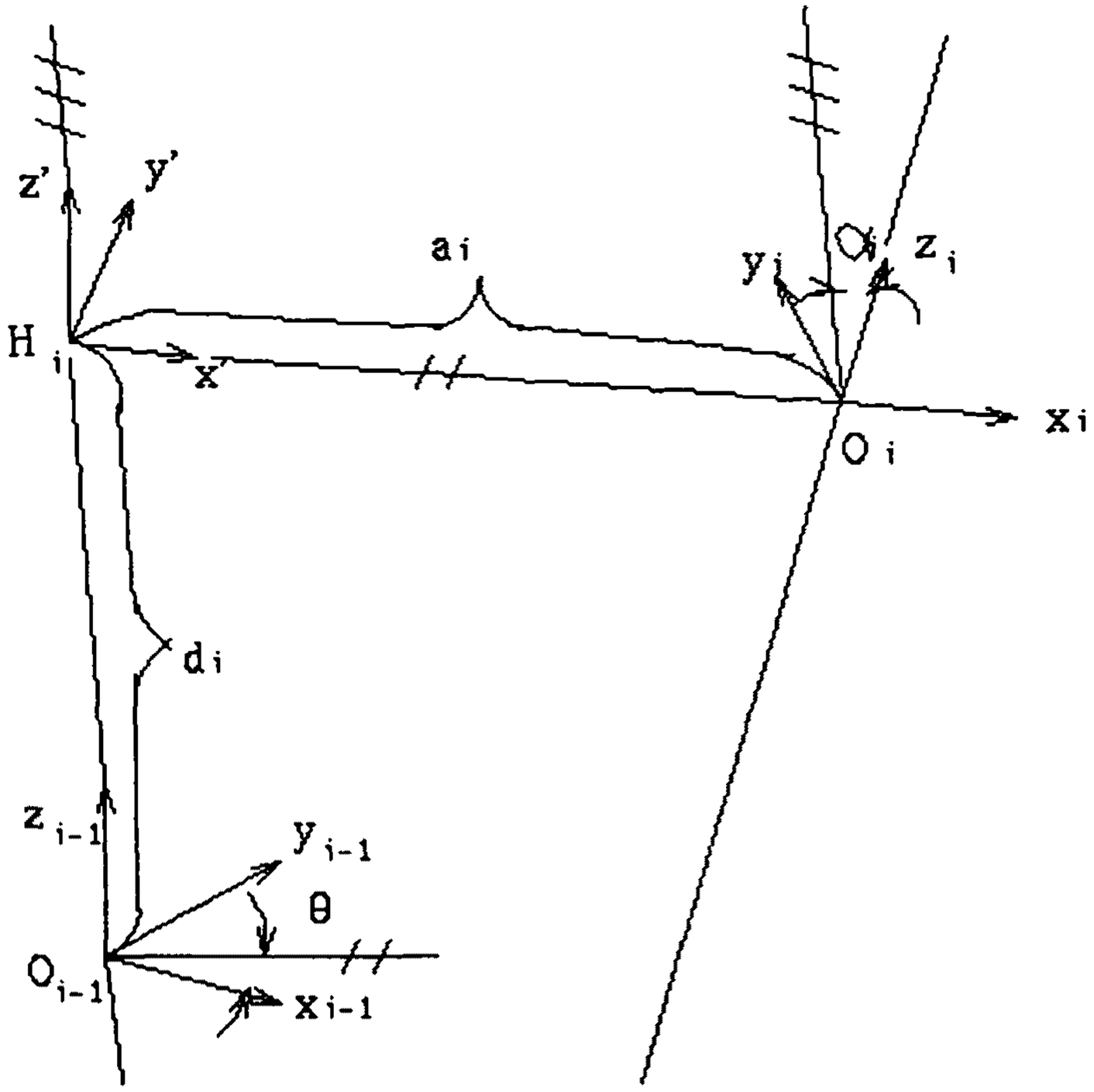


그림 9: Denavit-Hartenberg 표기상에서 인접한 좌표축들간의 관계

인접한 두 링크들 간의 운동역학적인 관계를 4×4 메트릭스를 이용하여 공식화 해 보자. 0에서 설명된 수학적 성질에 의해 $i-1$ 좌표계에 대한 i 좌표계의 상대적인 위치를 나타내는 4×4 메트릭스는 i 좌표계로부터 $i-1$ 좌표계의 좌표로 변환시키는 변환메트릭스를 구함으로써 얻어질 수 있다.

그림 9 는 두 좌표계 $O-x_iy_iz_i$ 와 $O-x_{i-1}y_{i-1}z_{i-1}$ 과 점 H_i 에 설정된 중간 좌표계 $H-x_i'y_i'z_i'$ 를 보여주고 있다. $\mathbf{X}_i$, $\mathbf{X}'$, $\mathbf{X}_{i-1}$ 를 각각 $O-x_iy_iz_i$, $O-x_{i-1}y_{i-1}z_{i-1}$, $\mathbf{X}_i$, $\mathbf{X}'$, $\mathbf{X}_{i-1}'$ 좌표계 안에 있는 점들이라고 하면, **그림 9** 에서처럼 점 $\mathbf{X}_i$ 로부터

X 로의 변환은 다음과 같다.

$$\mathbf{X}' = \mathbf{A}_i^{int}\mathbf{X}_i\mathbf{A}_i^{int} = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{14}$$

마찬가지로 $\mathbf{X}'$ 로부터 $\mathbf{X}_{i-1}$ 로의 변환은 다음과 같이 주어진다.

$$\mathbf{X}_{i-1} = \mathbf{A}_{int}^{i-1}\mathbf{X}'\mathbf{A}_{int}^{i-1} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{15}$$

(14)와 (15)로부터

$$\mathbf{X}^{i-1} = \mathbf{A}_i^{i-1}\mathbf{X}_i\mathbf{A}_i^{i-1} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{16}$$

메트릭스 $\mathbf{A}_i^{i-1}$ 는 i-1 좌표계에 대한 i 좌표계의 변위와 방향을 나타낸다. $\mathbf{A}_i^{i-1}$ 메트릭스의 처음 세 3×1 행벡터들은 i 좌표계의 좌표축들의 방향코사인을 포함하고 마지막 3×1 행벡터는 원점 O_i 의 위치를 나타낸다.

다. Kinematic Equations

Denavit-Hartenberg 표현법을 이용하여 우리는 end-effector 의 위치를 joint 의 변위의 함수로써 표현할 수 있다. 각 joint 의 변위는 joint 의 형태에 따라 각도인 θ_i 혹은 거리 d_i 이다. 일반적으로 joint 의 변위를 q_i 로 표현하고, 이것은 다음과 같이 정의된다.

$$q_i = \theta_i \qquad \text{for a revolute joint}$$

$$q_i = d_i \qquad \text{for a prismatic joint}$$

i 번째 링크의 i-1 번째 링크에 대한 변위와 방향은 \$4\times4\$ 행렬, $\mathbf{A}_i^{i-1}$를 이용한 q_i의 함수로 설명된다.

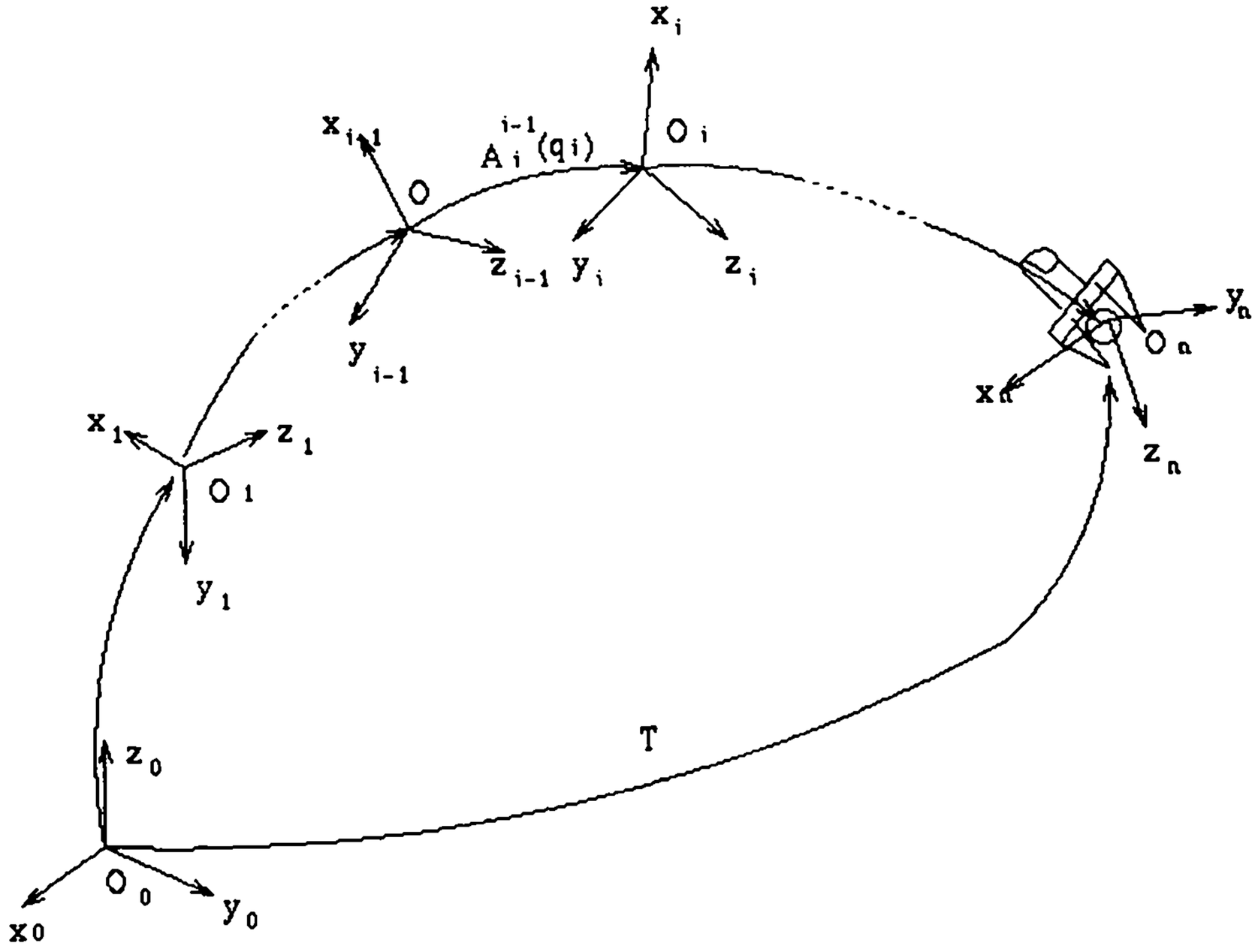


그림 10: 4 × 4 행렬을 사용한 end-effector 의 표현

이 절의 주된 모적은 마지막 링크의 방향과 변위를 base 좌표계에 대하여, q_1에서 q_n까지의 joint 변위들의 함수로 나타내는 것이다. **그림 10** 에서 볼 수 있드시, 로봇의 팔은 base 에서 끝까지 n+1 개의 링크들로 이루어져 있고, 인접한 링크들의 상대적인 위치는 4 × 4 메트릭스들로 나타난다. 연속적으로 연결된 링크들을 따라 좌표변환을 수행하면 우리는 base 좌표계에서의 end-effector 의 위치를 유도할 수 있다. 다시 말하면, 마지막 링크의 base 좌표계에 대한 상대적 변위와 방향은 다음과 같이 주어진다.

$$\mathbf{T} = \mathbf{A}_1^0(q_1)\mathbf{A}_2^1(q_2)\Lambda\ \mathbf{A}_n^{n-1}(q_n) \tag{17}$$

T는 **그림 10** 에서 볼 수 있드시, 마지막 링크의 방향과 변위를 나타내는 4 × 4 메트릭스 이다. (17)은 open kinematic chain 에 있는 각 모든 joint 들의 변위와 마지막 링크의 위치 사이의 함수적 관계를 제공해 준다. 이것은 로봇 팔의 운동역학적 등식이라고 하며, 로봇 팔의 기본적인 역학적 행태를 지배한다.

3. Inverse Kinematics

앞 절에서 유도된 운동역학 방정식은 joint 변위들과 그것들에 의한 end-effector 의 위치 사이의 함수적 관계를 제공한다. 식의 우변에 joint 변위들을 값으로 치환하면 그 값들에 해당되는 end-effector 의 위치를 찾을 수 있다.

End-effector 를 정해진 위치로 옮기기 위해서 각 joint 의 변위를 구할 필요가 있다. 이것은 앞 절에서 다룬 것과 반대되는 내용이므로, inverse kinematics problem 이라 한다. End-effector 위치가 주어진후 각 joint 변위들에 대해 운동역학 방정식을 풀어야 한다. 일단 방정식이 풀리면 원하는 end-effector 의 움직임은 각 joint 를 결정된 값으로 옮김으로써 이루어질 수 있다.

로봇 팔은 공간상의 임의의 지점에 임의의 각도로 end-effector 를 고정시키기 위해서 적어도 6 degree of freedom 을 갖아야 한다. 로봇 팔이 6 degree of freedom 보다 더 많은 degree of freedom 을 갖는다면 운동역학 방정식에 무한히 많은 해집합을 갖게 될 것이다. 6 degree of freedom 을 갖으며 산업현장에서 많이 이용되는 로봇의 구조는 5_R_1_P(5 revolute, 1 prismatic links 로 이루어진 로봇) 구조이다. 로봇 시뮬레이터의 구현에서는 5_R_1_P 와 fast_printer_device 등 두 가지 형태의 로봇에 대한 invese kinematic procedure 가 구현되었다. 사람의 팔을 생각해 보면 손가락에 있는 joint 들을 빼더라도 7 degree of freedom 을 갖는다. 그러므로 손을 table 위에 고정하더

라도, 손의 위치를 바꾸지 않고 팔꿈치의 위치를 연속적으로 바꿀 수 있다.

참고 문헌

[1] Hubertus Franke, "Multi Robot Simulation in Distributed Systems," IEEE , 1009

[2] "ANSI C toolset user manual," INMOS, 1990.

[3] "TTGS : Transtech Graphics library Release 3.1," Transtech Parallel Systems, jun 1992

[4] Asada and Slotine, *Robot Analysis and Control,* 1985.

4 절 트랜스퓨터상에서 실시간 커널을 위한 다중우선순위 스케쥴러

1. 서론

시뮬레이터가 가상 모델의 동작과정을 실시간으로 사용자에게 보여준다면 사용자는 가상 모델의 행동에 대한 이해도를 높이게 된다. 따라서 그래픽한 동작과정을 보여주는 시뮬레이터는 실시간으로 설계하는 것이 보다 적합하다. 실시간 시스템의 기본조건들로는 빠른 응답시간, 예측 가능성, 안정성 등이 있으며 특히 예측가능성의 지원을 위해 여러가지 스케쥴링 알고리즘들이 연구, 제안되고 있다. 하지만 이들 알고리즘들은 다중 우선순위를 바탕으로 제안된 것이므로 이들 알고리즘을 실제로 적용하기 위해서는 시스템이 다중 우선순위를 지원해야 한다.

반면 본 연구에서 사용하는 병렬 컴퓨터 TiME 내의 프로세서인 트랜스퓨터는 하드웨어 자체에서 스케쥴링을 지원해서 스케쥴링 커널을 사용자가 직접 구현하는 불편함을 덜어주지만 2 개의 우선순위만 지원한다. 따라서 다중 우선순위를 지원하는 소프트웨어 스케쥴러를 사용해서 TiME 을 실시간 시스템에 적합하도록 한다.

2. 시스템 환경

이미지 처리 전용으로 개발된 병렬 컴퓨터 TiME 은 32 개의 트랜스퓨터로 구성되어 있다. 이들 트랜스퓨터는 기본적으로 4 × 4 의 메쉬(mesh)구조를 이루고 있어서 프로세서의 확장이 용이하다. 호스트로는 IBM/PC 호환기종을 사용하며 DOS 를 기본 운영체제로 한다. 32 개의 트랜스퓨터외에도 이미지 입력장치와 출력장치 디스크 제어용 트랜스퓨터를 포함하고 있다. 각 트랜스퓨터에는 벡터 프로세서가 내장되어 있으며 peak performance 는 1.2Gflops 에 달한다.

병렬 시스템의 CPU 로 사용되는 트랜스퓨터는 자체에 하드웨어 스케쥴러가 내장되어 있다. 하드웨어 스케쥴러는 2 단계의 우선순위를 지원하며 높은 우선순위 큐

와 낮은 우선순위 큐를 관리한다. 이들 큐는 메모리상에서 list의 형태로 유지되며 이 list의 front와 back을 가리키는 front-pointer(FPtrReg0, FPtrReg1)와 back-pointer(BPtrReg0, BPtrReg1)들을 하드웨어 스케쥴러가 사용한다. 이들 한 쌍의 pointer는 우선순위 큐마다 존재한다. 하드웨어 스케쥴러는 각 우선순위마다 우선순위 큐 이외에도 시간대기 큐, 통신대기 큐를 관리한다. 하드웨어 스케쥴러는 우선순위 큐마다 스케쥴 정책이 다른데 높은 우선순위 큐에 대해서는 CPU를 점유한 프로세스의 실행이 끝나거나 시간대기, 통신대기에 의해서 대기상태에 들어가기 전까지는 인터럽트되지 않는다. 반면 낮은 우선순위 큐에서는 time slice가 존재한다. 이 time slice는 대개 2048μsec인데, 하드웨어 스케쥴러는 time slice만큼 프로세스에게 CPU를 할당하고 time slice가 지나면 강제적으로 프로세스에게서 제어권을 빼앗아 그 프로세스를 낮은 우선순위 큐의 끝에 붙이고 낮은 우선순위 큐의 앞에 있는 프로세스를 실행시키는 round robin 방식을 사용한다. 낮은 우선순위의 프로세스가 CPU를 점유하고 있을때 높은 우선순위의 프로세스가 준비상태가 되면 실행중이던 프로세스는 즉시 실행이 중단이 되고 프로세스 상태는 특정한 기억장소에 보관된다. 실행상태나 준비상태의 높은 우선순위 프로세스가 있는 경우에는 낮은 우선순위 프로세스는 실행되지 않는다. 생성되거나 대기상태에서 벗어나 준비상태가 된 프로세스는 항상 프로세스 큐의 끝에 붙게 된다. 이런 작동방식 때문에 높은 우선순위 프로세스라 할지라도 낮은 우선순위 back-pointer register의 값을 마음대로 바꾸지를 못한다. 높은 우선순위 프로세스가 back-pointer register 값을 바꾸는 도중에 낮은 우선순위 프로세스가 준비상태가 되면 하드웨어 스케쥴러가 즉시 현재의 back-pointer register가 가르키는 기억장소에 준비상태의 프로세스를 붙여버리므로 프로세스 큐가 불완전한 상태에 놓이게 되어 시스템이 다운된다.

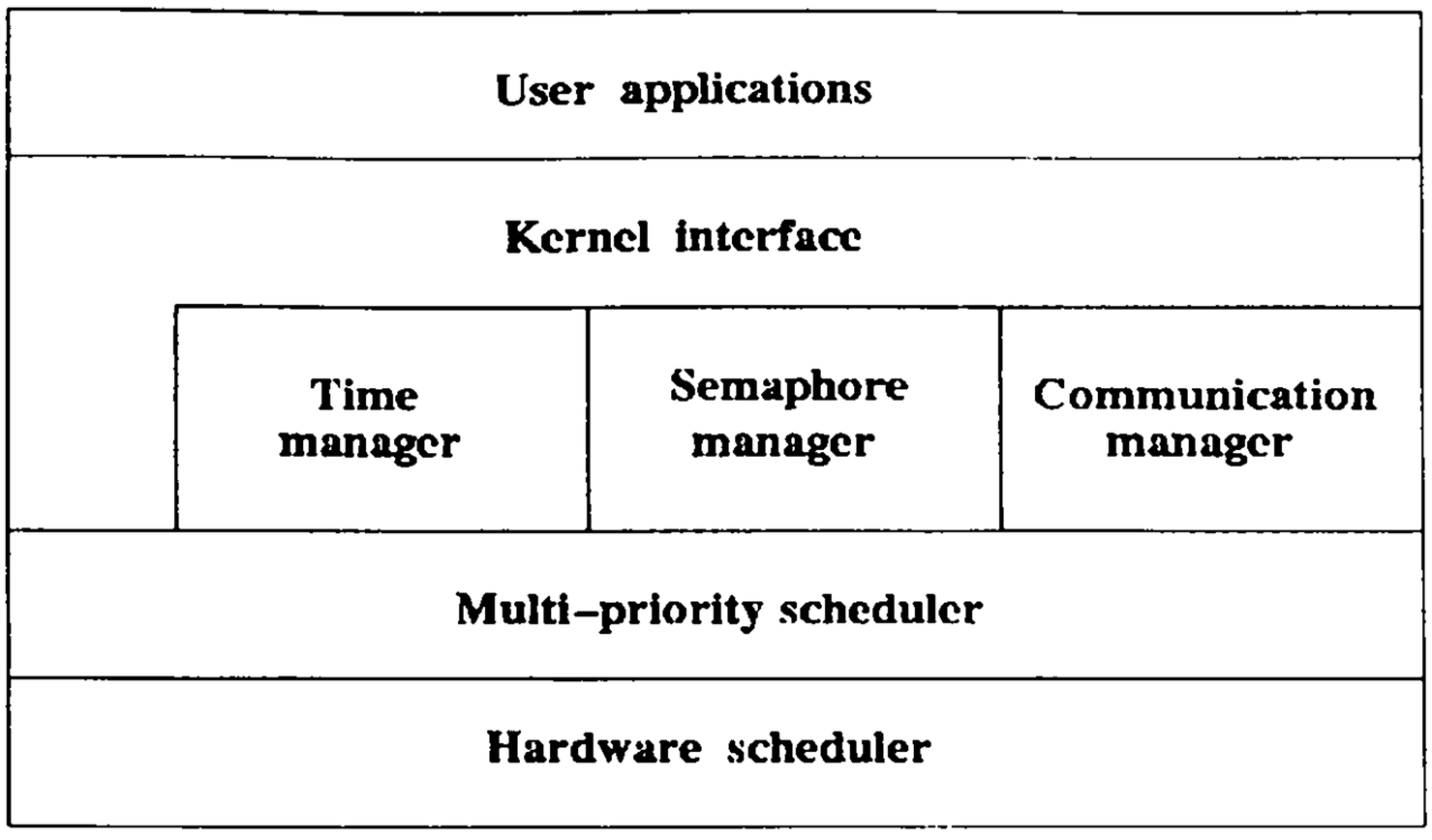


그림 11 : 시스템 구조

3. 이전 연구

트랜스퓨터를 실시간 시스템에서 사용하려면 다중 우선순위를 지원해야 하는데, 트랜스퓨터 자체는 2 단계의 우선순위만을 지원하기 때문에 실시간 시스템에 적합하지 않다. 트랜스퓨터로써 실시간 시스템을 구현하기 위해선 다중 우선순위 스케쥴러를 소프트웨어로 구현하고 그 위에서 실시간 응용프로그램을 실행시켜야 한다. 이미 트랜스퓨터상에서 다중 우선순위 스케쥴러를 구현하는 연구가 있어왔다 \cite{shea92,ckmp90,swb90,we90,ploeg94}. 그 중 \cite{ckmp90,swb90,we90}의 스케쥴러들은 고수준 언어인 OCCAM \cite{occam89}으로 구현되어 있으며 이로 인해 새로 준비상태가 된 프로세스는 우선순위나 긴급함에 관계없이 프로세스 큐의 끝에 붙게 되어 이미 준비상태인 다른 프로세스들이 CPU 를 양보하기 전까지는 실행할 수가 없었다. 이 때문에 발생하는 시간의 불예측성을 피하기 위해 응용프로세스들에게 CPU 를 양보하는 코드를 삽입시켰다. 이 방법은 복잡한 프로그램을 생성할 뿐만 아니라 고수준 언어로 구현되었기 때문에 효율이 떨어지는 문제가 있었다. \cite{shea92}는 낮은 우선순위 큐를 직접 다룸으로써 스케쥴러의 효율을 높인다.

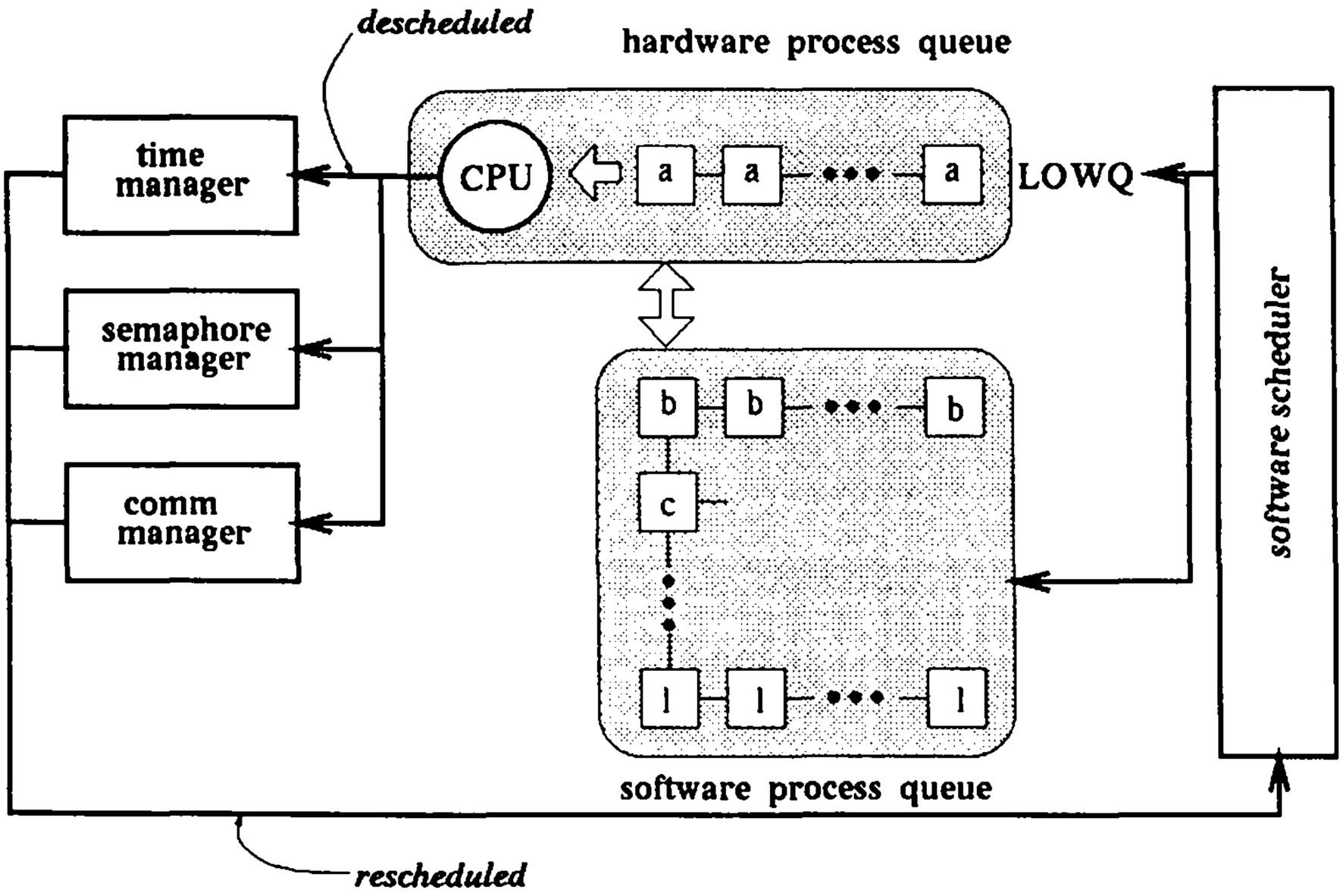


그림 2: 스케쥴러 동작의 개관 : n 단계의 우선순위가 있는 시스템

하지만 스케쥴러가 거의 주기적으로 호출이 되므로 같은 우선순위의 프로세스들만 실행되는 경우와 같이 큐를 다를 필요가 없는 경우에도 실행중인 프로세스를 자료구조에 넣고 자료구조에서 다시 빼내는 오버헤드가 존재한다. \cite{ploeg94}의 경우는 control-system 이라는 특수한 환경에서 실행되는 소프트웨어를 제어하므로 일반적인 다중 우선순위 스케쥴러와는 거리가 멀다.

본 보고서에는 \cite{shea92}의 스케쥴러의 문제점을 보완하기 위해 낮은 우선순위 큐의 크기가 변하는 사건이 발생했을 때만 호출되는 소프트웨어 스케쥴러를 구현하였다. 여기서 사건이 발생하는 경우는 프로세스 생성, 소멸, 대기가 발생한 경우, 세마포 대기, 세마포 대기 종료, 통신 대기, 통신 대기 종료 등이다. 이들 사건들의 발생을 스케쥴러에게 알리기 위해 사건 관리기가 필요하다. 사건 지향적 스케쥴러는 다음과 같은 두가지 성질에 의해서 스케쥴러의 부하를 줄이게 된다. 우선 이 소프트웨어 스케쥴러는 사건이 발생했을 때만 호출된다. 즉, 하드웨어 프로세스 큐의

크기가 바뀌었을 때만 호출되므로 같은 우선순위의 프로세스들이 실행되는 경우와 같이 하드웨어 프로세스 큐의 크기가 바뀌지 않는 경우에는 호출되지 않기 때문에 스케쥴러 자체의 호출횟수가 줄어든다. 두번째로 스케쥴러는 호출되더라도 항상 하드웨어 프로세스 큐를 재조정하지는 않는다. 이 상황은 사건의 종류를 관찰함으로써 쉽게 알 수 있다. 예를 들자면 중단된 프로세스가 대기상태로 바뀌었을 때 실행중인 프로세스와 같은 우선순위를 가지고 있으면 이 프로세스는 중단 큐에서 바로 하드웨어 프로세스 큐로 이동한다. 이 작업은 하드웨어 프로세스 큐를 재조정하고 자료구조에서 프로세스를 꺼내어 하드웨어 프로세스 큐에 넣는 것보다 적은 실행 시간을 요구한다.

4. 다중우선순위 스케쥴러의 구조

트랜스퓨터의 하드웨어 스케쥴러는 점프나 프로시져 호출과 같이 저장될 정보의 양이 최소화가 되기까지 기다린 후 문맥전환을 하므로 1μ sec 이하의 문맥전환 오버헤드를 가진다. 따라서 하드웨어 스케쥴러의 문맥전환을 이용한다면 상당히 적은 오버헤드로 문맥전환이 가능하다. 다중우선순위를 구현하는 소프트웨어 스케쥴러는 준비상태의 프로세스들 중에서 실행될 프로세스를 선택한다. 이 때 프로세스의 우선순위가 실행될 프로세스를 결정한다. 스케쥴러는 사건이 발생했을 때 호출된다. 스케쥴러는 스스로는 실행되지 않고 사건이 발생했음을 알리는 사건처리기에 의해서만 호출된다. 스케쥴러에 의해 선택된 프로세스는 낮은 우선순위 큐의 끝에 붙어서 실행된다. 그림 1 (a)는 응용 프로세스와 스케쥴러, 사건처리기, 하드웨어 스케쥴러들 상호간의 관계를 보여준다. 응용 프로세스들은 실제로 낮은 우선순위 하드웨어 스케쥴러에 의해 실행되며 우선순위가 같은 응용 프로세스들은 여러개가 동시에 실행된다. 이때는 하드웨어 스케쥴러의 제어를 받게 되어 round robin 방식으로 스케쥴된다. 소프트웨어 스케쥴러와 사건처리기는 하드웨어 스케쥴러에게 역시 프로세스로 인식된다. 이 프로세스들은 낮은 우선순위 프로세스 큐를 조절하고

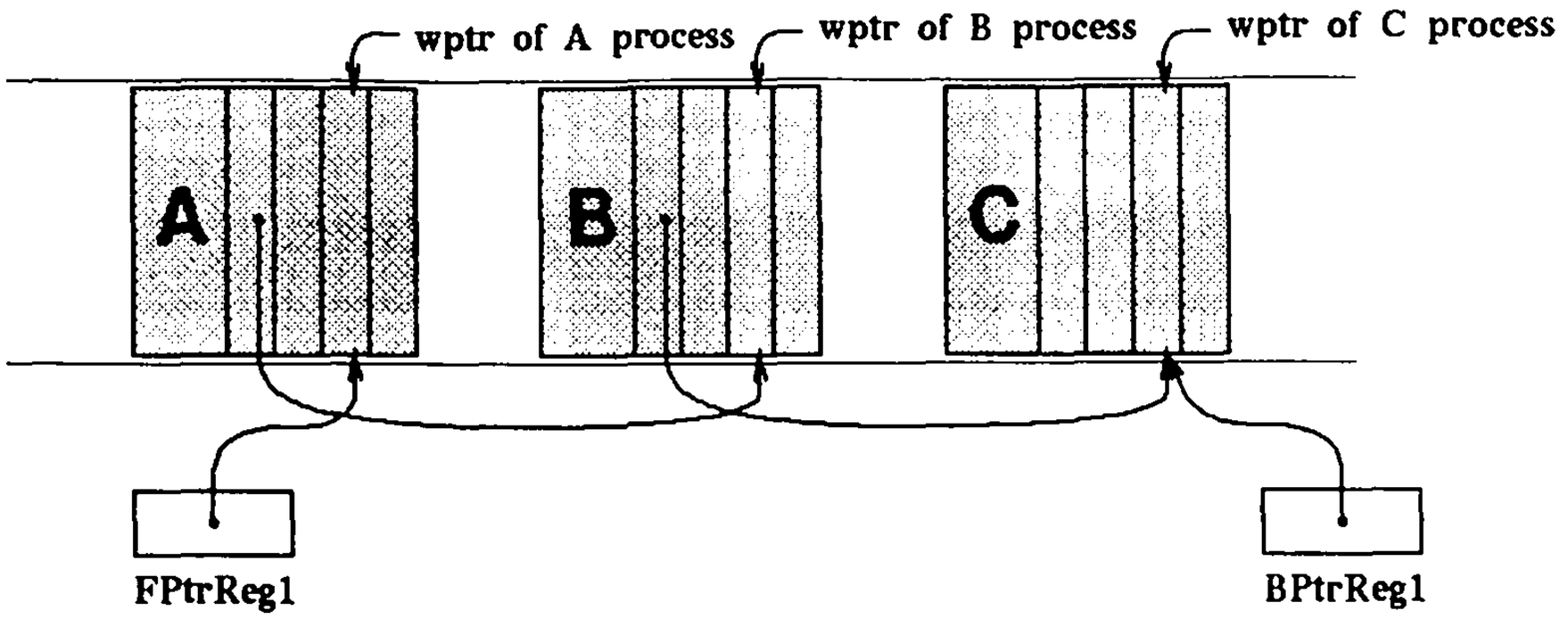


그림 3 : 하드웨어 큐 구조

응용 프로세스들을 제어하기 위해 그림 1 (b)와 같이 높은 우선순위 프로세스로서 하드웨어 스케쥴러와 사건처리기에 의해 다중우선순위를 가지고 실행된다.

일반적으로 응용 프로세스가 사건을 발생시키는 함수를 호출하면 사건처리기가 그 사실을 소프트웨어 스케쥴러에게 알려서 스케쥴러가 실행되도록 한다. 여기서의 사건이란 대기상태의 프로세스가 준비상태가 된다든지 새로운 프로세스가 생성되어 전체 우선순위 큐의 크기가 증가한다든지, 실행중인 프로세스가 대기 상태가 되거나 실행이 끝나게 되어 우선순위 큐의 크기가 감소하게 되는 경우를 말한다. 트랜스퓨터에서 프로세스가 대기상태가 되는 경우는 통신대기, 프로세스 대기/프로세스 대기 해제, 세마포 대기/대기 해제 등이 사건이 된다. 이들 함수는 대개 라이브러리로 지원되고 있으나 발생하는 모든 사건을 사건처리기가 알아야 하므로 사건처리기가 이들 함수를 구현한다.

그림 2 는 스케쥴러가 동작하는 개요를 보여주고 있다. 준비상태에 있는 프로세스들은 다중우선순위 프로세스 큐에 있으며 가장 우선순위가 높은 우선순위 큐를 하드웨어 스케쥴러가 직접 다루며 낮은 우선순위 프로세스 큐와 일치한다. 따라서 하드웨어 스케쥴러가 이 큐를 라운드 로빈 방식으로 스케쥴하며 그동안 소프트웨어 스케쥴러는 호출되지 않는다. 다른 우선순위 큐들은 소프트웨어 스케쥴러의 자료구조내에 저장된다. 따라서 이때는 스케쥴러의 오버헤드가 없게 된다. 사건처리기는 대기상태에서 준비상태가 된 프로세스를 스케쥴러에게 보내고 대기상태가 된 프

로세스를 스케쥴러로부터 받는다. 이때 스케쥴러가 호출되는데 이때의 알고리즘은 다음 장에서 설명한다.

하드웨어 프로세스 큐는 기억장치내에서 list 의 형태로 구현된다. 그림 3 은 낮은 우선순위 하드웨어 프로세스 큐의 FPtrReg1 이 큐의 첫번째 프로세스의 workspace(wptr)를 가리키며 첫번째 프로세스 workspace 의 -2 위치(wptr-2)에 두번째 프로세스의 wptr 를 가리키는 pointer 값이 존재한다. 큐의 마지막 프로세스의 wptr 은 BPtrReg1 가 가리키며 마지막 프로세스의 (wptr-2)값은 정의되지 않는다.

5. 스케쥴링 알고리즘

우선순위가 가장 높은 우선순위 큐를 제외한 나머지 우선순위 큐들은 스케쥴러의 자료구조에 저장된다. 이 자료구조는 그림 4 에서 보는 바과 같이 같은 우선순위를 가진 프로세스들이 하나의 리스트를 형성하고 이들 리스트들이 우선순위의 순서대로 전체 리스트를 형성한다. 여기서 보관되는 프로세스들은 준비상태에 있으며 우선순위가 낮아서 하드웨어 스케쥴러에 의해 실행되지 않는다. 우선순위가 i 인 P1i, P2i, ... , Pmi 프로세스들이 하나의 리스트를 이루고 있고 외부에서 필요로하는 프로세스는 우선순위가 가장 높은 프로세스들이므로 상수시간에 프로세스 큐를 자료구조로부터 꺼낼 수 있다. 자료구조에 프로세스가 들어오게 되면 최대 프로세스의 우선순위만큼 전체 리스트를 검색한 후 자신의 순서에 해당하는 위치의 큐에 삽입된다.

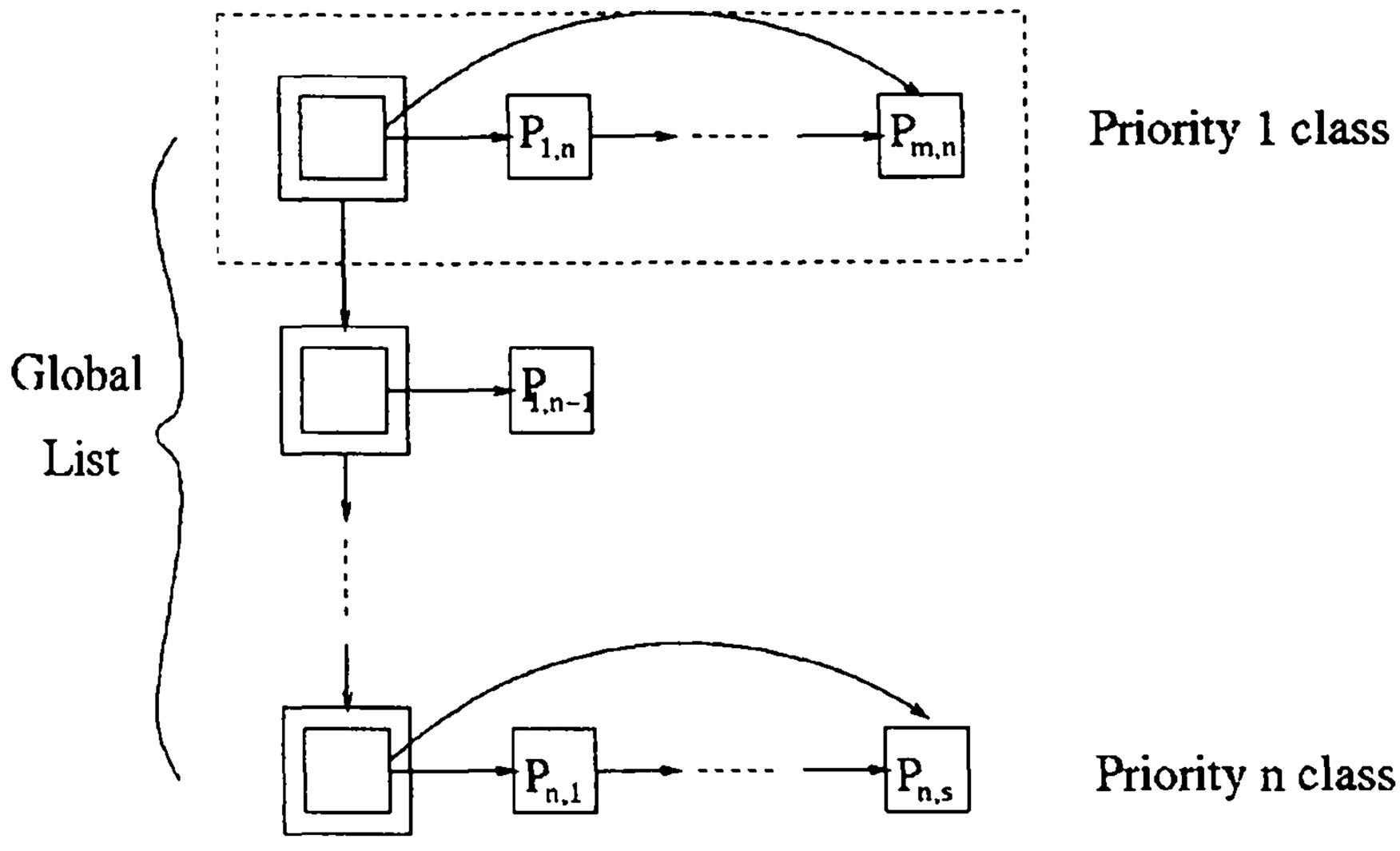


그림 4: 스케쥴러의 자료구조

다중우선순위 스케쥴러는 그림 5와 같이 동작한다. 스케쥴러는 각 사건의 종류에 따라 그에 적절한 행동을 취하게 된다. 사건은 크게 프로세스가 우선순위 큐에서 빠져나가는 경우와, 우선순위 큐로 들어오는 두가지 경우로 나뉜다. 한 프로세스가 우선순위 큐로 들어오는 경우는 그 프로세스의 우선순위와 우선순위 큐내의 가장 높은 우선순위에 의해 3가지 경우로 다시 나누어진다. 프로세스가 우선순위 큐에서 나가는 경우는 CPU에서 실행되던 프로세스가 실행이 중단되는 경우로 우선순위가 가장높은 프로세스 큐, 즉 하드웨어 프로세스 큐의 길이가 0인 경우와 0보다 큰 경우 두가지로 나누어진다.

- 준비상태가 된 프로세스 우선순위가 실행중인 프로세스의 선순위보다 낮은 경우

새로들어온 프로세스는 실행중인 프로세스 큐보다 우선순위가 낮은 큐에 삽입된다. 이 큐는 스케쥴러의 자료구조에 존재하므로 프로세스는 하드웨어

큐에 삽입되지 않고 바로 자료 구조로 들어간다. 따라서 이때는 스케쥴러가 하드웨어 큐를 다룰 필요가 없다.

- 준비상태가 된 프로세스의 우선순위가 실행중인 프로세스의 우선순위와 같은 경우

이 경우는 준비상태가 된 프로세스가 가장 높은 프로세스이므로 하드웨어 프로세스 큐의 끝에 붙게된다. 이때도 스케쥴러는 큐를 다룰 필요가 없다. 또한 우선순위 큐 전체가 비게되어 하드웨어 프로세스 큐에 아무런 프로세스가 없게 될 경우에도 새로운 프로세스는 하드웨어 프로세스 큐의 끝에 붙게된다.

- 준비상태가 된 프로세스의 우선순위가 실행중인 우선순위보다 높은 경우

이때는 실행중인 프로세스를 인터럽트하고 하드웨어 프로세스 큐에 있는 프로세스들과 실행이 중단된 프로세스를 소프트웨어 스케쥴러의 자료구조에 넣는다. 준비상태가 된 프로세스는 하드웨어 프로세스 큐의 끝에 붙게 되고 다른 프로세스가 하드웨어 프로세스 큐에 없으므로 곧바로 실행된다. 하지만 실제로는 실행중인 프로세스의 상태에 대한 자세한 정보를 알 수 없어서 실행중인 프로세스를 곧바로 제거할 수 없다. 따라서 time slice 가 경과하거나 대기상태, 실행이 끝나서 스스로 CPU 에서 제거될 때까지 기다리게 된다. 다른 경우와 달리 이 경우에 유일하게 low queue front pointer(BPtrReg1)의 값과 하드웨어 프로세스 큐를 다루게 되어 많은 오버헤드가 발생한다. 이 경우의 예제를 나중에 설명하기로 한다.

```
IF a process enters into ready state THEN
        IF new process priority < current priority THEN
                insert the process into Data Structure
        ELSE IF new process priority = current priority THEN
                insert the process into LOWQ
        ELSE IF new process priority > current priority THEN
                reset LOWQ and insert the process into LOWQ
ELSE IF a process enters into block state or terminate THEN
        IF LOWQ is empty THEN
                restore highest priority class in DS to LOWQ
        ELSE IF LOWQ is not empty THEN
                do-nothing
```

그림 5 : 제안된 스케쥴러의 전체적인 알고리즘

- 실행중이던 프로세스가 종료하거나 대기상태가 되고 하드웨어 프로세스 큐가 비어있는 경우

하드웨어 프로세스 큐에는 실행될 프로세스가 없으므로 스케쥴러의 자료구조가 프로세스가 1개 이상 있다면 자료구조내의 프로세스 큐중 우선순위가 가장 높은 우선순위 큐가 하드웨어 프로세스 큐로 옮겨진다. 이때는 각 큐의 프로세스를 앞부분부터 하나씩 꺼내어 하드웨어 프로세스 큐의 끝에 옮겨지며 이 동작은 트랜스퓨터 어셈블리 명령인 runp 명령으로 구현한다.

- 실행중이던 프로세스가 종료하거나 대기상태가 되고 하드웨어 프로세스 큐가 비어있지 않은 경우

소프트웨어 스케쥴러는 현재 하드웨어 프로세스 큐의 프로세스들 갯수가 1개 감소한 것만 확인한다.

그림 6은 준비상태가 된 프로세스의 우선순위가 실행중인 프로세스의 우선순위보다 높은 경우의 예를 보여준다. 시간은 (a),(b),(c), ... ,(h)의 순으로 경과하며 사각형은 CPU를 나타내며 원은 응용 프로세서를 나타낸다. (a)시간에 lp1-프로세스가 CPU를 점유하고 있으며 이와 같은 우선순위를 가지는 lp2,..,lpn의 프로세스들이 준비상태로 낮은 우선순위 하드웨어 프로세스 큐에 있으며 한 프로세스가 준비상태가 된 것을 소프트웨어 스케쥴러가 알게 된다. 이때 소프트웨어 스케쥴러는 CPU를 점유한 프로세스와 하드웨어 프로세스 큐에 있는 스케쥴러를 모두 자료구조로 옮기게 된다.

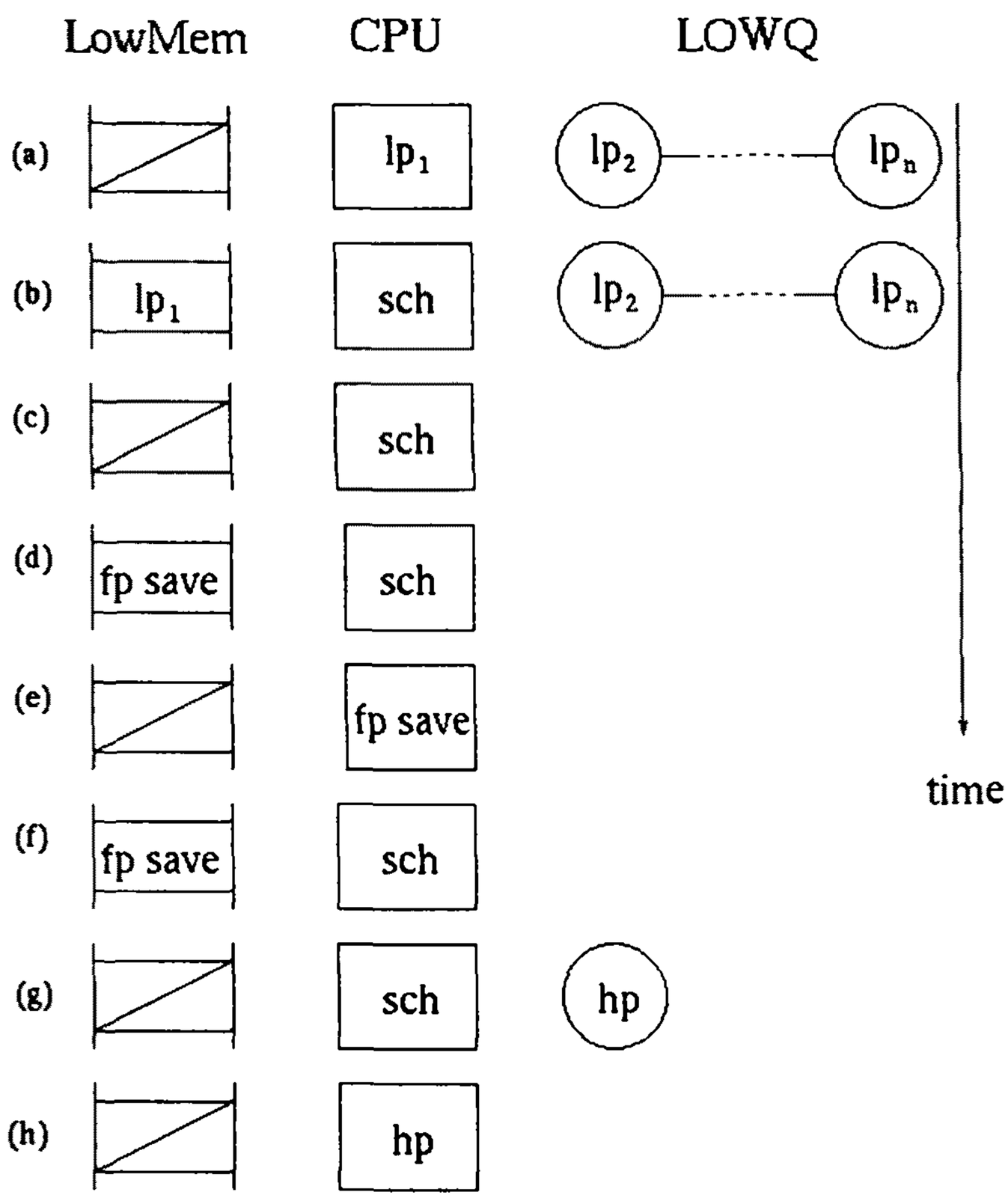


그림 6: 낮은 우선순위 프로세스들을 자료구조로 옮기는 알고리즘

우선 높은 하드웨어 우선순위인 소프트웨어 스케쥴러(sch)가 실행중인 낮은 하드웨어 우선순위를 가지는 응용 프로세스(lp1)를 인터럽트한다. 인터럽트 당한 lp1 의 Wdesc, Iptr, register, 와 status 들은 특정 메모리 영역인 LowMem 에 저장된다(그림 6.(b)). sch 는 LowMem 의 정보들과 LOWQ 내의 프로세스들을 스케쥴러의 자료구조 내에 저장하고 LowMem 과 FPtrReg1 register 의 값을 Process_p 값으로 설정해 인터럽트된 프로세스나 실행을 기다리는 프로세스가 없도록 한 다(그림 6.(c)). LowMem 에는 부동 소숫점 연산을 위한 레지스터의 값은 보관되지 않는다. 이들 부동 소숫점 레지스터들은 부동 소숫점 계산 유니트(FPU)내에 기억되며 같은 하드웨어 우선순위의 프로세스만 이들 레지스터의 값들에 접근할 수 있다. 따라서 이들 레지스터 값

들을 저장하기 위해서 fp save란 낮은 하드웨어 우선순위 프로세스를 사용한다. 스케쥴러가 fp save의 wptr과 iptr값을 LowMem에 넣은 뒤(그림 6.(d)) 스스로 블럭되면 fp save는 마치 인터럽트 되었던 것처럼 수행을 시작해서 FPU의 레지스터들을 자료구조에 보관한다(그림 6.(e)). fp save가 저장을 끝내면 스케쥴러에게 다시 메시지를 보내서 스케쥴러를 깨운다(그림 6.(f)). 스케쥴러는 fp save를 인터럽트해서 fp save를 LowMem에서 제거하고 새로 실행된 프로세스인 hp를 LOWQ에 붙인다. 스케쥴러는 다시 스스로 블럭되고 LOWQ에 있던 hp가 실행된다.

통신대기 큐나 time wate 큐에 대기중인 프로세스가 대기상태에서 준비상태가 되면 하드웨어 프로세스 큐의 끝에 붙게 된다. 이 동작은 하드웨어에 의해 직접 발생하므로 다중우선순위 스케쥴러같은 높은 우선순위 프로세서도 이 움직임을 알지 못한다. 실행중인 응용프로세서의 우선순위보다 준비상태가 된 프로세서의 우선순위가 높은 경우에는 현재 하드웨어 프로세스 큐를 자료구조로 내려보내고 그 반대의 경우에는 준비상태의 프로세스를 하위 프로세스 큐에 넣어야 하기 때문에 스케쥴러는 이 프로세스의 우선순위를 알아야 한다. 하지만 하드웨어 스케쥴러는 한 프로세스가 준비상태가 되었음을 소프트웨어 스케쥴러에게 알리지 않으므로 소프트웨어 스케쥴러를 다른 방법으로 준비상태가 되는 프로세스의 우선순위를 알아야 한다. 한가지 방법은 스케쥴러가 주기적으로 하드웨어 프로세스 큐를 검사해서 새로운 프로세스가 삽입되었는지를 조사하는 것이다. \cite{shea92}는 이 방법을 사용하고 있으며 스케쥴러가 주기적으로 호출되는 오버헤드가 발생한다. 여기서는 통신대기나 프로세스대기, 세마포대기를 스케쥴러 이외의 다른 높은 우선순위 프로세스인 사건처리기에 맡긴다. 통신, 프로세스대기, 세마포 관련 함수는 모두 사건처리기를 호출하고 필요한 정보를 전달한다. 사건처리기는 내부에 통신대기 큐, 프로세스대기 큐, 세마포대기 큐를 관리하며 대기해야 할 프로세스를 큐에 넣고 대기가 끝난 프로세스를 큐에서 꺼낸다. 이때 준비상태가 된 프로세스를 하드웨어 큐에 넣지 않고 소프트웨어 스케쥴러를 호출한 후 소프트웨어 스케쥴러에게 프로세스를 넘겨

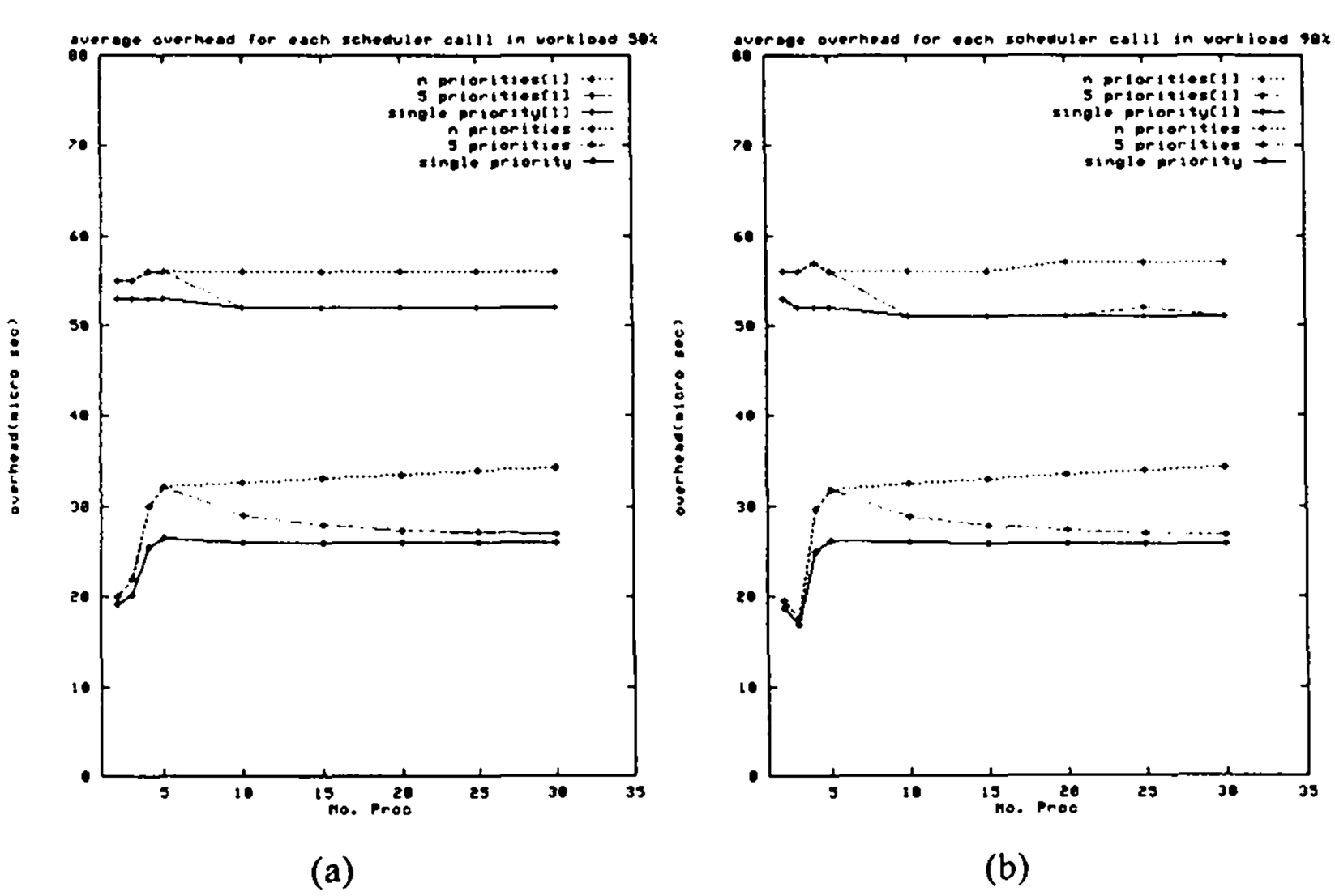
준다. 소프트웨어 스케쥴러는 주기적으로 하드웨어 큐를 검사할 필요없이 새로 준비상태가 된 프로세스를 스케쥴링 알고리즘에 따라 처리하면 된다.

6. 실험 결과

스케쥴러는 INMOS ANSI C \cite{ansi_user,ansi_ref}와 트랜스퓨터 어셈블러 \cite{trans_inst}로 구현했으며 성능비교를 위해 \cite{shea92}의 스케쥴러를 INMOS C로 바꾸어 실행했다. 실험 결과는 스케쥴러를 10번 실행한 값들의 평균을 사용했다. 실행 횟수는 적지만 결과값들의 분산값은 1μ sec 이내를 나타낸다. 실험에 사용된 응용 프로세스들은 주기적인 프로세스들로 rate monotonic algorithm\cite{liu73}에 의해 우선순위가 할당되었다. 공정한 비교를 위해 같은 실험환경상에서 \cite{shea92}를 실행시켰다. 전체시간에서 프로세스가 CPU를 점유한 시간의 비율을 50%와 90%로 두었으며 우선순위의 분포를 3가지 경우로 나누었다. 즉 각 프로세스가 다른 우선순위를 가지는 경우, 5개의 우선순위 단계가 있는 경우, 모든 프로세스가 같은 우선순위를 가지는 경우 등이다. 그림 10에 실험결과가 나타나 있으며 \cite{shea92}의 알고리즘과 비교해 볼 때 모든 경우에 더 나은 성능을 보여주고 있다. \cite{shea92}의 알고리즘은 모든 경우에 거의 일정한 오버헤드를 나타내고 있는데 이는 호출되는 상황에 관계없이 하드웨어 프로세스 큐의 재설정과 프로세서의 이동이라는 거의 동일한 작업을 수행하기 때문이다.

표 1은 각 사건 형태에 따른 스케쥴러 오버헤드를 나타낸다. 본 실험은 프로세스 생성, 프로세스 대기 종료, 그리고 프로세스 대기의 3가지 사건 형태에 프로세스 호출이 집중되고 있다. 우선 프로세스 생성의 경우에 대부분의 프로세스들은 실행되는 프로세스의 우선순위보다 낮기 때문에 스케쥴러의 자료구조에 삽입된다. 모든 프로세스들이 같은 우선순위를 가지는 환경에서는 자료구조가 우선순위에 의해서 정렬되어 있지 않기 때문에 자료구조의 리스트를 따라가면서 자신의 위치를 찾지 않고 바로 삽입된다. 반면 모든 프로세스들이 다른 우선순위를 가지는 경우에는 새

로 생성된 프로세스들이 자료구조에 삽입될 때 자료구조의 리스트를 따라서 자신의 우선순위에 적합한 위치에 삽입되기 때문에 오버헤드가 늘어난다. 하지만 프로세스 생성은 30 회만 발생했기 때문에 전체 스케쥴러 오버헤드의 큰 부분을 차지하지 않는다.



(a) (b)

그림 7 : (a)프로세스 부하가 50%일 때의 스케쥴러 오버헤드를 프로세스의 갯수를 변화시키며 측정 (b)프로세스 부하가 90%일 경우

두번째 경우는 프로세스 대기 종료의 경우이다. 대기상태에서 막 준비상태가 된 프로세스가 현재 실행중인 프로세스보다 낮은 우선순위를 가진 경우에는 프로세스가 스케쥴러의 자료구조에 삽입된다. 이때의 오버헤드는 약 50μ sec 이다. 반면 실행중인 프로세스와 같은 우선순위를 가진 경우에는 준비상태가 된 프로세스는 LOWQ의 끝에 붙게 된다. 이때의 오버헤드는 약 15 μ sec 이 된다. 이 프로세스가 실행중인 프로세스보다 높은 우선순위를 가진 경우에는 LOWQ에 있는 프로세스들을 모두 자료구조로 옮기고 새 프로세스를 실행시킨다. 이때의 오버헤드는 약 51 μ sec 이

된다. 모든 프로세스들이 같은 우선순위를 가지면 준비상태가 된 프로세스는 모두 LOWQ에 붙게 된다. 반면 모든 프로세스들이 다른 우선순위를 가지면 다른 두가지 경우가 많이 발생하게 되므로 같은 우선순위의 경우보다 많은 오버헤드를 가지는 작업을 수행한다.

마지막으로 프로세스 대기가 시작되는 경우에는 다른 프로세스가 LOWQ에 남아 있는 경우와 그렇지 않은 경우가 있는데 전자의 경우에는 별도의 작업이 필요없지만 후자의 경우에는 스케쥴러의 자료구조로부터 실행될 프로세스들을 LOWQ로 이동시켜야 하기 때문에 48μ sec 정도의 시간을 요하는 작업이 필요하다. 모든 프로세스들의 우선순위가 같은 경우에는 대부분의 프로세스들이 LOWQ에 있기 때문에 별도의 작업이 필요하지가 않지만 모든 프로세스들의 우선순위가 다른 경우에는 항상 LOWQ가 비게된다. 즉, 같은 경우에 스케쥴러가 호출되더라도 우선순위의 차이에 의해서 오버헤드가 달라지게 된다. 특히 우선순위가 다른 경우에는 비교적 많은 실행시간을 요구하는 작업이 대부분이므로 우선순위가 같은 경우보다 많은 오버헤드를 나타낸다.

스케쥴러 호출 상황			1-우선순위		n-우선순위	
사건	형태	우선순위 / 큐 길이	횟수	평균시간	횟수	평균시간
프로세스 대기	생성	Pnew < Pcur	30	80	30	162
		Pnew = Pcur	0	0	0	0
		Pnew > pcur	0	0	0	0
	종료	LOWQ 가 비어있지 않음	14	13	0	0
		LOWQ 가 비어있음	0	0	29	50
프로세스 대기	종료	Pnew < Pcur	0	0	122	51
		Pnew = Pcur	152	15	0	0
		Pnew > pcur	10	29	40	52

	생성	LOWQ 가 비어있지 않음	152	13	0	0
		LOWQ 가 비어있음	10	18	162	48

표 1 : 프로세스의 CPU 부하가 50%, 90%일 때 1-우선순위 단계와 n-우선순위 단계의 비교, Pnew 는 준비상태가 된 프로세스의 우선순위이고 Pcur 는 현재 실행되고 있는 프로세스의 우선순위를 나타낸다.

프로세스 대기, 세마포 대기, 통신 대기, 대기 종료 등의 사건이 발생했음을 다중우선순위 스케쥴러에 알리는 사건 처리기의 실행 시간의 분석 결과는 다음과 같다. 사건 처리기는 크게 세마포 대기 처리기와 프로세스 대기 처리기 두개의 모듈로 나누어 진다. 세마포 대기 처리기는 세마포 대기에 의해 대기 상태가 된 프로세스들을 세마마포 리스트에 우선순위순으로 보관한다. 실험결과 세마포 대기는 약 $(58+0.05\times n)\mu$ sec 의 오버헤드를 가진다. 여기서 n 은 세마포 리스트의 길이를 나타낸다. 세마포 대기 종료로 세마포 대기 처리기가 불려질 때는 리스트의 가장 앞에 놓인 프로세스가 선택되므로 리스트를 검색하는 오버헤드는 생기지 않는다. 이때의 오버헤드는 약 58 μ sec 이 된다. 프로세스 대기 처리기도 스케쥴된 시간에 스스로 리스트내의 프로세스를 깨우는 명령어를 제외하면 세마포 대기 처리기와 같은 구조를 가진다. 프로세스를 리스트에 입력할 때는 약 (50+0.05 n)μ sec 를, 리스트에서 프로세스를 선택할 때는 약 50μ sec 의 오버헤드를 나타낸다.

7. 결론

본 연구에서 제안한 다중우선순위 스케쥴러는 우선순위 큐의 상태가 변하는 사건이 발생했을 때만 호출되므로 주기적으로 호출되는 프로세스보다 호출횟수가 적기 때문에 전체 실행시간에서 스케쥴러가 차지하는 비율이 줄어든다. 또한 스케쥴러가 호출된 경우라도 하드웨어 프로세스 큐를 다루는 경우가 적기 때문에 호출 횟수

당 오버헤드도 줄어든다. 더우기 모든 프로세스가 같은 우선순위를 가진 경우에는 스케쥴러의 오버헤드는 프로서스의 갯수에 무관하게 25μsec 정도로 나타나게 되어 \cite{shea92}의 알고리즘의 오버헤드인 53μsec 보다 적게 나타난다.

스케쥴링 알고리즘 구현을 위해 본 연구에서는 C 언어의 malloc()과 같은 동적 할당 함수를 사용해서 스케쥴러의 자료구조를 관리했으나 실제로 실행환경에 대한 정보를 알 수 있 경우에는 정적 할당으로 스케쥴러의 자료구조를 관리한다면 더욱 낮은 오버헤드를 나타내리라고 본다.

5 절 결 론

본 연구에서는 병렬 실시간 시스템의 환경 구축을 위해 병렬 시스템인 TiME 상에서 다중 우선순위 스케쥴러를 개발했다. 구현된 스케쥴러는 프로세스큐의 상태가 변화할 때만 호출됨으로써 스케쥴러의 부하를 줄였다.

또한 로봇 시뮬레이터를 병렬 시스템 상에서 구현함으로써 병렬 시스템이 시뮬레이션 환경에 적합한지를 테스트하였다. 로봇 시뮬레이터를 구현한 결과 병렬 시스템은 복잡한 시뮬레이션 환경에 적합함을 알 수 있었다. 또한 로봇 시뮬레이터를 구현한 노우하우는 다른 시뮬레이터 개발에 이용될 수 있을 것이다

여 백

제 1 차년도
연차 보고서

RTS 기술개발

RTS Technology Development

미시적 시뮬레이터 구축 기술 개발

연구기관
시스템공학연구소

과 학 기 술 처

여 백

제 4 장 미시적 실시간 시뮬레이터 구축 기술 개발

1 절 서론

1 연구의 배경

미시적 실시간 시뮬레이션은 실시간 시스템을 구성하는 프로세서들의 기능적 행위(Functional Behaviors)와 시간적 행위(Temporal Behaviors)를 정밀하게 모형화하고 이를 고성능의 컴퓨터를 이용하여 실시간으로 실행하여 봄으로써 실시간 시스템의 구성 요소간의 유기적인 관련성과 영향을 분석 평가하는 기술이다. 실시간 시스템의 정밀한 모델링을 위하여 실시간 시스템을 구성하는 구성 요소를 객체 지향 모델링 기법을 이용하여 모델링할 때 객체의 기능적 행위와 시간적 행위를 상세히 기술할 수 있는 기법이 필요하다. 또한 기술된 모델을 시뮬레이션 하려면 시뮬레이션을 수행시키는 엔진과 프로그램 언어와 유사한 시뮬레이션 언어의 개발이 요구된다.

일반적인 실시간 시스템은 수십, 또는 수백 개의 프로세서의 집합으로 구성되어 있다. 기존의 시뮬레이션은 단일 프로세서 상에서 시뮬레이션을 실행하고 있기 때문에 실시간성의 검증이 약화되고 시뮬레이터의 성능도 저하되는 경향이 있다. 최근에는 분산 및 병렬 처리 컴퓨팅이 실용화되어 분산 병렬 컴퓨터 환경을 이용한 고성능의 실시간 시뮬레이션이 가능하고, 그 활용이 중요시되고 있다. 미시적 실시간 시뮬레이션 기술은 이러한 사실을 근거로 개발되고 있다.

이러한 고성능, 고정밀의 실시간 시뮬레이션을 통하여 신뢰성 있는 실시간 시스템의 개발이 이루어지게 되고, 대규모의 복잡한 실시간 시스템의 최적 설계와 효율적인 운영에 긴요한 의사 결정 정보가 제공되므로 미시적 시뮬레이터 기술개발이 필연적으로 요구된다고 할 수 있다.

미시적 실시간 시뮬레이션 기술은 다음과 같은 측면에서 경제, 사회적 효과를 가져다 준다.

- 실제로 실시간 시스템을 구현하면 위험 부담이 크고 막대한 비용이 소요될 때, 고정밀 고성능 실시간 시뮬레이션은 실시간 시스템의 경제적, 기술적 타당성을 적은 비용으로 신속하게 파악할 수 있게 도와준다.
- 실시간 시스템은 첨단장비의 핵심 제어 기술로 사용되므로 실시간 시뮬레이션 기술은 첨단 산업 발전에 매우 중요한 기술이다.
- 실시간 시뮬레이션 기술은 고정밀의 훈련용 시뮬레이터, 대규모 실시간 시스템 개발 도구 및 시스템 성능 평가 도구 등으로 사용되며 타 산업의 기술 파급 효과가 매우 크며 그 응용 분야도 매우 넓다.
- 실시간 시뮬레이션은 견고한 실시간 시스템 구축에 매우 긴요한 기술로서 실시간 시스템이 오류 동작 시에 발생하는 막대한 재해를 막을 수 있다.
- 실시간 시뮬레이션은 교통 및 통신 분야에 응용되며 이들 시스템의 성능과 안전성을 제고하여 안락하고 안전한 사회, 문화생활에 기여한다.

실시간 시스템 및 시뮬레이션 기술은 미국, 유럽, 일본 등지의 선진국에서

주로 개발되어 후발 국가에 수출하고 있는 실정이다. 그러나 이 분야의 기술은 선진국에서 조차 정립되지 못하고 있으며, 자체 보유 기술을 개발 도상 국가로 이전하는 것을 매우 꺼리고 있는 실정이다. 최근 국내에 실시간 시스템의 도입이 급속히 확산되어 각 분야에 활발히 적용하고 있으나 국내 기술 수준은 현재 기초 단계로서 대부분의 시스템을 수입에 의존하고 있다.

실시간 시뮬레이션 모델링 기술은 프로세스 중심의 모델링 방법에서 객체 지향 기법을 지향하는 추세이며 그간 다수의 객체 지향 기법이 제안되었지만 실시간 시스템의 특성을 포괄적으로 표현하는 모델링 기법은 거의 찾아 보기 힘들다. 국내 현실은 선진국에서 개발한 프로세스 중심의 고전적인 모델링 기법을 주로 답습하고 있는 실정이다.

실시간 시뮬레이션의 적용 분야는 실시간 시스템의 다양화와 함께 빠른 속도로 확산되고 있으나 포괄적이고 체계적인 접근 방법이 개발되지 못하고 있다. 국내에서도 많은 업체가 실시간 시스템을 도입하고 있으나 도입된 시뮬레이터의 운영 기술 확보에 치중하고 있는 형편이다. 가끔 선진국에서 개발한 시뮬레이터의 확장 등을 통하여 개발 기술 확보 노력을 하고 있으나 기본 기술의 부족 등으로 애로를 겪고 있다.

위에서 살펴본 바와 같이 실시간 시뮬레이션 기술의 효과 및 다양한 적용 분야에 비해 기본 기술 확보가 매우 미흡한 실정이다. 따라서 선진국의 기술 장벽을 넘어서 기술 자립을 하기 위해서는 이 분야의 기초 기술 확보를 서두를 필요가 있다. 본 과제는 이러한 배경하에서 실시간 시뮬레이션의 기본 기술을 정립하기 위하여 특정 응용 분야에 치우치지 않는 포괄적인 개발 기술을 정립하고자 한다. 즉, 실시간 모델링, 실시간 시뮬레이션 언어 및 시뮬레이

션 엔진 등의 개발을 통하여 실시간 시뮬레이션의 통합적인 개발 환경을 구축하고자 하는 것이다.

2. 연구의 목표

본 연구의 최종 목표는 실시간 객체 지향 모델링 기법 정립을 통한 실시간 시뮬레이션의 개발 환경을 구축하고 이를 이용한 지능형 생산 공장 시뮬레이터를 구축하는 데 있다.

본 연구는 3 년에 걸쳐 연차적으로 다음과 같은 목표를 두고 있다.

- 제 1 차 년도 : 대규모 실시간 시뮬레이션 조사 및 미시적 실시간 시뮬레이션 모델 정립
- 제 2 차 년도 : C++T 언어 처리기 및 실시간 시뮬레이션 실행 지원 개발, 실시간 데이터 베이스 설계 및 미시적 시뮬레이션 모델 구축
- 제 3 차 년도 : 병렬 처리 지원 기능 개발, 그래픽 시뮬레이션 도구 개발 및 지능형 생산공장 시뮬레이터 구축

2 절 실시간 시뮬레이션의 소개

1. 실시간 시스템

가. 실시간 시스템의 정의

실시간 시스템이 가져야 하는 특성에 대해서는 많은 해석들이 있다. 그러나 모든 해석들의 공통점은 시스템의 응답 시간에 주의를 기울이고 있다. 즉 시스템이 관련된 입력으로부터 출력을 생성하는데 소요되는 시간에 관심을 두고 있다. 옥스포드 컴퓨터 사전에는 다음과 같이 실시간 시스템을 정의하고 있다.

"출력이 생성되는 시간이 중요한 시스템. 이것은 보통 입력이 실세계에서의 활동에 대응되고 출력이 동일한 활동에 관련되어야 하기 때문이다. 입력시간에서 출력시간까지 지연 시간은 허용되는 범위내에서 적시성에 맞게 충분히 작아야 한다."

위 정의에서는 언급된 적시성(timeliness)의 예들은 상황에 따라 다르다. 미사일 유도 시스템의 출력은 수 millisecond 내로 요구되는데 비해 컴퓨터 제어 자동차 조립 공정의 응답은 일초 이내일 경우도 있다

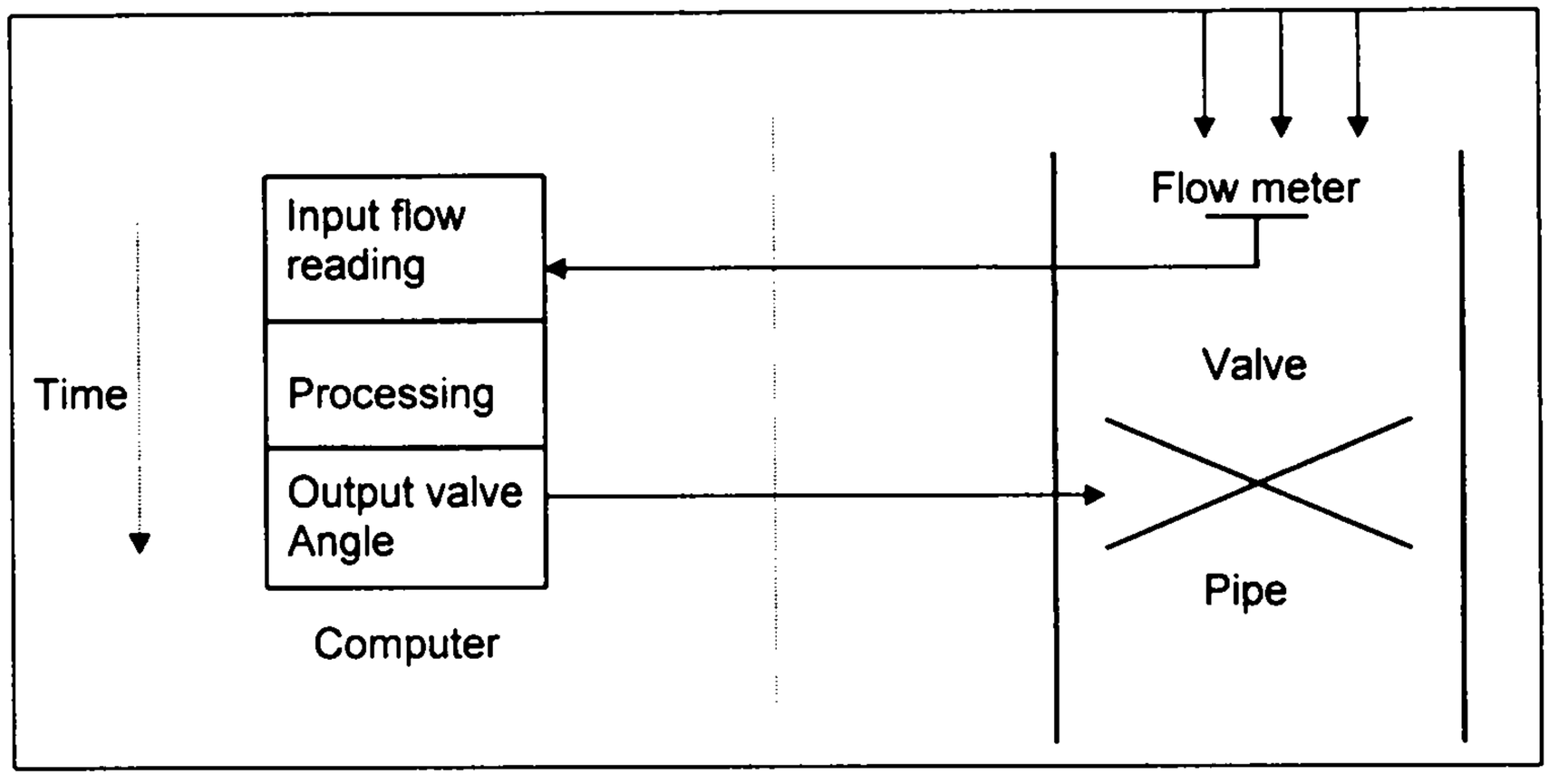
Young(1982)은 실시간 시스템을 다음과 같이 정의하였다.

"외부로 부터 생성되는 입력 자극에 대해 유한하고 규정된 기간안에 응답하여야 하는 정보 처리 활동 또는 시스템."

일반적인 의미에서 위의 두 정의는 컴퓨터 활동들의 매우 넓은 범위를 포함하고 있다. 예를 들어 UNIX 와 같은 운영체제는 사용자가 명령을 입력하고

수초내에 응답을 기대할 수 있으므로 실시간으로 간주된다. 다행스럽게도 이것은 보통 응답이 곧 나오지 않더라도 재난을 일으키는 것은 아니다. 이런 유형의 시스템들은 응답의 실패가 잘못된 응답과 마찬가지로 나쁜 것들과 구별된다. 실제로 어떤 것에 대하여는 이러한 측면은 실시간 시스템을 다른 것, 즉 응답시간이 중요하나 결정적이지는 않는 것과 구별하는 척도가 된다. 결론적으로 말하면 실시간 시스템의 정확성은 계산의 논리적인 결과는 물론 결과가 생성되는 시간에 달여 있다는 것이다. 실시간 컴퓨터 시스템 설계 분야에 종사하는 종사자들은 실시간 시스템을 경성(hard)과 연성(soft) 실시간 시스템으로 구분한다. 경성 실시간 시스템은 응답이 규정된 마감시간(deadline)내에서 반드시 발생되어야 함을 절대적으로 강요한다. 연성 실시간 시스템은 응답시간이 중요하나 마감시간을 때때로 어겨도 시스템이 제대로 기능하는 것이다. 연성 시스템은 명확한 마감시간이 없는 대화형 시스템과는 구별된다. 예를 들어 전투기의 항공 제어 시스템은 경성 실시간 시스템이다. 왜냐하면 마감시간이 지나면 파국으로 빠지기 때문이다. 그러나 공정 제어의 데이터 획득 시스템은 연성 시스템이다. 왜냐하면 이것은 입력 센서(sensor)를 일정한 간격으로 샘플링(sampling)하도록 정의되어 있으나 간헐적인 지연은 허용하기 때문이다.

경성 또는 연성 실시간 시스템에서 컴퓨터는 보통 실제 장비에 직접 접속(interface)되어 그 장비의 작동을 감시 또는 제어를 전담하게 된다. 즉 대형 공학 시스템내에서 정보 처리 요소로서의 컴퓨터의 역할은 매우 중요하다. 이러한 응용들이 임베디드(embeded) 컴퓨터 시스템으로 알려지게 된 것은 이런 이유 때문이다.



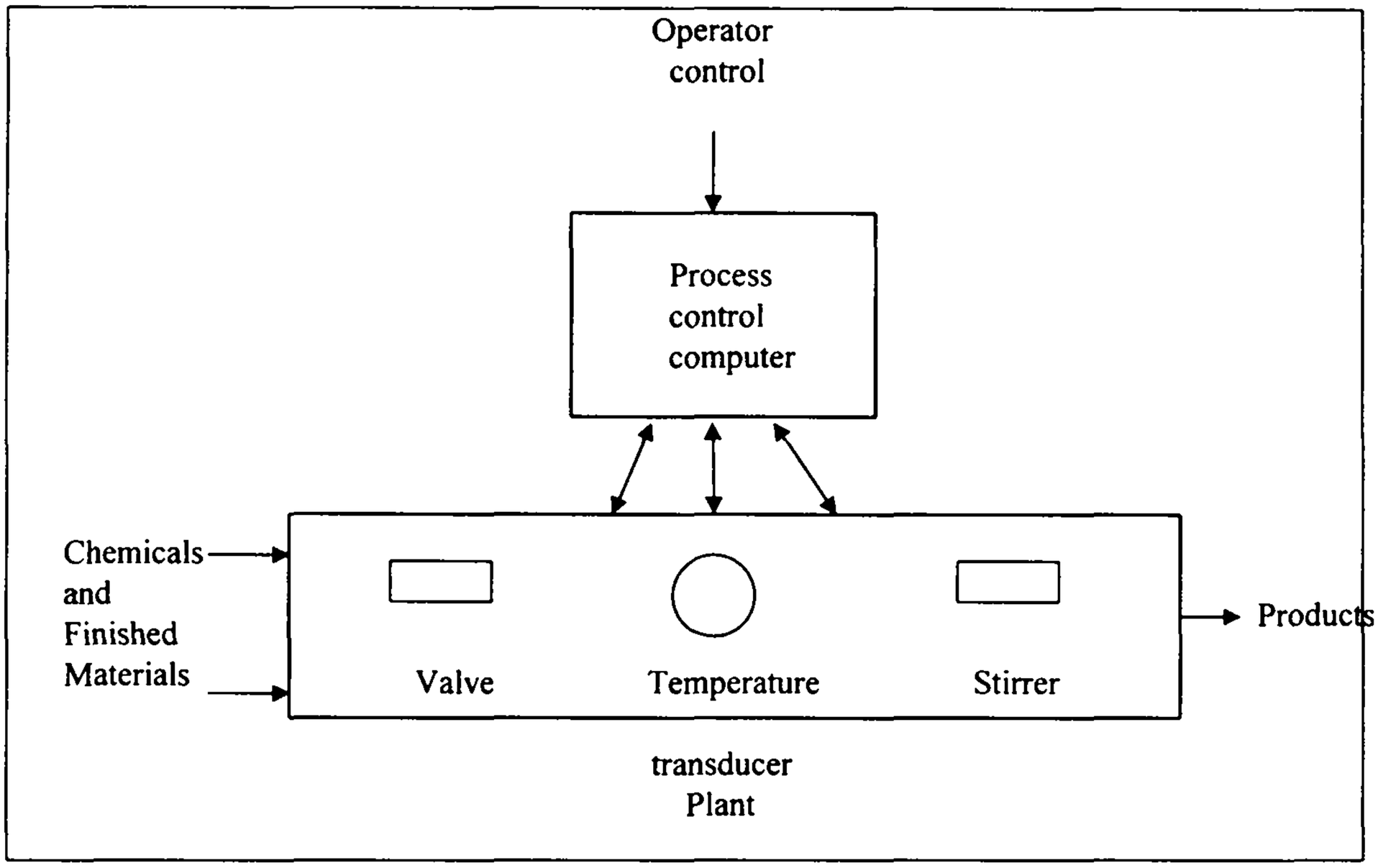
[그림 2-1] 유체 제어 시스템

나. 실시간 시스템의 응용

(1). 프로세스 제어

대형 공학 시스템의 한 요소로서 컴퓨터가 최초로 사용된 것은 1960 년대초에 프로세스 제어 공업에서 시작되었지만, 최근에는 마이크로 프로세서의 사용이 일반화 되어 있다. 컴퓨터가 단일 행동을 하는 경우의 간단한 예가 [그림 2-1]에 나타나 있다. 여기서 컴퓨터는 밸브를 제어하여 파이프내의 유체를 일정하게 흐르도록 하는 것이다. 흐름의 증가를 감지하면 컴퓨터는 밸브의 각도를 변경함으로써 반응한다. 이 경우 파이프의 입력단에서 장비가 과부하되지 않게 하려면 이 반응은 유한 기간내에 발생되어야 한다. 이때, 실제 응답은 새로운 밸브 각도를 계산하기 위하여 꽤 복잡한 계산을 요구한다.

[그림 2-2]는 완전한 프로세스 제어 환경에 임베드된 실시간 컴퓨터의 역할을 보여준다. 컴퓨터는 센서(sensor)와 액츄에이터(actuator)를 사용하여 장비와



[그림 2-2] 프로세스 제어 시스템

상호작용한다. 밸브는 액츄에이터의 예이고 온도 및 압력 변환기(transducer)는 센서의 예이다. (변환기는 측정된 실제 양에 비례한 전기 신호를 생성하는 장치이다.) 이 때, 컴퓨터는 적절한 시간에 올바르게 플랜트가 작동이 되도록 보장하기 위하여 센서와 액츄에이터의 작동을 제어한다. 필요하다면 제어 프로세스와 컴퓨터간에 디지털(digital) 또는 디지털 투 애널로그(Digital to Analog)와 같은 변환기 등을 삽입하여야 있다.

(2) 제조 시스템

최근 들어 제조업에서의 컴퓨터의 이용은 생산 비용을 절감하고 생산성을 향상시키기 위하여 필수적인 것이 되었다. 컴퓨터는 설계에서 제작에 이르기까지 전체 제조 공정의 통합을 가능하게 하였다. 임베디드 시스템이 가장 잘

설명되는 분야가 생산 통제 분야이다. 실제 시스템은 각종 기계 장비로 구성되는데 이들 모든 것이 컴퓨터에 의해 제어되고 조정된다.

(3) 명령 통제

명령 통제(communication, command and control)가 군대 용어이지만 이와 비슷한 특성을 가지는 여러가지 다양한 응용 분야가 존재한다. 예를 들면 항공기 좌석 예약, 항공 관제, 원격 은행 구좌 관리 등이 있다. 이러한 시스템들은 복잡한 정책, 정보 수집 장비 및 행정 절차들로 이루어 지며, 이를 바탕으로 의사 결정을 가능하게 한다. 의사 결정을 구현하기 위하여 필요한 정보 수집 장비들은 지리적으로 널리 분포되는 경우가 많다.

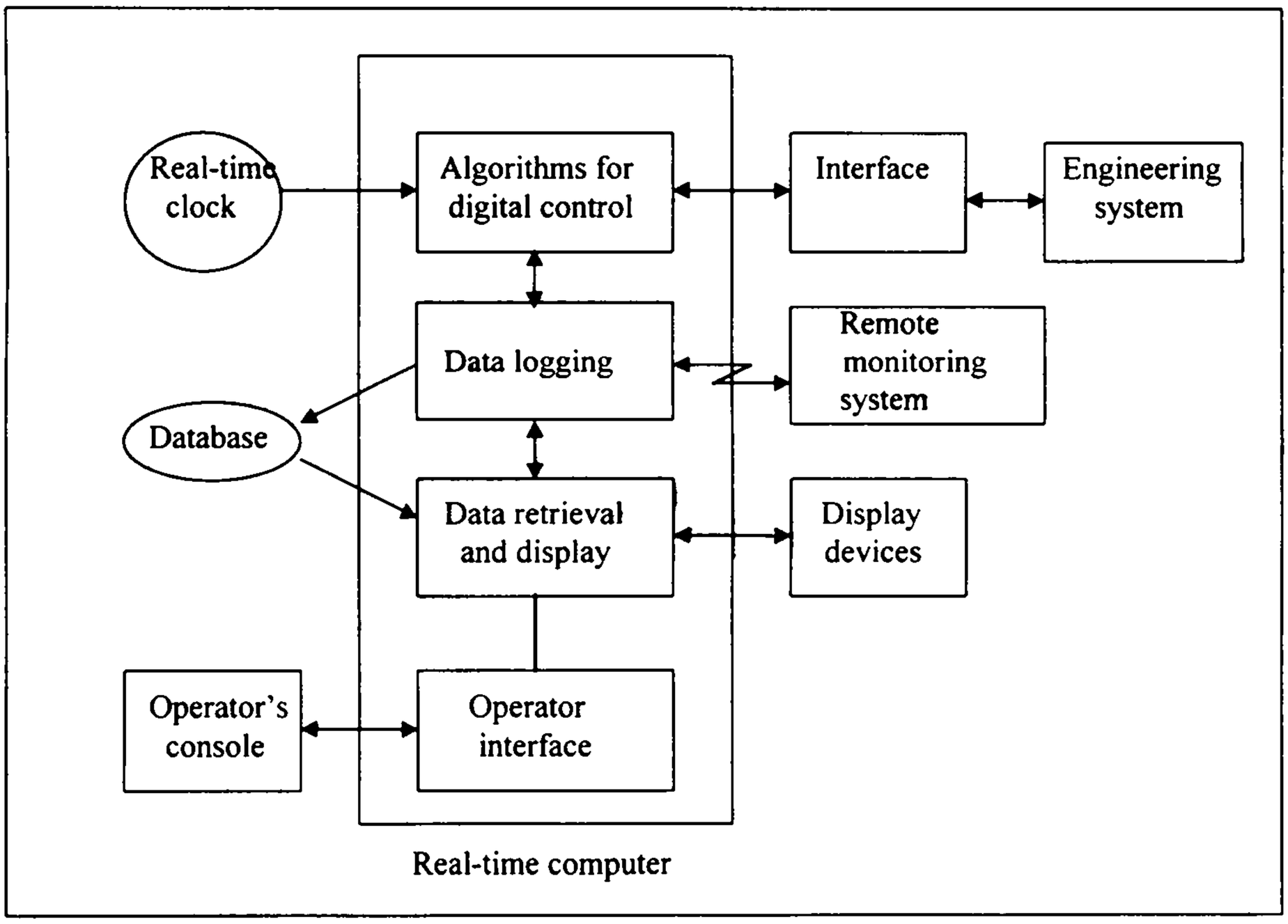
(4) 일반적인 임베디드 컴퓨터

지금까지 보아온 예들은 컴퓨터가 실세계의 물리적인 장비에 직접 접속되어 있는 경우들이다. 이런 실세계의 장비들을 제어하기 위해 컴퓨터는 일정 간격으로 측정 장비에서 데이타를 샘플링할 필요가 있다. 따라서 실시간 클럭이 요구된다. 보통 수작업에 의한 간여를 허용하기 위하여 조작자의 콘솔도 있다. 조작자는 그래픽을 포함한 여러가지 유형의 디스플레이에 의해 시스템의 상태를 지속적으로 감지한다.

시스템 상태 변화의 기록들은 정보 베이스(Information Base)에 저장되어 조작자가 조회할 수 있고, 시스템 고장 시 사후 조사용으로 사용되며, 행정 처리를 위한 정보를 제공한다. 실제로 이 정보는 시스템의 일일 수행에 대한 의사 결정을 지원하기 위하여 사용하는 추세가 증가하고 있다. 예를 들면 화학

및 공정 산업에서 공장 감시는 단순히 생산량을 극대화 하는 것보다는 경제적 이익을 극대화 하는데 필수적이다. 한 공장에서의 생산에 관련되는 결정은 원격지의 다른 공장에는 심각한 영향을 줄 수 있다. 예를 들어 한 공정의 제품이 다른 공정에서 원자재로 사용되는 경우이다.

전형적인 임베디드 시스템은 따라서 [그림 2-3]과 같이 나타난다. 시스템의 운영을 제어하는 소프트웨어는 환경의 물리적 특성을 반영하는 모듈로 작성될 수 있다. 일반적으로 장비를 물리적으로 제어하는데 필요한 알고리즘을 포함한 모듈이 있고 시스템의 상태 변화를 기록하는 것을 책임지고 있는 모듈이 있으며 이러한 변화를 조회하고 디스플레이 하는 모듈, 조작자와 상호작용하는 모듈들이 있다.



[그림 2-3] 전형적인 임베디드 컴퓨터 시스템

다. 실시간 시스템의 특성

(1) 규모 및 복잡도

소프트웨어를 개발하는데 관련된 대부분의 문제는 그 규모 및 복잡도에 관련된다. 소규모 프로그램을 작성할 경우, 한 사람이 설계, 코딩, 유지보수하더라도 별 문제가 없고 오히려 바람직한 경우가 많다.

불행히도 모든 소프트웨어가 이런 가장 바람직한 소규모라는 특성을 가지고 있지는 않다. Lehman 과 Belady(1985)는 대규모 시스템이 단순히 명령어의 수(코드의 라인 수 또는 프로그램 모듈의 수) 에 의존하지는 않는다고 보았다. 오히려 그들은 대규모를 다양성, 대규모의 정도를 다양성의 양에 연관 시켰다.

여기에서 다양성은 실세계에서의 필요성과 활동들이며 프로그램에서의 이들의 반영이라고 볼 수 있다. 그러나 실세계는 지속적으로 변화하고 진화하며 사회의 필요성과 활동도 역시 진화한다. 따라서 이와 같이 모든 복잡한 시스템을 포함한 대규모 시스템은 계속적으로 진화해야 한다는 것인 Lehman 과 Belady 의 주장이다.

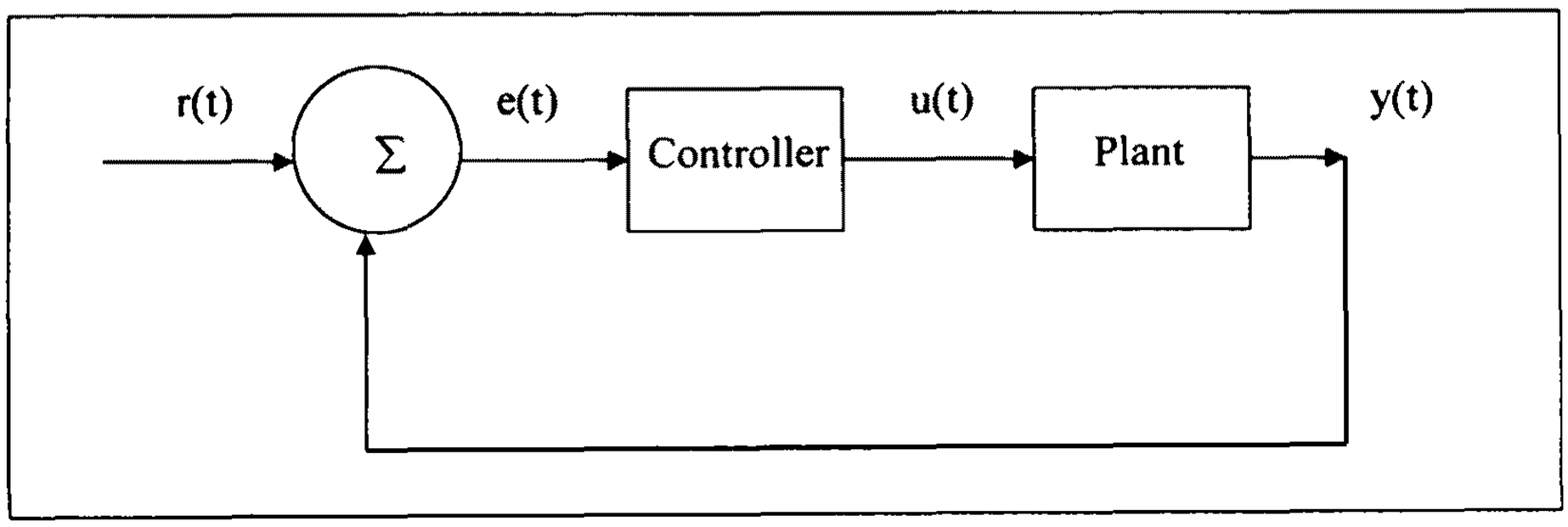
임베디드 시스템은 실세계의 사건에 반응하여 동작하여야 한다. 이러한 사건들에 관련된 다양성은 프로그램이 대규모의 원치않는 특성을 보여주는 경향에 대해서도 대비할 수 있어야 한다. 실시간 시스템은 실세계의 지속적인 변화 사항에 대해 그 생명주기 동안에 계속적인 유지보수와 확장을 할 수 있어야 한다.

실시간 소프트웨어가 대부분 복잡하다. 그러나 실시간 언어 및 실시간 시스템 개발 환경을 이용하면 복잡한 시스템을 효과적으로 관리가 가능한 더 작은 요소로 분리할 수 있게 하여 준다.

(2) 실수의 조작

많은 실시간 시스템은 여러가지 공학적 활동의 제어에 사용된다. [그림 2-4]는 간단한 제어 시스템을 보여준다. 제어되는 객체인 플랜트는 시간에 따라 변하는 y(t)라는 출력변수 벡터를 가진다. 이 출력은 참조 신호 r(t)와 비교되어 오차 신호 e(t)를 발생한다. 제어기는 이 오차 벡터를 참조하여 입력변수를 변화시킨 후 플랜트에 u(t)를 내보낸다. 이 때, 매우 간단한 시스템에서는 제어기가 연속신호상에서 작동하는 아날로그 장치가 될 수 있다.

다음 [그림 2-5]는 피드백(feedback) 제어기를 나타낸다. 이것은 가장 일반적인 형태이나 피드포워드(feedforward) 제어기도 사용될 수 있다. 이 제어기는 입력 변수를 어떻게 변화시켜야 원하는 출력벡터를 발생시킬 수 있는지를 계산하기 위한 수학적 모델을 가져야 한다. 이러한 모델을 찾는 것은 제어공학의 관심 분야이다.

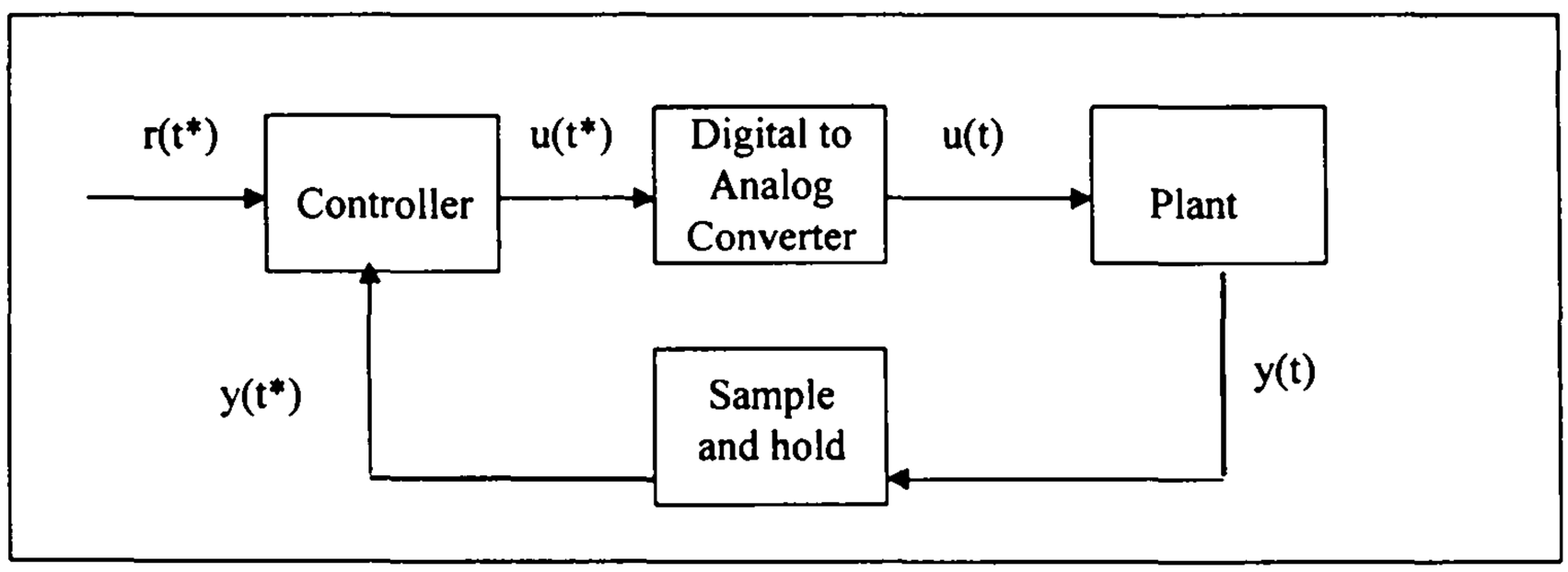


[그림 2-4] 간단한 제어기

많은 경우 플랜트는 1차 미분 방정식으로 표현되어 플랜트의 내부 상태와

입력 변수와 시스템의 출력간의 관계를 모형화한다. 플랜트의 출력을 변경하는 것은 이러한 방정식을 풀어서 요구하는 입력치를 주는 것이다. 많은 시스템들은 그러한 변경이 즉시 일어나지 않도록 하기 위한 관성(inertia)특성을 가지고 있다. 정해진 시간 내에 새로운 설정값으로 변경하여야 하는 실시간 시스템의 요구 조건은 수학적 모델 또는 실제 시스템이 필요로 하는 조작 내용을 훨씬 복잡하게 만든다. 또한 선형 일차 방정식은 실제 시스템의 특성을 근사치로 표현하였기 때문에 실시스템의 복잡성은 더욱 증가한다.

이런 어려움 때문에 모델의 복잡도와 각각의 입출력의 수 그리고 대부분의 제어기들은 컴퓨터로서 구현된다. 컴퓨터로 구현하게 되면 시스템에 디지털 요소를 도입하게 되고, 제어값을 이산치로 나타내게 되어 제어 사이클의 특성이 변화한다. [그림 2-5]는 [그림 2-4]를 디지털 모델로 나타낸 것이다. * 로 표시한 항목은 이산치이다. 아날로그 투 디지털 변환기에 의해 수행되는 샘플링 및 보존 조작이 그것인데 이것은 모두 컴퓨터의 제어하에 있다.



[그림 2-5] 간단한 컴퓨터화된 제어기

(3) 신뢰성과 안전성

사회가 그 기능을 컴퓨터에게 양도하면 할수록 컴퓨터는 더욱 실패없이 작동 되어야 한다. 실시간 시스템의 장애로 발생할 수 있는 재해는 그 피해가

엄청나기 때문에 컴퓨터 하드웨어와 소프트웨어는 신뢰성 있고 안전하게 작동되어야 한다. 군사적 응용에서 그 중요성은 더욱 커다고 할 수 있다. 더욱이 조작자와 상호작용이 요구될 때 사람에 의한 오류의 가능성과 영향을 최소화하기 위해 인터페이스의 설계에도 주의를 기울여야 한다.

실시간 시스템의 규모와 복잡도만으로도 신뢰성 문제를 악화시킬 수 있다. 즉, 응용에 내재된 예상되는 난이도가 증가하면 소프트웨어 설계시에 발생하는 오류도 증가할 가능성이 커진다.

(4) 개별 시스템 요소의 동시성 제어

임베디드 시스템은 여러 컴퓨터(또는 컴퓨터 프로그램)들이 동시에 상호작용하도록 만들어지는 경향이 있다. 전형적인 임베디드 컴퓨터의 예에서 프로그램은 공학 시스템(로봇, 컨베이어 벨트, 센서, 엑츄에이터등과 같은 많은 병행 활동들로 구성됨)과 컴퓨터 디스플레이 장치, 오퍼레이터 콘솔, 데이터 베이스 및 실시간 클럭등과 상호작용하도록 만들어져 있다. 보통 이러한 활동들이 컴퓨터상에서는 순서적으로 수행되고 있으나, 매우 빠른 속도 때문에 거의 동시에 작용하는 것처럼 보여준다. 그러나 어떤 임베디드 시스템에서는 그렇지 못하다. 예를 들면 데이터가 지리적으로 다양하게 분산된 지역에서 수집되고 처리되는 경우 한 컴퓨터에 의해 처리되기는 매우 어렵다. 이 경우에는 분산 및 멀티프로세서 임베디드 시스템을 고려할 필요가 있다.

동시성을 고려하여야 하는 시스템의 소프트웨어를 개발할 때, 프로그램의 구조내에서 동시성을 어떻게 표현하는가가 중요한 문제가 된다.

(5) 실시간 장치

모든 임베디드 시스템에서 시스템의 응답시간은 매우 중요하다. 그러나, 모든 조건하에서 적절한 시간에 적절한 출력이 생성되는 시스템을 설계하고 구현하는 것은 매우 어렵다. 또 모든 가용한 컴퓨팅 자원을 완전히 사용하는 것은 종종 불가능하다. 이러한 이유 때문에 실시간 시스템은 보통 어느정도 여분의 용량을 가진 프로세서로써 구성되어 시스템의 안전을 보장하는 경우가 많다. 적절한 처리 용량이 주어졌을 경우 프로그래밍 언어와 실행 지원 시스템은 다음과 같은 일을 할 수 있도록 지원하여야 한다.

- 활동이 수행되는 시간의 명세
- 활동이 완료되는 시간의 명세
- 모든 시간적 요구조건이 만족될 수 없는 상황에 대한 응답
- 시간적 요구조건이 동적으로 변하는(모드 변경) 상황에 대한 응답

실시간 제어 장치는 프로그램을 시간 그 자체에 동기화 시킬 수 있어야 한다. 직접 디지털 제어 알고리즘을 써서 특정 시간(예를 들면 2pm, 3pm 등)에, 규칙적인 간격(예를 들면 매 5 초마다)으로 센서로 부터 읽어오는 것을 샘플링한 후 그 결과를 바탕으로 일을 수행할 필요가 있다. 예를 들면 발전소에서는 가정 수요의 피크에 대비하여 매주 월요일 5pm 에서 금요일 까지 가정 수요자에 대한 전기의 공급을 증가시킬 필요가 있다.

모드 변경의 예는 항공 제어 시스템에서 발견된다. 비행기가 압력이 낮아지면 이 응급상황을 처리하기 위해 주어진 모든 가용 컴퓨팅 자원을 즉시 가동

시켜야 한다.

(6) 하드웨어 인터페이스와의 상호작용

임베디드 시스템에서는 컴퓨터 요소들이 외부 세계와 상호작용을 한다. 컴퓨터는 다양한 실세계 장치들의 동작을 알아내는 센서와 제어 액츄에이터를 감시해야 한다. 이러한 장치들은 입출력 레지스터(register)들을 통해 컴퓨터와 접속되고 그들의 운영 요구 조건들은 장비와 컴퓨터에 의해 결정된다. 장비들은 프로세서에게 어떤 동작을 수행시키거나 오류조건을 생성시키기 위하여 인터럽트(interrupt)를 발생시킬 수도 있다.

과거에는 장비에 대한 접속이 운영체제의 제어하에 있거나 응용 프로그래머가 어셈블리(assembly) 언어를 써서 레지스터와 인터럽트를 제어, 처리하는데 의존했다. 현재는 장비의 다양성과 그들과 관련된 상호작용의 시간 임계 특성 때문에 그것들에 대한 제어는 운영체제의 기능을 통하지 않고 직접 하는 경우가 많다. 더욱 하위 수준 프로그래밍 기술을 사용하여 신뢰성을 확인하기는 매우 어렵다.

(7) 효율적인 구현

실시간 시스템을 효율적으로 구현하는 것은 그 시간 임계 특성을 고려해야 하기 때문에 다른 시스템에 비해 더욱 중요하다. 상위 수준 언어를 사용하면 프로그래머는 구현의 명세로부터 문제를 쉽게 해결하도록 집중할 수 있다. 그러한 임베디드 컴퓨터 시스템은 프로그램이 하드웨어에 종속되어 있기 때문에 상위 수준의 언어를 사용지 못하는 경우가 많다. 프로그래머는 사용하는

프로그래밍 언어의 특성에 관심을 기울여 적절한 것을 선택할 수 있어야 한다. 예를 들면, 어떤 입력의 응답이 microsecond 내로 요구된다면 실행시 millisecond 가 소요되는 언어 특성을 사용할 수는 없는 것이다.

2. 시뮬레이션

가. 시뮬레이션의 정의

넓은 의미로서 컴퓨터 시뮬레이션은 실시스템의 수학적인 논리 모델을 설계하고 컴퓨터 상에서 이 모델을 시험하는 과정이다. 이와 같이 시뮬레이션은 이 모델과 관련된 적절한 실험의 설계 및 구현 뿐만 아니라 모델 구축 과정을 포함한다. 이러한 실험 또는 시뮬레이션은 실제 시스템에 대하여 설계, 운영 절차 해석 및 성능 평가에 관한 다음의 추론을 가능하게 한다.

- 시스템을 구축하지 않고서도 이것이 제안된 시스템인지를 판단한다.
- 시스템을 방해하지 않고서도 시스템상에서 실험하는데 있어서 운영시스템이 비용이 많이 드는지 또는 안전하지 못한지를 판단한다.
- 시스템을 파괴하지 않고서도 실험 대상의 강도의 한계를 판단한다.

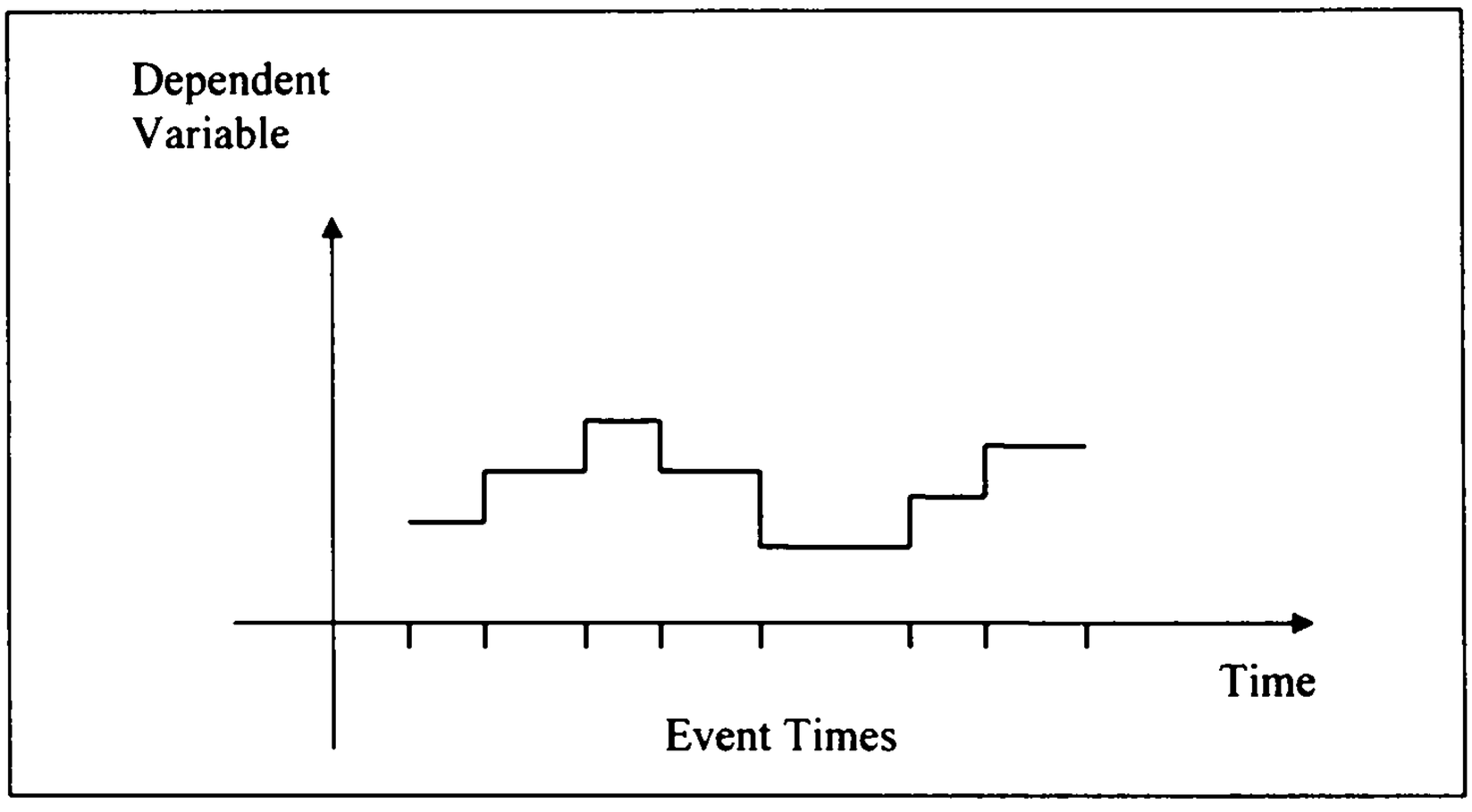
시뮬레이션 모델링은 컴퓨터 시스템에서 쓰이는 용어로 시스템을 표현할 수 있다고 가정한다. 이점에 관해서 중요한 것은 시스템 상태 기술의 개념이다. 시스템이 변수의 집합 즉 시스템의 고유한 상태 또는 조건을 나타내는 변수값의 조합으로 특성화 될 수 있다면 변수값의 조작은 시스템의 움직임을 하나의 상태에서 다른 상태로 시뮬레이션 한다. 시뮬레이션 실험은 모델내에

설계된 잘 정의된 운영 규칙을 따라서 상태에서 상태로 움직이는 모델의 동적인 행동을 관찰하는 것과 관련된다.

시스템 상태내의 변화는 시간에 따라 연속적으로 발생할 수도 있고 이산적인 시간 순간에 발생할 수도 있다. 이산적인 순간들은 모델 입력의 특성에 따라 결정적으로 성립될 수도 있고 추계적(stochastic)으로 성립될 수도 있다. 이산적 변화 모델과 연속적 변화 모델의 동적인 행동을 기술하는 절차는 달라도 시간에 따라 변하는 시스템의 상태를 표현하여 시스템을 시뮬레이션하는 기본적 개념은 같다.

나. 시뮬레이션 모델의 분류

시스템의 모델은 이산적 변화와 연속적 변화로 구분될 수 있다. 이러한 용어는 모델을 표현하는 것이며 실제 시스템을 표현하는 것은 아니다. 실제로 동일한 시스템을 이산적으로 변화하는 모델과 연속적으로 변화하는 모델로 모두 모델링하는 것이 가능하다. 대부분의 시뮬레이션에서 시간은 중요한 독립변수이다. 시뮬레이션에 포함되는 다른 변수들은 시간의 함수이며 종속변수이다. 시뮬레이션을 수식할 때 "이산적"과 "연속적"이란 형용사는 종속변수의 행동을 지칭한다.



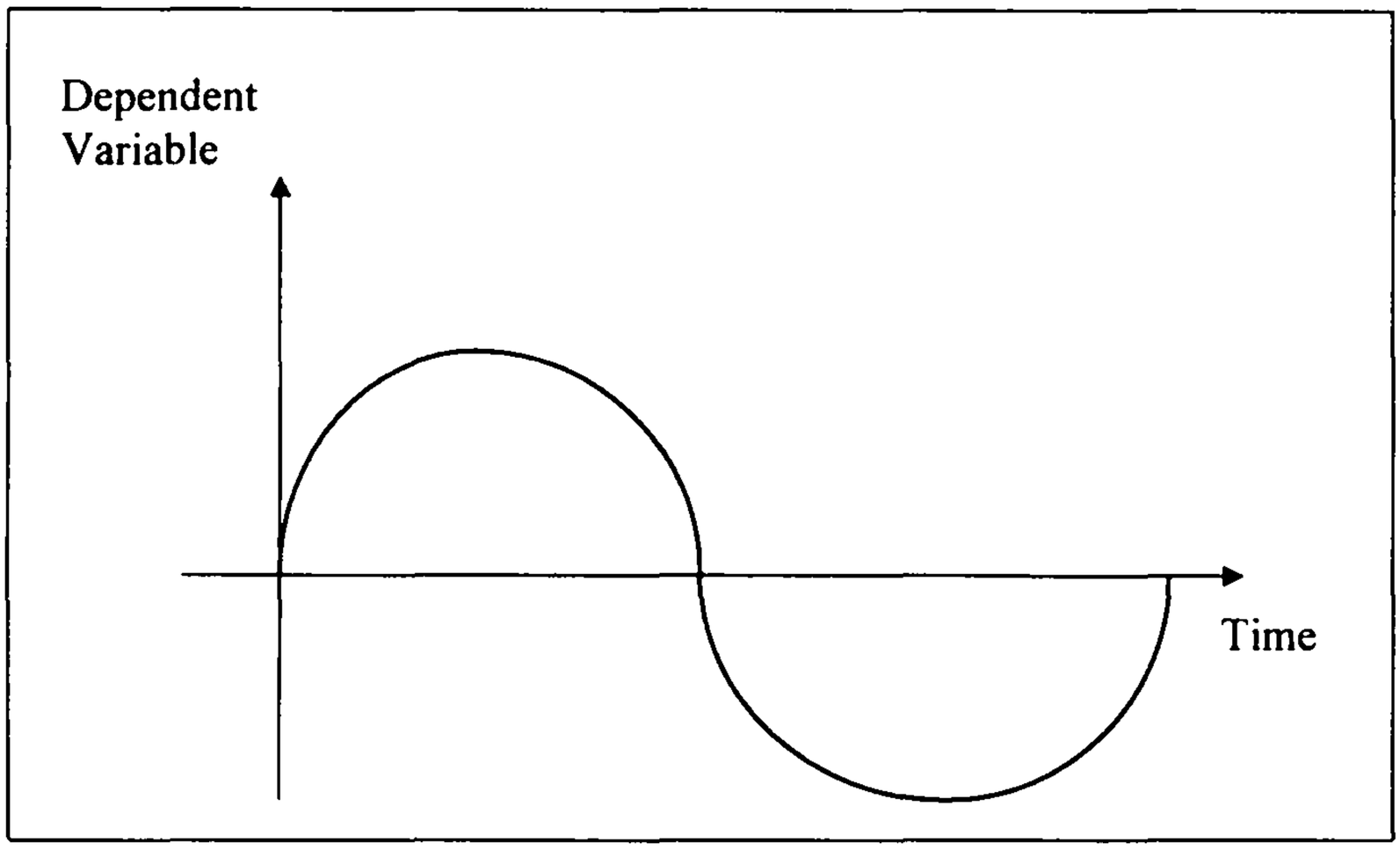
[그림 2-6] 이산 사건 시뮬레이터로 부터의 응답 측정

이산 시뮬레이션은 종속변수가 사건 시각(event time)이라고 지칭되는 시뮬레이션 시간내에 규정된 시점에서 이산적으로 변화할 때 발생한다. 시간변수는 이러한 모델내에서 연속적이거나 이산적일 수 있는데 이것은 종속변수내의 이산적인 변화가 시간내의 어떤 점에서도 발생할 수 있는지 또는 단지 규정된 점에서만 발생할 수 있는지에 좌우된다.

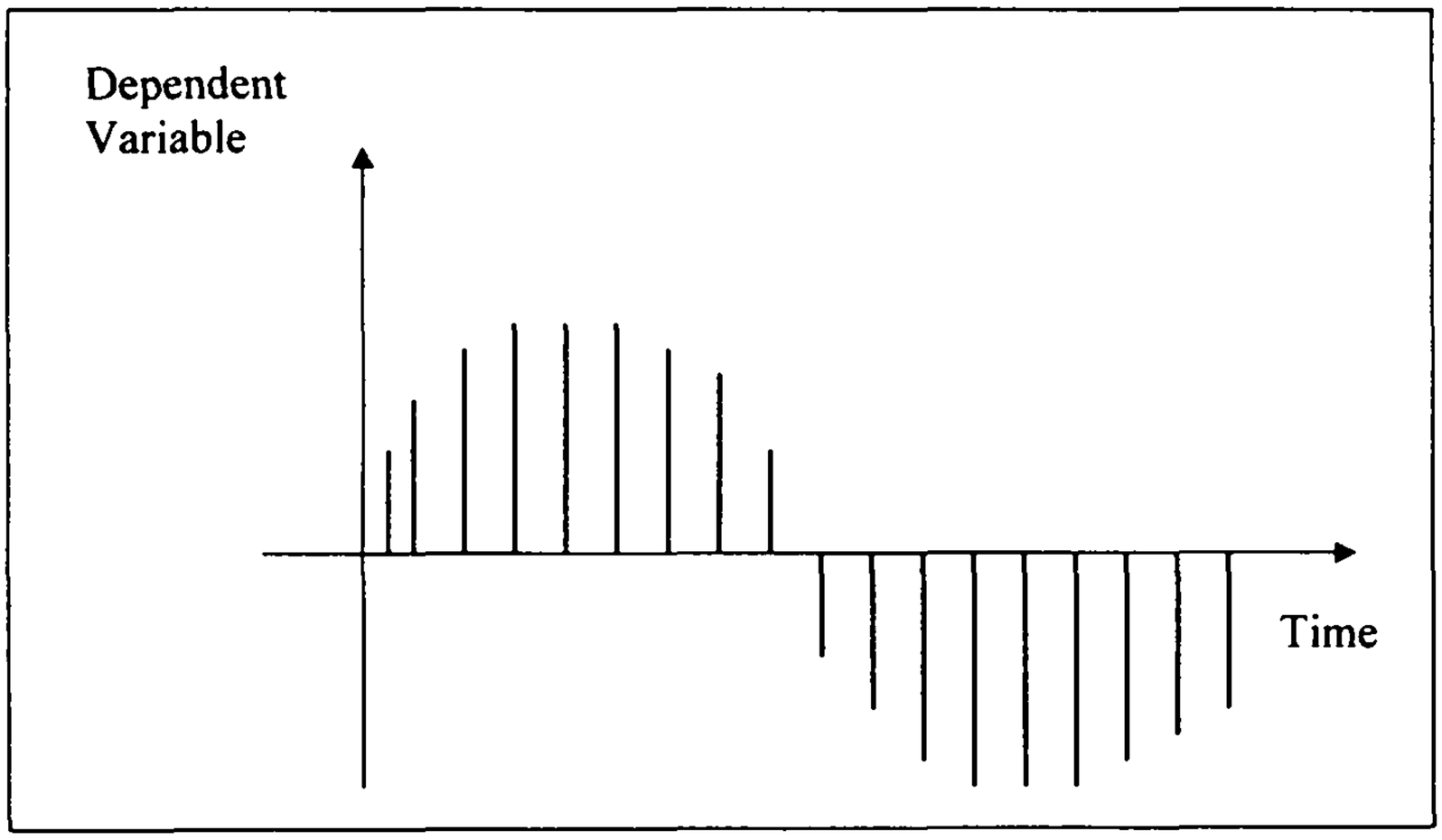
은행원 문제는 이산 시뮬레이션의 한 예다. 이 예의 종속변수는 은행원의 상태와 대기 고객의 수가 될 수 있다. 사건 시간은 고객이 시스템에 도착하는 시각과 은행원에 의해 서비스가 종료됨에 따라 시스템을 떠나는 시각이다. 일반적으로 이산 모델의 종속변수는 사건 시간 사이에는 변하지 않는다. 이산 시뮬레이션의 종속변수에 대한 응답의 예가 [그림 2-6]에 표시되어 있다.

연속 시뮬레이션에서는 모델의 종속변수가 시뮬레이션 시간에 따라 연속적으로 변한다. 연속 모델은 시간에 따라 연속적일 수도 있고 이산적일 수도 있

는데 종속변수의 값이 시뮬레이션 시간의 어떠한 점에서도 가능한지 또는 시뮬레이션 시간내의 단지 규정된 점에서만 가능한지에 의해 좌우된다. 연속 시뮬레이션에 대한 응답 측정의 예가 [그림 2-7]과 [그림 2-8]에 표시되어 있다.



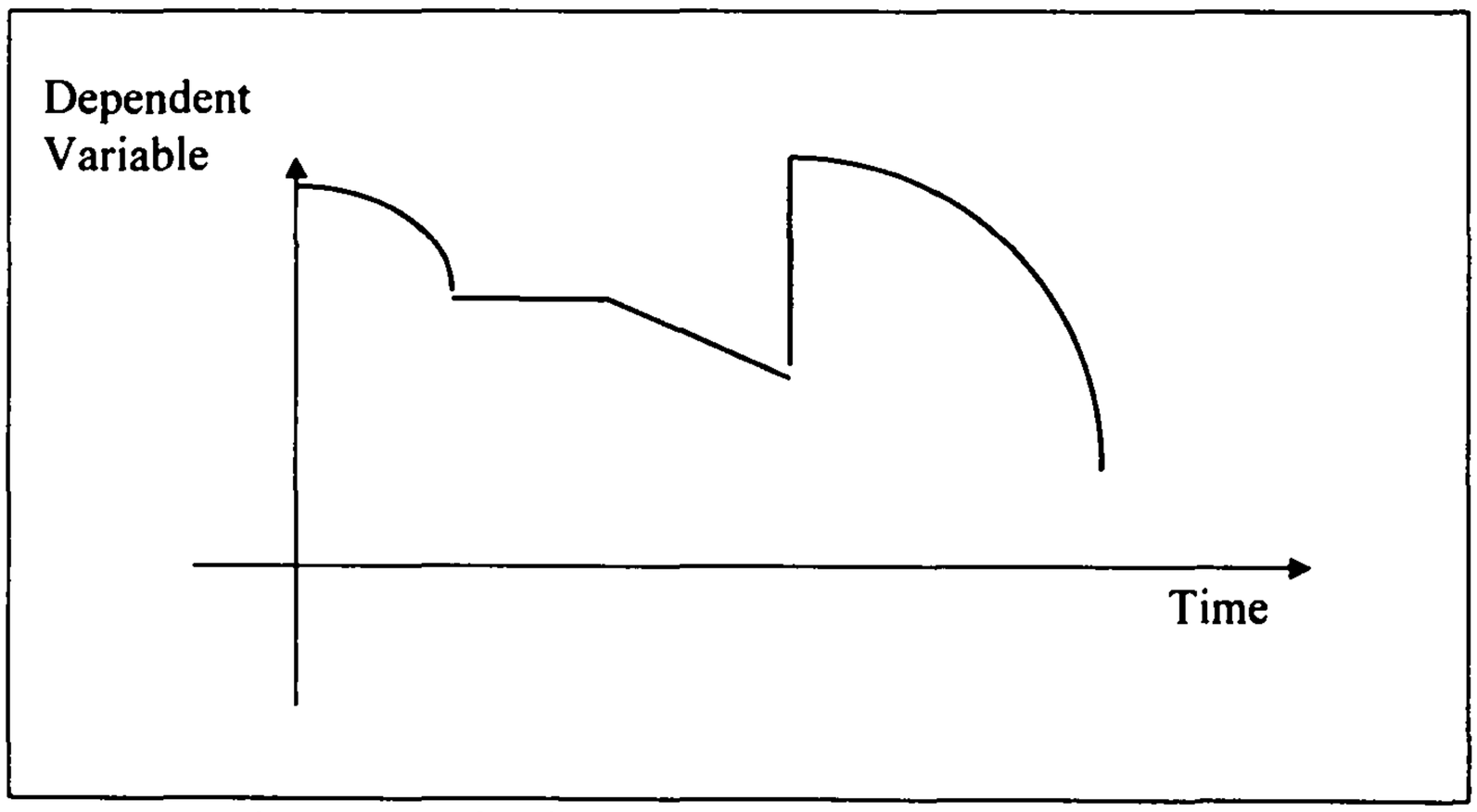
[그림 2-7] 연속 시뮬레이터로 부터의 응답 측정치



[그림 2-8] 이산 시간을 사용한 연속 시뮬레이터로 부터의 응답 측정치

화학 공정의 반응 물질 농축 또는 우주선의 위치와 속도등의 모델링은 연속적인 표현이 적절한 상황의 예이다. 그러나 어떤 경우에 있어서는 시스템내의 개체(entity)들을 각각의 개체가 아니라 집합적인 개체로 인식하여 연속적인 표현을 써서 이산 시스템을 모델링하는 것이 또한 유용할 때가 있다. 예를 들면 실제로 군집의 변화는 이산적이데도 불구하고 어떤 종류의 생물 군집을 모델링할 때 연속적인 표현을 써서 하는 것이 유리할 수 있다.

혼합 시뮬레이션에서는 모델의 종속변수가 이산적으로, 연속적으로 또는 이산적인 도약이 겹쳐지면서 연속적으로 변화한다. 시간 변수는 연속적이거나 이산적이다. 혼합 시뮬레이션의 가장 중요한 측면은 이산적으로 변화하는 변수와 연속적으로 변화하는 변수간의 상호작용으로 부터 발생한다. 예를 들면 화학 공정의 반응 물질의 농축 수준이 규정된 수준에 도달할 때 공정은 종료될 수 있다. 혼합 시뮬레이션 언어는 이러한 조건들의 발생을 탐지하고 그것의 결과를 모델링하는 규정들을 포함하여야 한다. 혼합 시뮬레이션 모델의 응답을 나타내는 예가 [그림 2-9]에 표시된다.



[그림 2-9] 조합 시뮬레이터로 부터의 응답 측정

다. 분산 시뮬레이션

일반적으로 가장 많이 사용하는 이산 사건 시뮬레이션(Discrete Event Simulation)의 기본적인 작동 방법은 다음과 같다. 시뮬레이션 클럭과 사건 리스트(Event List)는 상호 작용하여 어떠한 사건이 다음에 처리될 것인지를 결정한다. 시스템 클럭이 이 사건의 시각까지 진행하면 컴퓨터는 사건 논리(Event Logic)를 실행하여 상태 변수를 수정하고 큐와 사건들에 대한 리스트를 조작한다. 그리고 임의의 수와 변수를 발생하고 통계치를 수집한다. 이러한 방식은 사건들이 발생하는 시뮬레이션 시간에 따라 순차적으로 하나의 컴퓨터에서 수행된다.

최근의 컴퓨터 기술은 각각의 컴퓨터 또는 프로세서들을 서로 연결하여 병렬 또는 분산 컴퓨터 환경을 가능하게 하였다. 예를 들면 비교적 값이 싼 여러개의 미니 컴퓨터 또는 마이크로 컴퓨터는 네트워크로 연결될 수 있고 더 큰 컴퓨터는 다수의 프로세서들을 묶어서 자신의 일을 할 뿐 만아니라 서로간의 통신도 가능하게 한다. 이러한 환경에서는 컴퓨팅 과업의 각기 다른 부분들을 각각의 프로세서에 분산하여 동시에 또는 병렬로 실행하여 과업을 완료하여 전체 수행 시간을 줄일 수 있다. 이러한 능력은 컴퓨팅 과업의 특성과 가용한 하드웨어와 소프트웨어에 좌우된다.

동적 시뮬레이션을 분할하여 여러 작업을 다수의 프로세서들에 분산하는 방법은 여러가지가 있다. 아마도 가장 직접적인 접근방법은 각각의 지원 기능들(임의의 수 발생, 임의의 변수 발생, 사건 리스트 처리, 리스트와 큐의 조작, 통계치 수집등)을 서로 다른 프로세서들에 할당하는 것이다. 시뮬레이션의 논리적인 실행은 아직까지 순차적이며 다만 시뮬레이션 주프로그램은 지원 기

능들을 다른 프로세서들에게 보내고 작업을 같이 한다.

분리된 프로세서들에 시뮬레이션을 분산하는 또다른 방법은 모델 그 자체를 분리하여 여러개의 서브모델(submodel)로 나누고 이것들을 다른 프로세서에게 할당하여 수행시키는 것이다. 예를 들면 제조 설비는 대기 스테이션들의 상호 연결된 네트워크로 모델링할 수 있고 각각은 각기 다른 행동 유형을 보여준다. 각각의 서브모델들은 다른 프로세서들에 할당되고 각 프로세서는 모델의 한 부분을 시뮬레이션한다. 프로세서들은 서브모델간의 적절한 논리적 관계를 유지하기 위하여 필요할 때 마다 서로 통신해야 한다. 제조 시스템의 예를 들면, 가공 제품이 한 대기 스테이션을 떠나 다른 스테이션(다른 프로세서에서 시뮬레이션되고 있음)으로 갈때 발생된다. 활동의 시간 순서를 올바르게 유지하기 위해서 주의를 기울여야 하는데 즉, 모델의 전체 활동을 올바르게 나타내기 위해서는 다른 프로세서상의 서브모델의 운영을 동기화 (synchronize) 하여야 한다. 이런 유형의 분산 시뮬레이션의 중요한 장점은 (전체적인) 시뮬레이션 클럭과 (완전한) 사건 리스트가 필요치 않다는 것이다. 왜냐하면 고전적 시뮬레이션 모델링에서 사건 리스트 처리는 프로그램을 수행하는데 많은 시간을 필요로 하기 때문이다. 사건 리스트를 제거하는 것은 매력적인 착상이다. 클럭과 사건 리스트를 대신하는 것은 프로세서간에 메시지를 전달하는 시스템이며 여기서 각 메시지는 “시각 스탬프”를 전달한다. 그러나 한가지 단점은 시뮬레이션상에서 발생할 수 있는 교착상태(deadlock)인데 (두개의 프로세서가 진행하기 전에 각각 다른 프로세서로 부터 메시지를 기다림) 이것은 시뮬레이션 되는 실제 시스템에서는 발생할 수 없는 것이다. 따라서 시뮬레이션을 정지 시키는 교착상태를 탐지하고 해제시키거나 사전에 피

하는 방법이 있어야 한다. 분산 시뮬레이션의 방법은 Chandy 와 Misra 에 의해 개발되었다.

병렬 프로세서에 걸쳐서 서브모델을 분산시키는 데 관련된 또다른 개념은 Time-Warp 방법에 의해 구현되는 가상 시간(Virtual Time)으로 알려진 방법이다. 위에서 본 바와 같이 각각의 프로세서는 모델의 자신의 부분을 시간에 따라 시뮬레이션하나 다른 속도로 움직일 수 있는 다른 프로세서로부터 메시지를 기다리지는 않는다. 이러한 기다림은 메시지 전달 접근 방법에서는 필요한 것이다. 만약 특정 프로세서상에서 시뮬레이션되는 서브모델이 과거에 받았어야 될 메시지를 받는다면 (따라서 이 시점으로 부터 활동에 영향을 줄 수 있다면) 받는 서브모델에 롤백(roll back)이 일어나고 이것의 시간은 받는 메시지의 시간을 되돌린다. 예를 들면 서브모델 B 가 시각 87 까지 시뮬레이션되고 서브모델 A 로 부터 온 메시지가 시각 61 에 받은 것일때 서브모델 B 의 클럭은 61 까지 롤백되고 시각 61 과 87 사이의 서브모델 B 의 시뮬레이션은 취소된다. 왜냐하면 이것은 시각 61 메시지의 내용을 모르고 부정확하게 수행되었을 가능성이 있기때문이다. 이러한 취소된 일의 부분은 다른 서브모델에 메시지를 보낼 수 있으며 이것들은 각각 대응되는 앤티메시지(antimessage)를 보내어 무효화 시킨다. 앤티메시지들은 그들의 목적 서브모델들에 부차적인 롤백을 생성시킨다. 시각 61 과 87 사이에 발생한 일은 불행하게도 손실되며 롤백을 수행함으로써 수반된 추가적인 오버헤드(overhead)를 감수하여야 한다. 그러나 모든 프로세서들은 시간이 감에 따라 올바로 전진하기 전에 메시지를 기다리기 위해 대기하지 않고 롤백 과정을 제외하고는 계속적으로 바쁘게 시뮬레이션을 한다. 추계적(stochastic) 시뮬레이션에서는 롤백이 필요할지가 불확

실하다. 따라서 Time-Warp 장치는 "기회의 게임(Game of Chance)"이라고 불리어 진다. 왜냐하면 추가적인 메모리와 롤백의 처리에 따른 오버헤드 때문에 효율이 저하될 수 있고 반면 롤백의 가능성이 희박하면 모든 프로세서가 계속적으로 전진할 수 있기 때문에 효율이 올라갈 수 있다. 더욱이 Time-Warp 장치를 사용하면 교착상태를 회피할 수 있다.

시뮬레이션에서 분산 처리의 개발 및 평가는 현재 활발한 연구 분야 중의 하나이다. 그러나 이 방법이 얼마나 잘 작동할는지는 가용한 컴퓨터 환경 뿐만 아니라 모델의 구조와 매개변수에 따라 좌우된다. 예를 들면 모델이 단지 약한 관계를 갖는 서브모델로 분리될 수 있다면 시뮬레이션을 분산하는 모델에 기반한 구조는 더욱 정확하게 보여줄 수 있다.

3. 실시간 시뮬레이션

가. 실시간 시뮬레이션의 정의

대부분의 실시간 시스템은 그 규모가 크고 복잡하기 때문에 사용자의 요구사항을 정확하게 규명하기가 어렵고, 개발하면서 당면하는 의사결정 사항들에 대해서 결정을 내리기가 매우 어렵다. 특히 시간적 행위에 대한 예측과 증명을 할 필요가 많은데 이를 해결하는 방법으로 실시간 시뮬레이션 기법이 필요하다.

실시간 시뮬레이션을 수행할 때 다음의 두가지 필수 조건을 만족시켜야 한다.

(1) 실제 응용 환경에서 발생하는 여러 사건들의 상대적 시간성이 정확하게

시뮬레이터에 의하여 모방되어야 한다. 따라서 사건들의 순서가 실제 환경에서나 시뮬레이터상에서 같아야 함은 물론이고 실제 환경에서 일어난 두 사건간의 시간 간격과 상응하는 시뮬레이터 상에서의 시간 간격과의 비율이 어느 두 사건을 고려하더라도 항상 일정하거나 미세한 변동폭을 보여야 한다.

(2) 시뮬레이션 속도는 실제 응용 시스템이나 환경에 진행하는 속도 만큼 빨라야 한다.

따라서 이러한 실시간 시뮬레이션은 응용 환경에 존재하는 여러 구성 요소간의 상호작용의 시간적인 면을 정확하게 보여줄 수 있는 것이다.

Zeigler(1993)는 실시간 시뮬레이션을 다음과 같이 정의 하였다.

"실시간 시뮬레이션은 시뮬레이션의 시간적 basis 가 컴퓨터 시스템의 클럭 타임과 매우 밀접히 연관되는 시뮬레이션이다."

이상적인 시스템에서는 모델에 의해 계획되는 모든 사건들이 계획된 시간과 정확하게 일치되는 시간에 발생한다. 그러나 한정된 처리시간 때문에 실제 사건 시각은 모델에 규정된 값과 달라질 수 있다. 따라서 경성 마감시간(Hard Deadline)을 가진 응용에서는 충분히 빠른 컴퓨팅 플랫폼(platform)이 필요하고, 계획된 시간의 허용된 한계(tolerance)내에서 사건이 발생하도록 하여야 한다.

나. 거시적 시뮬레이션과 미시적 시뮬레이션

복잡한 실시간 시스템을 모델링하고 시뮬레이션 함에 있어 두가지 접근 방법을 취할 수 있는데 그 하나는 대규모 응용 시스템을 적당한 수의 추계변수

로 구성된 거시적(macroscopic) 모델로 나타내어 시뮬레이션하는 것이고 다른 하나는 일단 사용자의 요구 사항과 전체적인 시스템의 주요 기능과 구조가 결정된 후 시스템의 상세 설계 과정에서 발생하는 여러가지 결정사항을 미시적(microscopic)으로 모델링하고 시뮬레이션 하는 것이다. 즉 시스템의 분할, 분할되어 생성된 서브 시스템들의 네트워크를 통한 연결 방법, 과업의 분할 및 스케쥴링등 설계 과정에서 결정되어야 할 많은 사항들이 실시간 성능에 직접적인 영향을 미치는데 미시적 고정밀 시뮬레이션이 이와 같은 결정을 체계적으로 내리며 결정되는 사항이 시스템 성능에 미치는 영향을 측정할 수 있도록 도와준다.

[표 2-1] 시스템 개발주기 및 미시적 시뮬레이션과 거시적 시뮬레이션의 역할

시스템 개발 주기	시뮬레이터의 역할
Requirement Engineering	거시적 시뮬레이터 : 요구사항의 추출과 요구명세의 검증
High-Level I Design	거시적 시뮬레이터 : 설계의 결정사항 지원
High-Level k Design	미시적 시뮬레이터 : 생산된 설계명세를 미시적 시뮬레이션에 의해 수행 가능
Low-Level Design	미시적 시뮬레이터 : 응용환경의 미시적 시뮬레이터을 사용하여 설계명세 검증
Testing (1 단계)	미시적 시뮬레이터 : 응용환경의 미시적 시뮬레이터를 사용하여 시스템 테스트
Testing(2 단계)	실제 환경에서 테스트

시스템 개발 주기에 있어서의 거시적 및 미시적 시뮬레이터의 역할은 [표 2-1]과 같다.

거시적 시뮬레이션은 [표 2-2]에 나타난 바와 같이 모델을 설정하고 시뮬레이터를 구현하는 것이 비교적 용이하고 비용이 적게 들어 시스템 요구사항을 설정하기 위하여 급히 사용할 수 있는 시제품 도구(Rapid Prototyping Tool)로 적합하다고 할 수 있다. 이러한 거시적 시뮬레이션은 시스템 개발 주기에 있어 초기 단계에 활용되며 교통 제어 시스템, 금융 관리 시스템, 에너지 유통 시스템, 정보 통신 시스템 및 재해 관리 시스템등의 설계 단계에서 커다란 역할을 할 수 있다.

[표 2-2] 미시적 시뮬레이션과 거시적 시뮬레이션의 특성 비교

구분	거시적 시뮬레이션	미시적 시뮬레이션
장점	시뮬레이터를 신속 저렴하게 설치할 수 있다.	응용 시스템의 계획, 개발 및 운영 단계 기간중에 상세한 결정사항에 대하여 정밀한 정보를 제공한다.
단점	시스템 개발 및 운영단계 기간의 결정사항에 대한 지원이 약하다.	시뮬레이터의 설치비용이 크다. (각 응용마다 상당한 설계 노력이 요구됨) 양질의 라이브러리를 구축하여 비용을 줄일 수 있다.

미시적 시뮬레이션은 목표로 하는 응용 시스템의 세부적인 구성 및 동작들을 분명히 표현하는 것으로 다음과 같은 성질을 갖는다. 즉 방대한 양의 각종 객체로 구성되며 각 객체의 동작, 특히 시간적 요소를 상세히 표현하여야 한다. 또한 목표로 하는 응용 시스템의 계속적인 확장 및 개조에 따른 모델의 확장 및 개조가 용이하여야 한다. 따라서 미시적 시뮬레이터를 구축하는데는

다음과 같은 난점을 극복하여야 한다.

정확하고 이해하기 쉽고 확장 및 개조가 용이한 미시적 모델을 구축하고 대규모의 고정밀 모델을 병렬 처리 등의 대용량 고성능 컴퓨터 시스템에서 빠른 시간내에 수행되는 프로그램으로의 전환이 쉬워야 한다. 이와같이 구축된 고정밀 미시적 시뮬레이터는 새로 개발하는 대규모 시스템의 최적 설계 선정 또는 기존 실시간 시스템의 최적 운영 방식에 공헌하고 불규칙적인 동작을 하도록 설계된 시스템의 시간별 성능의 정확한 표시에 사용될 수 있다.

다. 실시간 시뮬레이션의 응용

미국, 유럽, 일본등지에 원자력 발전, 우주 항공, 공장 자동화등의 핵심 설비에 실시간 시스템이 구축되어 있으며 이의 개발 및 성능 향상등의 목적으로 실시간 시뮬레이터가 개발 운영되고 있다. 1980 년대에 항공기 시뮬레이터의 개발이 활발하게 이루어 졌고 최근에는 원자력 발전등의 발전소 시뮬레이터의 개발이 활발히 이루어지고 있다.

항공 시뮬레이터는 실제 비행기의 조종석과 비슷한 환경을 구현하고 그 환경하에서 조종사로 하여금 가상적으로 비행기를 조종하게 함으써 상용 또는 군사적인 목적으로 조종사를 훈련시키는 데 이용되었다. 발전소 시뮬레이터 역시 실제 주제어반과 똑같이 모의 제어반을 구현하여 운전원에게 각종 훈련 상황을 부여하여 다양한 시나리오 상에서의 대처 요령을 숙달시키고 훈련결과를 분석하는 훈련용 시뮬레이터가 대부분이었다.

최근 들어 실시간 시뮬레이션의 응용 분야가 다양화 되고 있는데 즉, 시제품 개발 도구(Rapid Prototyping), 의사 결정 지원(Decision Support)등 다양한 분

야에 적용이 시도되고 있다.

실시간 시뮬레이션의 응용 분야는 크게 네가지로 분류할 수 있는데 즉, 시스템 개발 도구, 훈련용 시뮬레이터, Man-in-the-loop 시뮬레이터 및 의사결정 지원 시뮬레이터등이다.

시스템 개발 도구로서의 실시간 시뮬레이터는 실시간 시스템의 제어기를 개발하기 위한 용도로 이용된다. 대규모의 실시간 시스템을 개발하기 위하여 실제 시스템과 똑같이 동작하는 시뮬레이션 환경을 구축하며 이는 실시간 제어를 위해서 실시간으로 작동되어야 한다. 이러한 시뮬레이터에 실시간 제어기를 접속하여 제어기의 각종 특성을 테스트한다. 또한 실시간 시뮬레이터는 시제품 개발 도구로도 이용되는데 시스템의 복잡도가 매우 높고 개발 비용 또한 엄청나게 소요되는 대규모 실시간 시스템의 개발 초기에 이 시스템의 개발 타당성을 결정하기 위하여 시뮬레이터를 개발하여 이를 검토하는 것이다.

훈련용 시뮬레이터는 항공 및 원자력 발전등의 분야에 많이 이용되어 왔다. 이들 분야는 실제 시스템 운영시 많은 비용이 소요되고 시스템의 운영 오류시에 엄청난 재산 및 인명의 손실이 발생할 수 있는 시스템이기 때문에 고정밀 실시간 특성을 유지하는 시뮬레이터를 개발하여 조종사 또는 운전원을 실시스템에 투입하기 전에 이 시뮬레이터에서 반복 훈련을 시킴으로써 많은 효과를 거두어 왔다.

Man-in-the-loop 시뮬레이터는 자동차, 비행기, 우주선 등 육해공의 운항체등을 개발하는 과정에서 이러한 운항체의 성능, 안정성, 승차감 및 안전 한계 등의 정보를 설계자에게 제공하기 위한 것이다. 이 시뮬레이터를 구축하여 시

뮬레이션의 제어 루프(Conrtol Loop)내에 사람을 사용하여 시스템 운영의 피드백(feedback)을 제공한다. Hardware-in-the-loop 시뮬레이션은 이것과 유사하나 제어시스템이 사람을 보조하거나 대치하는 것이다. 조작자는 종합적인 환경을 경험하며 상황에 대응하는 제어를 조작하고 그 작동의 결과를 경험한다. 실시간 시뮬레이터는 조작자에게 피드백을 주면서 동적 모델을 계속 평가한다. Man-in-the-loop 시뮬레이션의 중요한 공학적 응용은 운항체 테스트 프로그램을 시작하기 전에 운항체의 반응을 결정하고 맨-머신 인터페이스(Man-Machine Interface)를 연구하는 것이다. 예를 들면 새로운 자동차 또는 제어 인터페이스가 설계실을 떠나기 전에 모델링되고 시험 주행될 수 있는 것이다. 운전자는 설계 변경이 가능한 충분히 이른 시점에서 자동차의 특성에 관하여 설계자에게 가치있는 피드백을 줄 수 있다.

의사 결정 지원용의 시뮬레이터는 공정제어등의 분야에 이용된다. 현재 주어진 공장의 앞으로의 행위를 신뢰성 있게 예측하여 미래의 일정 기간에 대해 시스템의 운영 상황을 모사하는데 사용되어, 의사결정의 대안에 대한 성능을 예측할 수 있게 한다. 또한 이것을 사용하여 실시간 스케쥴링(scheduling) 의사결정을 지원할 수 있다. 불확실성이 많은 동적인 생산환경의 어는 한 시점에서 통제 방법을 변경할 때, 변경 전에 생산 감독자는 그 변경이 유리하다는 것을 확신할 수 있도록 신뢰성 있게 예측할 수 있어야 한다. 실시간 시뮬레이션은 공장에서 발생하는 예측하지 못하는 사건들에 관련하여 최상의 대안을 선택할 수 있게 도와준다.

3 절 실시간 시뮬레이션의 연구개발 현황

1. 미시적 실시간 시뮬레이션의 분류

앞에서 말한 것과 같이 실시간 시뮬레이션은 크게 네가지로 분류할 수 있다. 시스템 개발 도구, 훈련용 시뮬레이터, Man-in-the-loop 시뮬레이터 및 의사결정 지원 시뮬레이터 등이다.

고정밀 실시간 시스템을 개발하기 위한 시스템 개발 도구로서의 시뮬레이터는, 실제 시스템과 똑같이 동작하는 시간 특성을 가진 시뮬레이션 환경을 구축하여 실시간 제어기 등의 각종 특성을 테스트하고 시스템 개발 초기에 시스템의 개발 타당성을 검토하는 것이다.

훈련용 시뮬레이터는 항공 및 원자력 발전소에서 많이 이용되어 왔다. 이것들은 실제 조작원이 운전하는 환경을 시뮬레이터를 통하여 유사하게 구현하고, 조작원에게 각종 훈련 상황을 부여함으로써 다양한 시나리오 상에서의 대처 요령을 숙달시키고 훈련결과를 분석할 수 있게 하는 것이다.

Man-in-the-loop 시뮬레이션은 자동차, 비행체 등을 개발하는 과정에서 운항체의 성능, 안전성 등에 대한 정보를 획득하기 위하여 시뮬레이터를 구축하고, 이 시뮬레이션 제어 루프내에 사람을 이용하여 시스템 운영의 피드백을 제공하는 것이다.

의사결정 지원용의 시뮬레이터는 공정제어 등의 분야에 이용되며 현재의 상태에서 주어진 공장의 미래 행동을 신뢰성 있게 예측하고 미래의 일정 기간에 대한 시스템의 운영을 모사하는데 사용되어, 고려하고 있는 의사결정 대안에 대한 성능을 예측할 수 있게 한다.

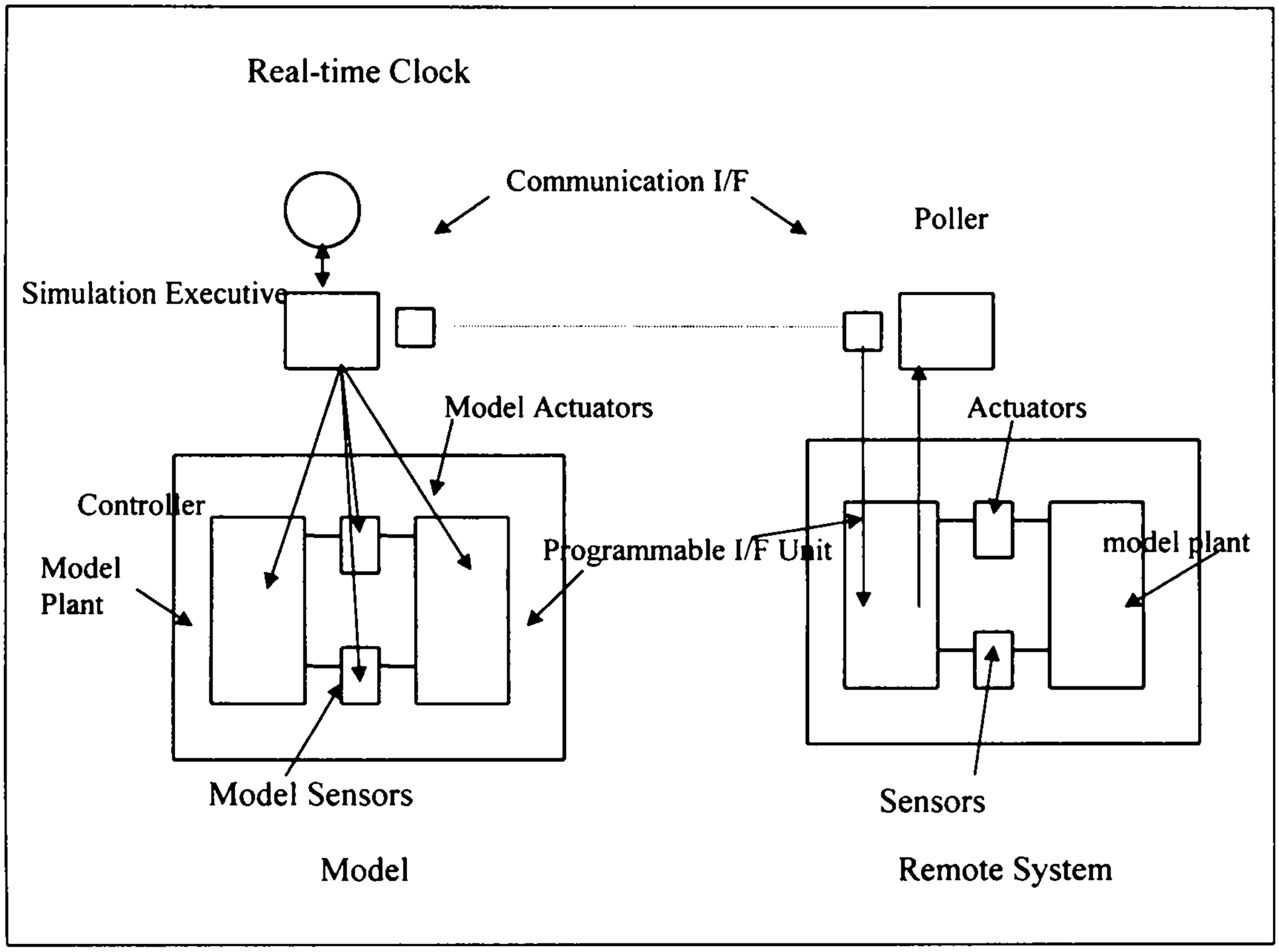
다음은 위의 네가지 분류에 대한 최근의 연구 개발 내용을 소개하고 그것을 표로 요약하여 시뮬레이션 특성 분류표로서 부록에 제시하였다.

가. 시스템 개발 도구

(1) 실시간 제어기 개발

Zeigler 는 자신이 제안한 시뮬레이션 모델인 DEVS-Scheme 을 확장하여 실시간 시뮬레이션 개발환경 구축을 시도하였다. DEVS-Scheme 은 이산 사건 모델의 구성 및 시뮬레이션 구축을 위한 지식기반(knowledge-based) 환경이며 Lisp 형태의 객체 지향 프로그래밍 언어를 이용한다. 이것은 심볼(symbol), 계층(layer) 및 모듈형태의 이산 사건 모델링 접근방법을 결합한 것이다.

인공지능을 기반으로 하는 프로세싱은 처리 시간이 길기 때문에 실시간 운영에 대한 반응 시간 제한 문제에 걸리게 된다. 따라서 지능 제어(intelligent control)의 중요한 설계 문제는 제어의 계층을 마감시간(deadline)의 임박성(stringency)에 따라 나누고 이들 계층을 적절한 컴퓨팅 파워를 가진 하드웨어 프로세서에 할당하는 것이다.



[그림 3-1] 실시간 제어 모드의 DEVS-Scheme 시뮬레이션 환경

현대의 생산 제조 시스템은 복잡한 프로세스들을 요구한다. 따라서 공장(plant)이 어떻게 움직이는가를 표현하고 실시간 제어기를 설계할 때 해석적(analytical) 방법만을 사용하는 것은 매우 어렵다. 따라서 제어기를 구현하기 전에 운영 상태를 시뮬레이션을 통해 확인(validate)하는 것은 매우 중요하다. 실시간 운영을 제공하는 시뮬레이션 환경은 임계 응답시간(Critical Response Time)을 예측하고 고려할 수 있기 때문에 시스템 설계에 많은 도움을 준다. Saridis 는 지능적 제어를 위해 3 개 계층(실행, 조정 및 관리)을 개발하였다. 이 방법은 정확성을 감소시키는 대신 지능(intelligence)을 증가시킨 것이다.

제어와 정보를 각 계층에서 결합하는 방법은 Albus에 의해 최근에 제안된 구조틀(framework)을 특성화한 것이다. 이것은 감각, 의사결정 및 행동을 통합하여 지능과 자율성을 보장한다. Albus의 계층 구분에서는 계층이 증가함에 따라 시간 결정성이 감소하며 따라서 실시간 반응에 대한 데드라인이 완화되어 있다.

모델에 기반한 구조에서는 지식이 각각의 제어 계층에서 모델의 형태로 축적되고 규정된 시스템 목표를 지원한다.

[그림 3-1]의 좌하단은 플랜트(plant), 센서(sensor), 구동기(actuator) 및 제어기(controller)의 요소 모델을 구성하는 DEVS-Scheme의 결합 모델을 나타낸다. 시뮬레이션에서는 제어기 모델이 플랜트 모델에서 작동하는 구동기 모델로 명령을 보내고, 센서 모델로부터 플랜트 상태에 대한 정보를 받는다. DEVS-Scheme에서는 모델과 시뮬레이션 실행기(simulation executive)를 완전히 분리시킨다. [그림 3-1]에 나타낸 시뮬레이션 실행기는 모델 요소들과 상호 작용하여 시간 관리와 사건 구동을 관장하고 있다. 실제로 모델 요소간의 모든 통신은 시뮬레이터를 통해서 전달된다. 시뮬레이터는 통합 명세를 가지고 있어서 모델 요소들에 의해 생성된 결과를 적절한 수신자에게 공급한다.

시뮬레이션이 실시간 제어가 가능하도록 바뀌기 위해서는 시뮬레이션 실행기가 실시간으로 작동하도록 변환되어야 한다. 모델내의 모든 시간들은 시스템 클럭과 관련되어 해석된다. 즉, 시뮬레이션 실행기는 모든 내외부 사건이 그 사건에 대해 계획된 시간과 가능한 한 일치되는 시스템 클럭에 따라 발생되어야 한다.

제어기가 실세계에서와 똑같이 동작하도록 하려면 제어기와 센서, 구동기

모델을 이전에 결합된 상태에서 플랜트 모델이 제거되면 된다. [그림 3-1]의 오른쪽 부분에서 실제 플랜트에 연결된 구동기에 명령을 보내려면 제어기는 그에 대응되는 구동기 모델에 먼저 명령을 보낸다. 이 때, 구동기 모델은 제어 명령을 하위 수준의 명령어로 변환하는 번역기 역활을 한다. 시뮬레이션 실행기는 구동기 제어 결과를 플랜트 모델로 보내지 않고, 통신 채널을 통해 프로그래머블 인터페이스 유닛(Programmable Interface Unit)에 보내서 명령(Instruction Sequence)을 수행시켜 실구동기를 작동시킨다. 플랜트의 응답은 실제 센서에 대응되는 모델 센서를 통해 제어기로 반대 방향으로 전달된다.

실시간 시뮬레이션은 시뮬레이션의 시간적 기초 체계가 컴퓨터 시스템의 클럭 타임과 가능한한 밀접하게 연관되어 있는 시뮬레이션이다. 이상적인 시시간 시뮬레이션에서는 모델에 의해 계획되는 모든 사건들이 계획된 시간에 정확하게 일치되는 실 클럭 타임에 발생한다. 그러나 한정된 처리시간 때문에 실제 사건 시간은 모델에 규정된 값과 달라질 수 있다.

(2) 시제품 제작 도구

시제품 제작 도구(Rapid Prototyping Tool)는 타당성 조사를 위한 중요한 방법으로 인식되고 있다. 이 방법은 개발 비용을 감소시켜 주고 실시간 분산 시스템의 복잡도를 줄여준다. University of Illinois 에서 개발한 UICPBX 는 전화 교환 시스템의 중요한 요소인 PBX(Private Branch Exchange)를 위한 시뮬레이터이다.

하드웨어 기술이 진전됨에 따라 많은 컴퓨터 시스템은 자원을 더욱 더 완전하게 활용하기 위해서 분산 구조를 채택하는 경향이 있다. 이들 중 많은

시스템들이 실시간 성능을 요구한다. 이러한 실시간 분산 시스템의 행동은 매우 복잡하여서 시스템의 명세를 검증하는 것이 매우 어렵다. 따라서 명세에 따라 시스템을 구축하는 것이 타당한지를 미리 알 수 있다면 매우 바람직하다. 시제품 재작은 하나의 타당성 조사 방법으로서, 목표를 달성할 수 있음을 보장해 줄 수 있는 방법이 될 수 있다.

이러한 시제품 재작은 실시간 분산 시스템의 개발 초기 단계에 적절한 방법이 될 수 있다. 실시간 분산 시스템은 그 복잡도가 매우 높고 개발 비용 또한 엄청나게 소요되기 때문이다.

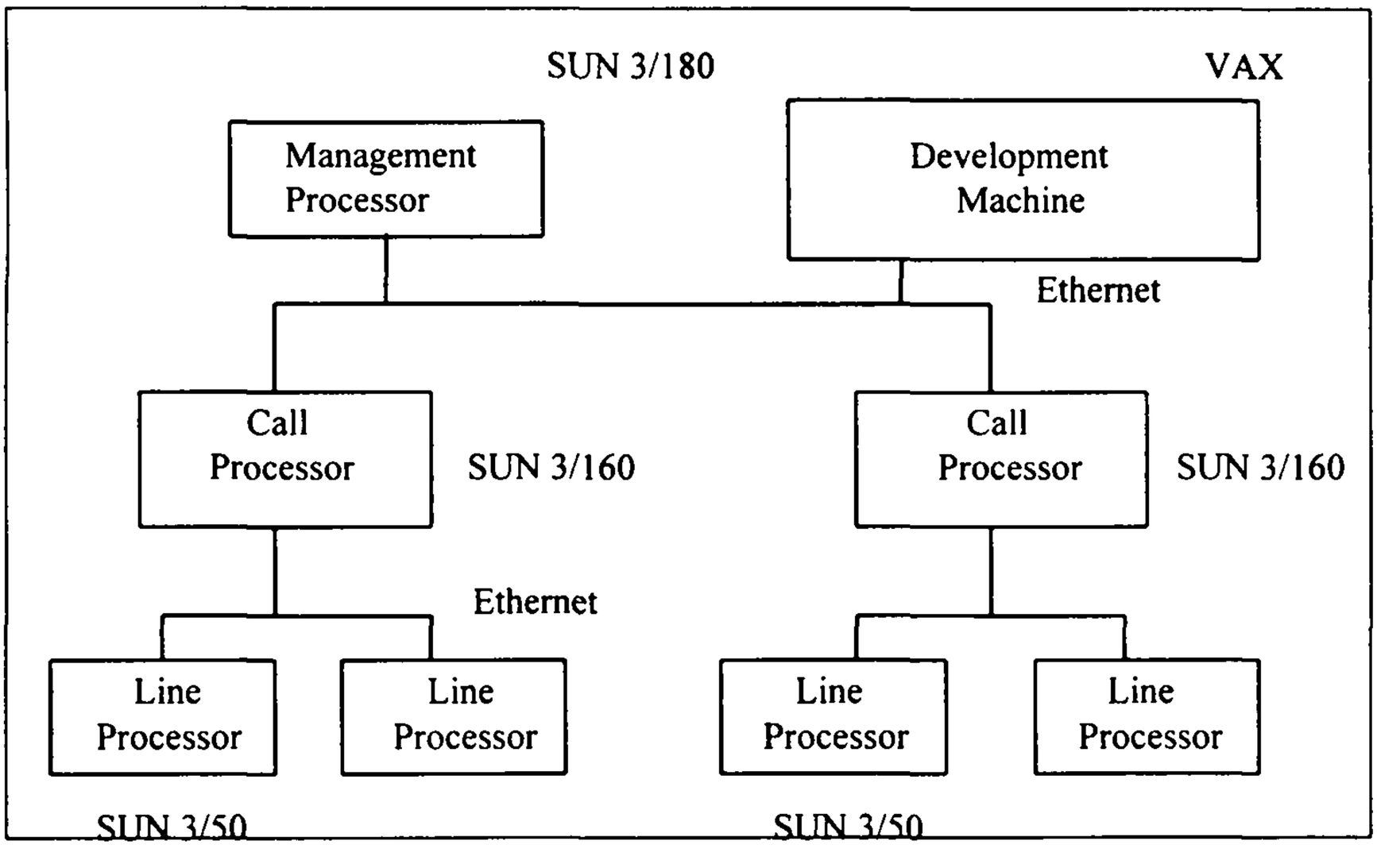
전화 교환 시스템의 하나인 PBX는 점점 분산화 되고, 프로세서간의 통신 또한 매우 복잡한 시스템이다. 전화 교환 시스템은 호스트 컴퓨터 내에서 수행되는 전형적인 교차 개발 환경(Cross-Development Environment)에서 개발되는데, 시뮬레이션 기술은 소프트웨어 시스템을 검증하기 위하여 적용된다.

UICPBX는 대상 시스템의 분산 및 실시간 특성을 시뮬레이션하는 것을 목표로 한다. 이것은 순수한 소프트웨어 시뮬레이션이다. 즉 전화기 또는 통신 회선 등의 하드웨어는 시뮬레이션 대상 시스템에 전혀 포함되지 않는다. 호 처리(Call Processing)와 같은 프로세스들은 여러 SUN 워크스테이션 상에서만 실행되고, 이 워크스테이션들을 연결한 이더넷(ethernet)을 통하여 통신된다. 여기에는 여러가지 기본 시뮬레이션 구조가 구현되어 있다. 타이머가 부착된 실시간 스케쥴러, IPC(Inter-Process Communication) 및 다양한 캡슐화(capsule)된 소프트웨어 기능들이 그것이다. SUN 워크스테이션의 윈도우 시스템과 이더넷을 결합하여 UICPBX는 실시간 분산 시스템의 행위을 명확하게 보여준다.

[그림 3-2]와 같이 UICPBX는 MPR, CPR 및 LPR의 계층구조를 가진다. 이

것들은 각각 다른 SUN 워크스테이션에 할당되며 이더넷으로 연결되어 있다.

UICPBX 에서는 UNIX 운영체제 및 SUN 윈도우 하에서 실시간 다중처리 환경을 시뮬레이션 하기 위하여 실시간 운영체제(Real-Time Operating System, RTOS)를 개발하였다. 또한 프로세서간의 통신을 시뮬레이션 하기 위해 Inter-Process Communication(IPC)을 구현하였다. UICPBX 에서는 이를 이용하여 교환 특성을 시뮬레이션하기 위한 캡슐화된 기능들을 개발하였다. 시뮬레이터의 사용자는 키보드를 통하여 시뮬레이터에 사건을 입력함으로써 시스템과 상호작용을 할 수 있다. 이 환경은 다중 윈도우 시스템을 이용하여 프로세서 또는 터미널에 대응되는 각각의 윈도우에 시스템 응답 및 다양한 정보가 나타도록 한다.



[그림 3-2] UICPBX 의 구조

소프트웨어의 구조는 다음과 같은 3 개 계층으로 구성되어 있다.

(1) 커널(kernel) 기능 : 프로세스간 통신, 실시간 다중 태스크 스케쥴러

(2) 기본 지원 기능 : 계산 기능, 맨 머신 인터페이스(Man Machine Interface) 등

(3) 호 처리 기능 : 기본 호 처리, 다양한 호 처리 서비스 기능등

(3) 시뮬레이션 통합 환경

NASA 에서는 인간에게 가장 위험한 환경중 하나인 우주에서의 우주 정거장 계획인 Space Station Freedom(SSF)을 구축하는 계획을 추진하고 있다. 이 계획에서는 SSF 기능에 맞추어 기계장치를 부착하는 것과 SSF 를 조립하는 시스템 및 절차의 안전성과 효과성을 확보하는 것이 매우 중요한 일이다.

NASA 에서는 시스템과 절차들이 모두 안전 요구 조건을 만족하고 있다는 것을 보증하기 위하여 SSAIAF(Space Systems Automated Integration and Assembly Facility)를 개발하고 있다. 이 장치는 운항 시스템, 원격 로봇 시스템 및 SSF 의 조립 및 유지보수에 관련될 사람들을 통합하여 시험하기 위한 것이다. 즉, 미소 중력 환경(Micro Gravity Environment)에서의 이들 요소들의 기계 및 동적 성능을 실시간 시뮬레이션을 사용하여 시험하기 위한 것이다.

전통적인 방법에서의 이러한 Hardware-in-the-loop, Human-in-the-loop 시뮬레이션의 구축 방법은, 각각의 시나리오를 지원하는 고유한 소프트웨어를 개발하는 방법이었다. 그러나 시나리오 기반 접근 방법은 계속적으로 소프트웨어를 개발하고 유지보수를 하여야 하기 때문에 예산 소요가 매우 크고 프로젝트 일정을 어렵게 한다. 그러나 SSAIAF 에서 SCE(Simulation and Control Environment)는 기본 소프트웨어 요소(모듈이라고 함)들의 개발을 지원하고,

여러 시뮬레이션 시나리오에서 재사용 될 수 있게 한다. 이러한 방법으로 새로운 시뮬레이션 개발의 요구를 줄여 가는 것이다.

SSAIAF 시스템들은 수행되는 모든 시험 및 개발 활동들에 대해 다음과 같은 자원을 제공한다.

- 컴퓨터 자원(Computing Resources)
- 조종사 개발 장치(Manipulator Development Facility)
- 원격 조종사 개발 장치(Mobile Remote Manipulator Development Facility)
- 공기 바닥 장치(Air-Bearing Floor)
- 6 차원 동적 시험 장치(Six-Degree-of-Freedom Dynamic Testing System)
- 시뮬레이션 및 제어 환경(Simulation and Control Environment)

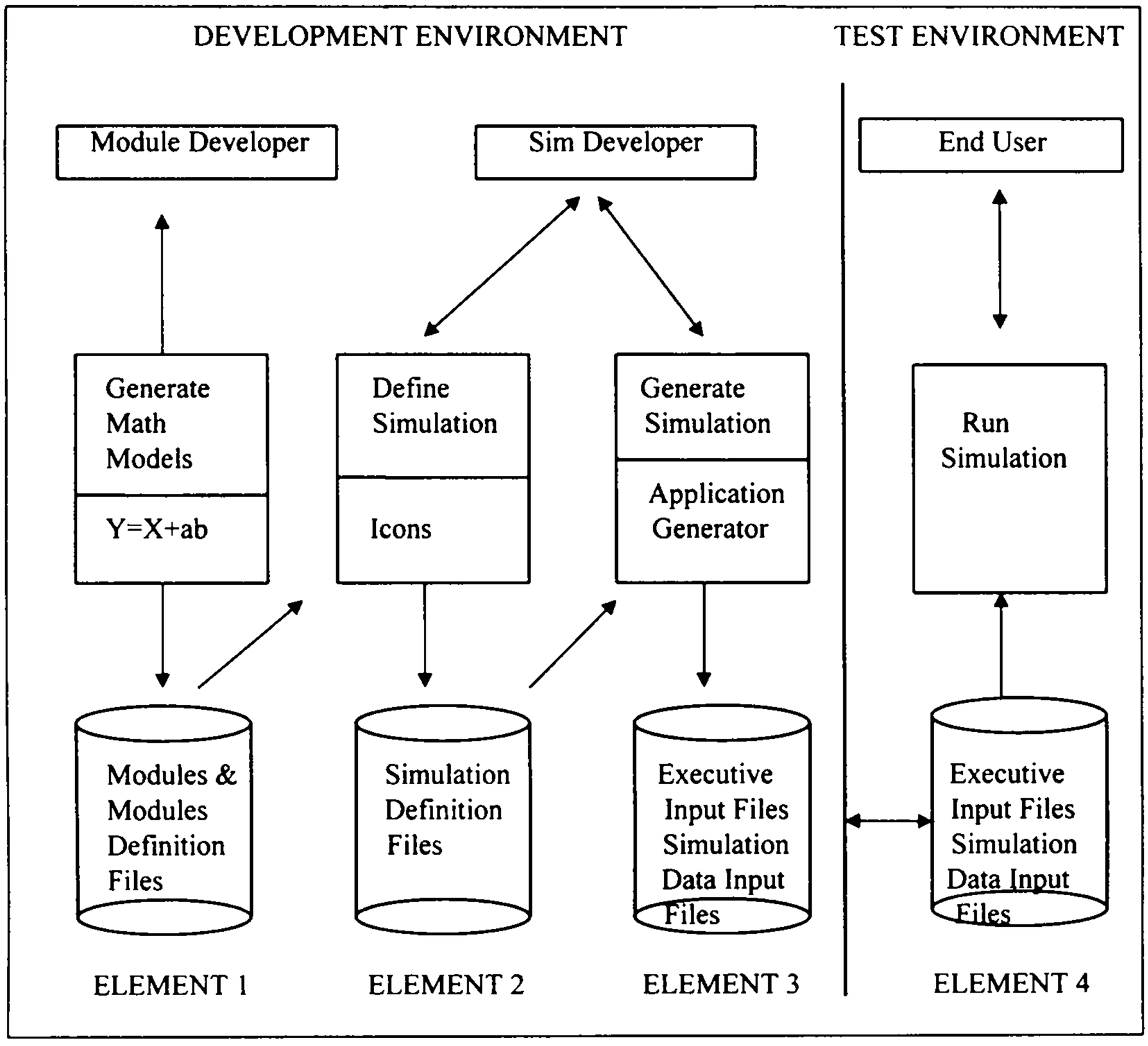
SCE 는 SSAIAF 실시간 시뮬레이션을 개발하는데 필요한 모든 요소들을 통합하는 소프트웨어 시스템이다. 모듈 방식의 접근 방법을 사용하여 이전 시뮬레이션에서 개발된 소프트웨어를 재사용할 수 있게 한다. SCE 를 사용하여 개발된 SSAIAF 시뮬레이션은 테스트 컴퓨터 시스템 상에서 수행할 수 있다. 사용자 인터페이스를 이용하여 사용자는 친숙하게 아이콘(icon)을 사용하여 워크스테이션 상에서 개발 및 유지보수 활동을 수행된다.

SCE 의 소프트웨어 구조에는 시장제품(Commercial-Off-the-Shelf, COTS) 및 개발 제품을 모두 포함할 수 있다. SCE 는 계층 구조를 사용한다. 이는 신속 개발 및 변경 용이성을 제공하여야 하는 응용 소프트웨어를 위한 생성 기법 및 모듈 방법, 재사용 소프트웨어 모듈 방법들을 지원한다.

SCE는 다음과 같은 세가지 이유 때문에 모듈 구조로 설계 되었다. 첫째 이러한 모듈 접근 방법은 완전한 시스템의 유지보수를 지원한다. 이는 시스템 유지보수자 및 개발자가 전체 SCE에 영향을 미치지 않고 특정 모듈을 개정하고 수정할 수 있게 한다. 이러한 모듈 설계는 또한 POSIX에 호환성에 있으므로 하드웨어가 계속적으로 변화되어도 다른 플랫폼에 쉽게 이전될 수 있다.

이 구조의 두번째 이유는 자동적인 시뮬레이션 개발을 지원하기 위한 것이다. SCE는 이전에 개발하는데 들인 노력을 재사용하도록 설계되었다. 따라서 개발자는 많은 개발 작업이 필요한 소프트웨어를 사용할 것인지, 이전에 사용한 시뮬레이션의 부분들을 사용할 것인지, 또는 새로운 작업으로 이들을 조합하여 사용할 것인지를 선택할 수 있다. 예를 들어 개발자는 한 시뮬레이션에서 팔(arm) 모델을, 다른 시뮬레이션에서 추진(propulsion) 모델을 선택할 수 있으며, 이들을 결합하여 새로운 시뮬레이션으로 개발할 수도 있다. 이와 같이 개발자는 수행하는 개발 노력에 가장 적합한 수준에서 소프트웨어를 재사용할 수 있는 능력을 갖게 된다.

이 구조의 세번째 이유는 시뮬레이션 설계자에 의해 전통적으로 결정된 많은 속성을 자동화하여 시뮬레이션 생성을 가능케 한다는 것이다. 예를 들어 SCE 내의 시뮬레이션 수행자 및 스케쥴러는 자동적으로 소프트웨어 모듈(프로세스)들을 공유 메모리를 통해 통신하는 CPU 하나 또는 그 이상으로 할당함으로써 시뮬레이션의 효율을 극대화 할 수 있다. 전통적인 방법에서는 이러한 결정들이 설계자에 의해 임의로 행해지며 시행착오(Trial-and-Error) 방법을 통해 평가되었다.



[그림 3-3] SCE 의 주요 요소

SCE 는 [그림 3-3]과 같은 4 개의 주요 요소들로 구성된다. 첫째 수학 모델은 모듈의 형태로 구현되는데 시뮬레이션 개발자에게 하나의 구축 단위(building block)로 인식된다. 모듈은 그 자체로서 하위 수준의 원시 함수(primitive)로 구축된다. 두번째 요소는 아이콘을 기반으로 하는 사용자 인터페이스이다. 시뮬레이션 개발자는 흐름도와 같은 표기법을 사용하여 특정 시뮬

레이션을 정의함으로써 여러 모듈간의 데이터 종속성 및 시뮬레이션 비율, 지연 간격 등을 표현한다. 특정 시뮬레이션을 나타내는 데이터들은 시뮬레이션 정의 파일에 저장된다. 세번째 요소는 응용 소프트웨어 생성기이다. 이것은 시뮬레이션 정의 요소에 의해 생성된 데이터를 번역하여 시뮬레이션 데이터 파일 및 실행기 입력 파일이라고 불리는 다른 파일에 저장한다. 테스트 컴퓨터에 종속된 명령에 따라 생성되는 실행기 입력 파일은 시뮬레이션 초기 데이터와 함께 테스트 환경으로 이전된다. SCE 의 네번째 요소는 테스트 환경에 상주하면서 실행기 입력 파일을 사용, 실행 환경을 구성하여 실제로 시뮬레이션을 계획 및 제어하고, 필요할 경우 병행 프로세서를 다중 CPU 에 할당한다.

나. 훈련용 시뮬레이터

항공 시뮬레이션(Flight Simulation)은 실제 비행기의 조종석과 비슷한 환경을 구현한 것이다. 상용 또는 군사적인 목적으로 사용되며 조종사는 그 환경하에서 가상적으로 비행기를 조종 기술을 연마한다. 일반적으로 시뮬레이션의 성공 여부는 그 모델이 얼마나 정확하게 실제 세계를 표현하느냐에 의해서 결정된다. 항공 시뮬레이션의 경우는 조종사가 실제 자신이 비행기의 조종석에 앉아 있다고 확신시킬 수 있어야 한다. 이를 위하여 항공 시뮬레이션에서 요구되는 조건들은 다음과 같다.

- 항공 시뮬레이션을 수행하는 조종사에게 실제 비행기의 조종석과 똑같은 환경을 제공해야 한다.
- 시뮬레이터의 디스플레이에 보여지는 영상이 현실감이 있어야 하며 그 움

직임이 실제 비행에서의 움직임과 유사해야 한다.

- 시뮬레이션 중에 조종사가 느끼는 상황의 변화가 실제 비행중에 발생하는 상황의 변화에서 받는 느낌과 유사해야 한다.
- 시뮬레이션 중에 일어나는 조종사의 반응이 미리 제한한 시간 이내에 처리되어 시뮬레이션에 반영되어야 한다.

이러한 요구 사항들의 대부분은 가상현실(Virtual Reality) 기술을 이용해서 해결할 수 있다.

가상현실은 컴퓨터와 입출력 장치를 이용해서 가상 세계를 만들고, 사용자는 이 가상 세계를 현실 세계처럼 느끼면서 정보를 교환하는 형태의 기술이다. 여기서 만들어지는 가상 세계는 실제 설비와 가상적인 영상의 혼합체이다. 사용자가 이 가상 세계를 마치 현실 세계인 것처럼 느끼게 될 때 이 가상 세계는 몰입감을 가졌다고 말한다. 이 몰입감을 갖추기 위하여 항공 시뮬레이션에서 해결하여야 할 중요한 문제는 움직임이다. 실제 상황에서 사람은 시각 정보와 촉각, 그리고 다른 물체에 대한 상대적인 속도감과 공간에서의 방향성 등 움직임에 필요한 정보를 얻는다. 항공 시뮬레이션에서는 앉아 있는 조종사에게 실제 비행중에 조종사가 보는 영상과 유사한 속도로 움직이는 영상을 보여줌으로써 상대적인 속도감을 준다. 조종사가 앉아있는 조종석은 수압 펌프를 이용해서 가속시키며, 바람과 중력등의 요소를 더해서 조종사로 하여금 마치 자신이 실제 조종석에 앉아 있는 것처럼 느끼게 한다. 여기서 조종사에게 보여주는 영상은 가상적인 영상으로 제공되며 조종석, 수압 펌프, 바람, 중력 등은 실제 설비로 구현된다. 이외에도 조종사에게 몰입감을 주기 위해서는

조종사의 반응이 제한된 시간내에 시뮬레이션에 적용되어야 한다.

지금까지 고려한 사항들 중에서 조종사에게 보여지는 영상이 변화하는 시간과 조종사의 반응이 시뮬레이션에 적용되는 시간들에 대해서는 시간 제약이 따른다. 이러한 제약 조건을 만족시키기 위해서는 실시간 시뮬레이션이 반드시 필요하다. 시뮬레이터가 수행하는 각 사건들은 어느 시점까지는 끝나야 한다는 마감 시간이 명시되면서 발생하게 된다. 항공 시뮬레이션의 경우 조종사의 반응을 시뮬레이터에 전달하는 사건은 그 사건이 처리되어야 하는 마감 시간이 명시되며, 시뮬레이터가 조종사에게 전달하는 영상이나 진동, 바람 등의 반응을 처리하는 사건도 마감 시간과 함께 발생한다.

다. Man-in-the-loop 시뮬레이션

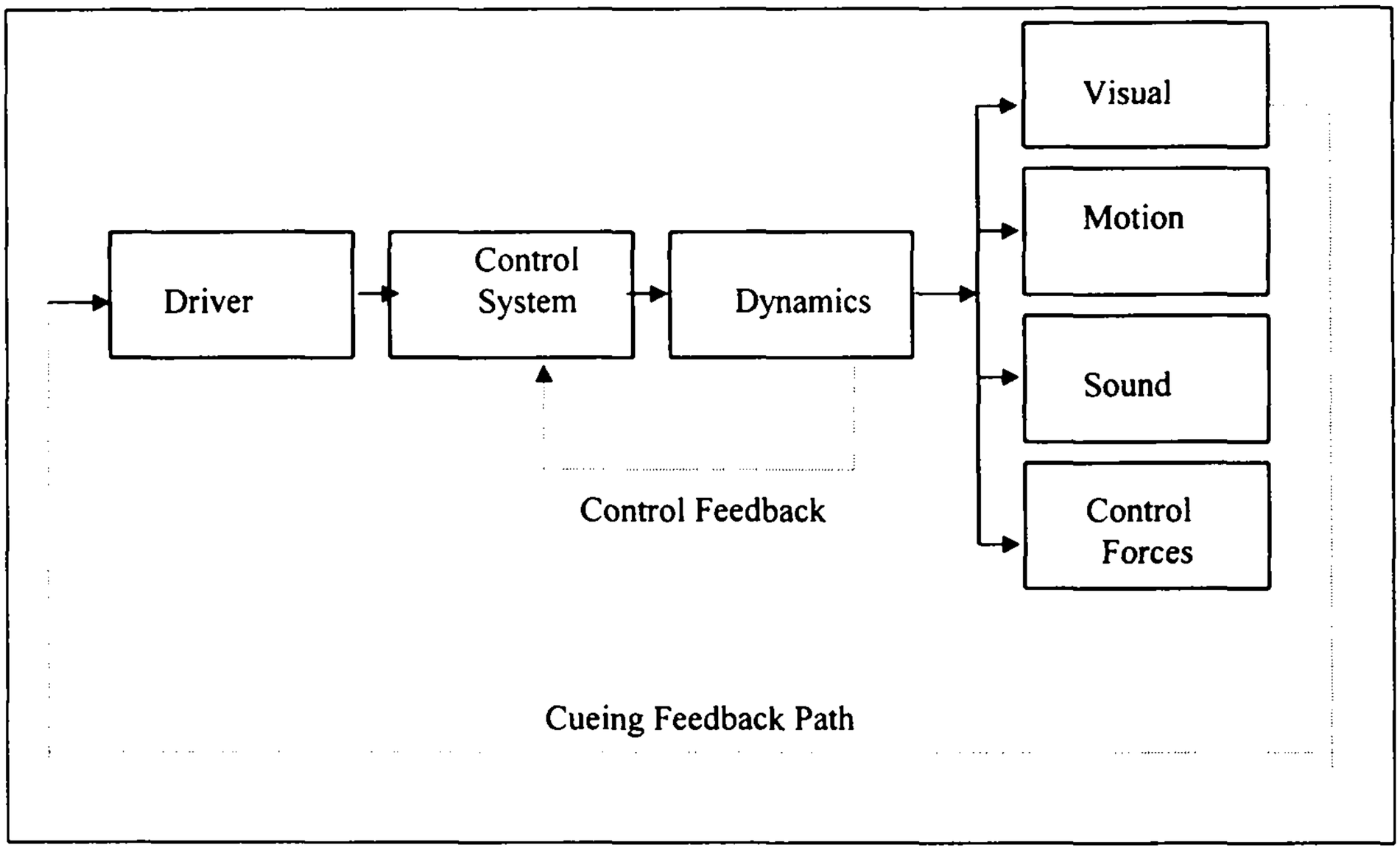
운항체 시뮬레이션을 위해서는 통합 실시간 환경을 구성할 필요가 있다. 이 환경이 효과적인 시뮬레이터가 되려면, 운전자가 자연스럽게 반응하도록 적절한 움직임, 시각, 음향 및 촉각 신호를 제공하여야 한다. 시뮬레이션된 운전 환경은 복잡한 "실시간 작은 세계"의 최상의 예이다. 여기서 사람은 인공적이나 강제적인 컴퓨터가 제어하는 세계와 완전하고 자연스럽게 상호작용하여야 한다.

운전 시뮬레이션은 운항체 시뮬레이션의 흥미로운 특수한 예이다. 왜냐하면 이것은 잘 알려진 풍부한 동적 환경을 가지고 있기 때문이다. 운전 시뮬레이터는 다음을 제공하여야 한다.

- 실제 차량 행동을 예측할 수 있는 정확한 자동차 동역학

- 물체에 의한 적절한 제어 및 속도-거리 판단을 위한 충분한 광학적 흐름 밀도를 제공하는 그래픽스
- 최적 가속 신호
- 물체에 대한 현실 감각을 증진시키기 위한 음향 및 제어력 피드백 신호 장치
- 최소 전달 지연 및 hardward-in-the-loop 능력에 대한 높은 수정 비율

사람에게 허용되는 운항체와의 상호작용의 목표는 제어, 과업 할당 및 안락도에 있어서 사람이 인내할 수 있을 정도의 성능, 안정성, 승차감 및 안전 한계에 대한 정보를 설계자에게 제공하는 것이다. 이 때 시뮬레이션은 설계 대안에 대한 조기 결정을 위한 안전하고 비용 효과적인 방법이다.



[그림 3-4] 시뮬레이션 기능도

지난 20 년간 컴퓨터와 실시간 그래픽스를 이용한 man-in-the-loop 시뮬레이션은 비행기 조종사 훈련, 육지, 해상 및 항공의 공학적 응용 및 우주선등에서 중요한 역할을 하고 있다. man-in-the-loop 시뮬레이션은 시뮬레이션의 제어 루프내에서 사람을 사용하여 시스템 운영의 피드백을 제공 하는 것이다 (hardward-in-the-loop 시뮬레이션은 이것과 유사하나 제어 시스템이 사람을 보조하거나 대치하는 것이다). 조작자는 종합적인 환경을 경험하며 상황에 대응하여 스스로 제어, 조작하여 그 작동 결과를 경험한다. 실시간 시뮬레이터에서는 조작자에게 피드백을 주면서 동적 모델을 계속적으로 평가한다. 이러한 데이터를 기반으로 하여 시뮬레이터는 사람에게 실제적인 환경 신호를 제공

하여 실세계에 있는 것과 똑같이 시뮬레이션 운항체를 제어하도록 하는 것이다. 신호 피드백은 [그림 3-4]에 나타나 있으며 주요 시뮬레이터 기능 관계가 표시되어 있다.

시뮬레이션을 사용하면 훈련 또는 설계 시험을 더욱 낮은 비용으로 할 수 있고, 통제된 실험을 반복적으로 수행할 수 있기 때문에 운항체를 설계하는 데는 최소한의 프로토타입을 가지고 평가할 수 있다.

Man-in-the-loop 시뮬레이션의 중요한 공학적 응용은 운항체 테스트 프로그램을 시작하기 전에 운항체의 반응을 결정하고 맨-머신 인터페이스를 연구하는 것이다. 예를 들면 새로운 자동차 또는 제어 인터페이스가 설계실을 떠나기 전에 모델링 되고 시험 주행될 수 있다는 것이다. 운전자는 설계 변경이 가능한 시점에서 설계자에게 자동차의 특성에 관하여 유용한 피드백을 줄 수 있다. 즉, 잠김 방지 브레이크(anti-lock brake) 및 4 륜 구동과 같은 제어 시스템을 앞선 시점에서 통합할 수 있다. 또한 운전자의 반작용이 측정될 수 있고 여러 대안들이 시험, 비교될 수 있다.

운항 시뮬레이터를 설계할 때, 운항체를 올바르게 조작하는데 필요한 신호들을 제공하는데 특별한 주의를 기울여야 한다. Man-in-the-loop 시뮬레이터는 올바른 신호를 발생하기 위하여, 밀도 높고 급격히 변하는 시각 화면을 제공하기 위해 복잡한 컴퓨터 이미지 생성기를 사용한다.

최근의 발전된 man-in-the-loop 시뮬레이터는 여러개의 복잡한 서브시스템으로 구성되어 있으며, 이들은 서로 결합되어 조작자에게 완전하고 통합적인 운항 환경을 생성시켜준다. Man-in-the-loop 시뮬레이터의 구성 요소들은 다음과

같은 특징을 가지고 있다.

- 호스트 컴퓨터는 운항체의 계산 및 운동 플랫폼의 동역학을 포함한 시스템의 실시간 운영을 제어한다. 이것은 호스트 컴퓨터에 밀접하게 결합되어 있는 가속 하드웨어 상에서 수행된다. 전체 시스템에 대한 모든 계산은 10ms 시간 간격으로 수행된다. 통합 데이터 베이스는 시각 모델, 시뮬레이션 환경의 지형 데이터, 음향에 관련된 데이터 및 운항체의 기계적 특성들을 포함하고 있다.
- 시각 시스템은 호스트 컴퓨터로부터 위치 정보에 기반한 올바른 이미지를 계산한다. 이것은 숨겨진 장소를 제거하고, 태양의 각도를 계산하며, 그늘을 완화하고, 색조를 준다. 이와 같은 계산은 10 GIPS 정도의 계산속도가 가능한 고도의 병렬처리가 필요하다.
- 다중의 CRT 프로젝터는 도움(dome)에 이미지를 디스플레이 한다. 특수한 스크린 자료들은 최대 밝기와 대조를 위하여 사용 된다.
- 운동 플랫폼은 수력 구동기들로 구성되는데 운전실을 x, y, z, yaw, pitch, 및 회전축들을 조합한 각 방향 및 각도로 운동한다. 운동 플랫폼에 대한 제어 신호는 필터링되어 동역학 소프트웨어에 의해 계산된 가상 실세계 운동으로 번역되며 시스템에 의해 성취되는 운동이 된다.
- 음향 서브시스템은 다중 스피커를 사용하여 적절한 음향을 생성하여 엔진, 기어 소리, 바람 소리 등을 시뮬레이션 한다.
- 기구들은 호스트 컴퓨터에 전체 시스템을 감시하고 운전자의 안전을 보장할 수 있는 데이터를 제공한다.

- 제어력 입력기는 실제 자동차에서 발견되는 반작용 힘을 모사하기 위해서 운전대, 브레이크 페달 및 변속 레버를 통해 운전자에게 적절한 힘을 제공하는데 사용한다.
- 운전콘솔은 한 사람이 모든 시뮬레이션 동작을 감시할 수 있게 하여 운항체 시험에 대한 탄력적인 제어를 할 수 있도록 한다.

다중 프로세스 상에서 실시간 제어를 유지하기 위해서는 통신 폭주를 피할 수 있는 특별한 기술을 사용하여야 한다.

라. 의사 결정 지원

이산 사건 시뮬레이션은 보통 "이러한 변화가 발생할 때 성능에 어떠한 영향을 미치는가?" 하는 형태의 질문에 자세한 답변을 주는 "What-if?" 도구이다. 이런 형태의 질문은 보통 시스템이 작동하기 전 시스템 설계 단계에서 발생한다. 시뮬레이션은 "설계당 한번" 결정을 지원하는 오프라인 모드로 사용되며 결정된 선택들은 시스템의 생명주기에서 변경될 수 없는 상태로 남는다.

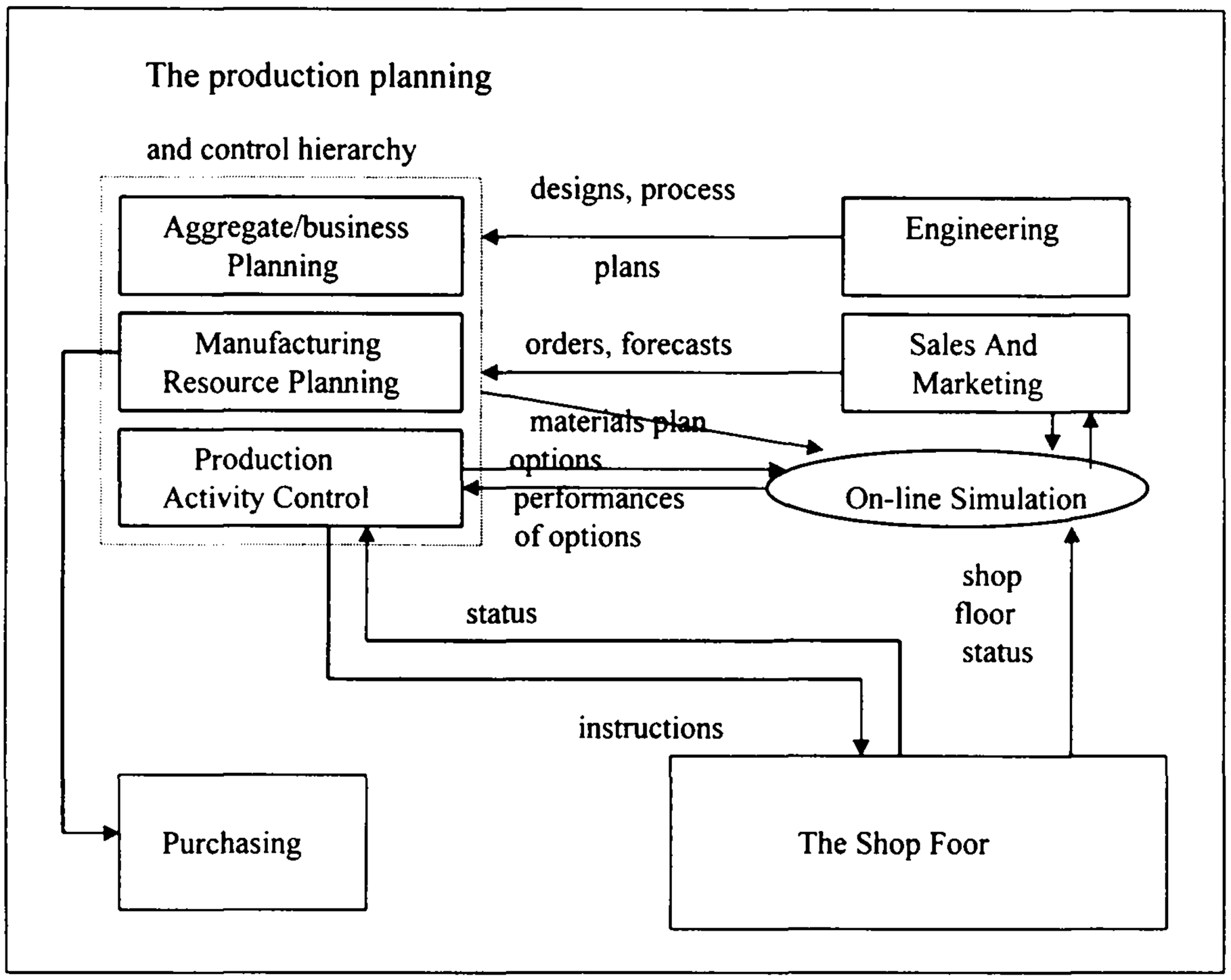
그러나, 시뮬레이션을 응용할 수 있는 범위를 확대하기 위한 관심이 점차 증가하고 있다. 즉 시스템의 전 생명 운영 주기에 걸쳐서, 주기적이며 반복적인 의사 결정을 지원하기 위해 시뮬레이션을 사용하자는 것이다. 이러한 온라인적인 역할의 시뮬레이션은 현재의 결정에 대한 조언을 줄 수 있는 "What NOW?" 도구인 것이다.

온라인 시뮬레이션이 전형적인 생산 계획 및 통제 기능에 적용되는 예를

살펴보기로 한다. 온라인 시뮬레이션은 조직의 특정한 기능 영역을 통합하는 능력을 제공하는데 이는 적어도 두 분야에 효과를 가져온다. 이 두가지 효과는 현재의 상태에 주어진 공장의 미래 행동을 신뢰성 있게 예측하는 능력에서 비롯된다. 온라인 시뮬레이션 모델은 현재의 시스템 상태로 초기화 될 수 있다. 또한 미래의 일정 기간에 대한 시스템의 운영을 모사하는데 사용되어, 고려하고 있는 의사결정 대안에 대한 성능을 예측할 수 있게 한다.

온라인 시뮬레이션의 첫번째 효과는 판매 마케팅 기능에게 고객에 대한 주문 완료 시간을 신뢰성있게 예측해 주는 것이다.

온라인 시뮬레이션의 두번째 효과는 이것을 사용하여 실시간 스케쥴링 의사결정을 지원할 수 있다는 것이다. 불확실성이 지배하는 동적인 생산환경에서 공장이 어떠한 시점에서 현재의 통제 방법을 변경하는 것이 필요할 때가 있다. 이러한 변경이 가해지기 전에 생산 감독자는 그 영향을 예측할 수 있어야 한다. 온라인 시뮬레이션을 이용하면, 공장에서 발생하는 예측하지 못하는 사건들을 고려하여 최상의 대안을 신속히 선택할 수 있다.



[그림 3-5] "What Now?" 도구로서 온라인 시뮬레이션

만약 제조 환경에 불확실성이 전혀 개입되지 않는다면 오프라인으로 생성된 생산 계획이 정확하지만, 실제로는 어느정도의 불확실성이 항상 존재한다. 즉 기계의 고장, 작업자의 결근, 자재 소진, 주문 상태 변경, 생산량 변화 및 빈번한 설계 변경 등이 발생하기 때문에, 현재의 이벤트들을 고려하여 오프라인으로 생성한 계획은 변경되어야 한다. 이 사실은 어느 정도의 스케쥴링 의사결정을 온라인으로 하고, 변경에 의해 영향을 받는 공장의 부분들을 효과적으로 재스케쥴링할 필요성을 제기한다.

온라인 시뮬레이션의 한가지 문제점은 수행 속도 즉 시뮬레이션 시스템으로부터 답을 얻는데 걸리는 시간이다. 이 문제의 한가지 가능한 해결책은 전문가 시스템과 시뮬레이션 기술을 연결하여 사람의 입력을 필요로 하는 많은 일을 수행하는데 있어서 인공지능을 사용하는 것이다. 시뮬레이션은 대안의 성능을 예측하는데 사용하고 전문가 시스템은 성능을 평가하여 더 나은 대안을 선택하도록 안내하는 역할을 하는 것이다.

2. 국내의 실시간 시뮬레이션의 활용 현황

가. 개요

국내의 대규모 실시간 시뮬레이터를 조사하는 목적은 실시간 시뮬레이터가 실시간 시스템의 구축에 필요한 기술로서의 사용될 수 있는가를 확인하기 위한 것이다. 기존의 시뮬레이션 시스템들을 사용할 경우의 한계점들을 조사하고 개선하기 위해 실시간 시스템 시뮬레이터의 가능성을 알아보기 위한 것이다.

실시간 시뮬레이터가 사용될 수 있는 응용 분야들은 다음과 같은 특성을 가지고 있다.

- 시스템과 시스템이 작동하는 환경은 많은 상호 작용을 하며 환경은 동적으로 변하므로 이에 대처할 수 있어야 한다.
- 입력과 출력의 병행 처리가 가능해야 한다.
- 논리적인 정확성과 더불어 실시간성의 보장이 되어야 한다.
- 지리적 이유 또는 연산 능력이나 신뢰도를 높이기 위해 많은 경우 분산/병렬 처리를 한다.
- 수행하는 업무의 중요성 때문에 높은 신뢰도가 보장되어야 한다.

● 기능의 추가와 변경에 따른 시스템의 확장을 고려하여야 한다.

본 조사에서는 위의 사항을 고려하여 국내에서 개발된 대규모의 실시간 시스템의 대표적인 것으로서 발전소 운전원 훈련용 시뮬레이터들을 조사하고 있다. 이와 함께 포항 종합 제철소에서 사용되고 있는 시뮬레이터를 찾아보았다.

나. 발전소 운전원 훈련용 시뮬레이터

(1) 개발 배경

대부분의 발전소에서의 프로세스는 자동 제어 시스템을 적용하여 중앙 제어실(Main Control Room)에서 집중 관리, 제어되고 있다. 즉, 대부분의 발전소 기기의 조작과 운전 상태의 감시는 중앙 제어실에서 이루진다. 발전소가 정상적으로 운전될 경우 시스템은 자동으로 제어되기 때문에 운전원이 직접 조작할 기기는 별로 없고 발전소의 정지 건수도 매우 적다. 따라서 운전원들이 운전 경험을 축적할 수 있는 기회는 많지 않다. 그러나 발전소의 경우 일단 사고가 발생하면 그 피해는 매우 크다. 원자력 발전 사상 최악의 사고인 미국의 TMI 원전 사고와 소련의 체르노빌 원전 사고 등에서 중요한 사고원인의 하나는 발전소 운전원의 실수였다.

따라서 중앙 제어실에서 발전소 운전을 담당하는 운전원은 고도의 학술적인 지식과 시스템에 관한 완벽한 이해를 바탕으로 풍부한 운전경험을 가져야 한다. 또한 사고에 효과적으로 대처하고 미연에 방지할 수 있도록 운전원들의 운전 능력을 향상시키고 유지시키기 위해 효과적이고도 체계적인 교육이 필요하다. 시뮬레이터를 통해 운전원들은 다양한 상황에 직면할 수 있으며 효율

적으로 대응할 수 있는 방안을 교육받을 수 있다.

발전소 운전원 훈련용 시뮬레이터는 실제 발전소의 주제어반과 같은 기능을 가지도록 구현된 모의 장치이다. 이를 이용하면 실제 발전소에서 경험하기 여려운 사고에 대한 훈련과 각 단계별로 다양한 운전 상태의 경험 또는 운전이 가능하다. 또한 지난 운전 상황을 반복하여 재현하거나, 현재 상태를 정지 및 천천히 동작시켜 계통의 변화를 이해 또는 분석할 때 상당한 도움을 준다. 이것은 발전소 운영 보수의 감독자, 보수 요원, 규제 기관의 운영 감독 요원 등의 기본 교육에도 필수적이며, 나아가서는 발전소 핵증기 공급 계통(nuclear steam supply system) 및 부품 설계, setpoint 연구, 그리고 안정성 연구에도 활용할 수 있다.

훈련용 시뮬레이터는 범위에 따라 축소형(Compack simulator)와 완전형(full scope simulator)으로 나눌수 있다. 축소형은 기본 동작 원리의 교육 등에 쓰이며 실제 제어 훈련용으로는 완전형 가운데서도 실제 대상 시스템과 외관 및 성능이 차이가 없는 완전 복제형(replica type)이 훈련효과면에서 뛰어난 것으로 밝혀져 보편화되어 있다.

발전소 운전원 훈련용 시뮬레이터는 실제 주제어반과 똑같이 구현된 모의 제어반과 운전원에게 각종 훈련상황을 부여하고 훈련결과를 분석하는 강사조작반(instructor station), 그리고 컴퓨터실로 구성된다.

실제 발전소의 주제어반은 원자로나 보일러 등 발전계통과 연결되지만 모의제어반의 모든 계장 제어 장치는 시뮬레이션 컴퓨터로 연결되며 컴퓨터는 발전계통 모델을 수행시킨 후 결과를 다시 모의제어반으로 보낸다. 강사는 강사 조작반을 통하여 임의의 상황을 모의제어반에 나타나게 하고 운전원은 주

어진 상황에 대처해야 하는데 운전원의 대처 상황은 모두 컴퓨터에 기록되며 이 기록은 추후 평가자료로 쓰인다.

(2) 시뮬레이터 개발 현황

한국 전력공사에서 가동하고 있는 원자력 발전소는 1994 년 현재 9 기이고 1990 년대 말까지 7 기가 추가로 준공되어 가동될 예정이다. 현재 한국 전력공사는 국내에 설치된 발전소 원자로 노형별로 3 기의 운전원 훈련용 시뮬레이터를 설치, 운영하고 있으며 또한 건설중에 있는 원전용으로 '90 년대 말까지 3 기의 시뮬레이터를 추가로 설치할 예정이다. 또한 화력발전소 운전원 훈련용으로 1 기의 시뮬레이터가 운전중이며, 1 기가 추가로 설치될 예정이다. 원전 운전원 훈련용 시뮬레이터 제작 비용은 매우 고가로 1994 년 현재 호기당 가격은 $1 천만 ~ $1 천 5 백만 정도에 달하고 있다. 현재 한전내에서 설치 운영되고 있는 원전 시뮬레이터들은 3 기로 세부내용은 다음 [표 3-1]과 같다.

[표 3-1] 시뮬레이터의 현황

구분	시뮬레이터 #1	시뮬레이터 #2	시뮬레이터 #3
훈련대상	고리 #1,2	고리#1, 2 및 영광 #1,2	울진 #1,2
제작사	EAI	Westinghouse	Thomson
reference plant	Surry #1	영광 1 호기	울진 1 호기
설치 일자	'78.7	'86.12	'90.2
설치 장소	원자력 연수원	원자력 연수원	울진원자력

기존에 운영되고 있는 완전형 시뮬레이터(full scope simulators)들은 모두 turn-key 방식으로 도입되어 국내의 시뮬레이터의 기술을 확립하지는 못하였다. 그 결과 유지보수 및 성능 개선등에서 문제점이 많이 발견되었고 이러한 문제점을 해결하기 위해 1994 년초, 한국전력공사 기술연구원, 삼성전자 그리고 미국의 S3 Technologies 사의 연구원들로 구성된 개발원들에 의해 3 PACK 으로 명명된 원자력 및 화력 발전소용 시뮬레이터 개발 프로젝트를 시작하였다. 이 과제는 한국의 시뮬레이터 산업을 발전시키고, 운전원 훈련 요구조건 및 ANSI/ANS 3.5 의 시뮬레이터 제조 요구사항을 따라 발전소 시스템의 적절한 모형화 능력을 개발하고 국내 시뮬레이터 전문가를 활용하기 위한 시도였다.

1994 년 3 월에서 1998 년 7 월까지 52 개월에 걸친 총개발 기간동안 다음 [표 3-2]와 같은 개발 일정을 가지고 있다.

[표 3-2] 한전의 시뮬레이터 개발 일정

시뮬레이터	기간	대상 발전소
영광 3,4 호기 시뮬레이터	'94.3.9-'96.11	영광 3 호기 NPP 1000MWe
보령 3,4 호기 시뮬레이터	'94.7.9-'96.10	보령 3 호기 FPP 500MWe
고리 2 호기 시뮬레이터	'95.7.9-'98.7.8	고리 2 호기 NPP 650MWe

위의 시뮬레이터들은 개발 대상 시스템인 원자력 발전소(영광 3,4 호기, 고

리 2 호기)와 화력 발전소(보령 3,4 호기)의 모든 계통들을 모델링하여 정상 및 비정상 운전 상태 등을 시뮬레이션하기 위한 완전 복제형 시뮬레이터 개발을 목표로 하고 있다. 또한 국제적으로 널리 적용되고 있는 ANSI/ANS-3.5 기준을 충족시키는 실시간 기능을 구현하기 위한 모델링 및 시뮬레이션 기술을 95% 이상 국산화하겠다는 목표로 진행중이다. 아울러 주제어반의 미터, 스위치, 제어용 컴퓨터 등을 포함하는 각종 계기 및 제어기의 동작과 내부 구조를 시뮬레이션 목적에 맞게 분석, 개조할 수 있는 기술 확보도 포함된다.

발전소 운전원 훈련용 시뮬레이터를 국내에서 개발하게 되면 다음과 같은 효과도 기대할 수 있다.

- 시뮬레이터를 도입할 경우 기당 $1,500 만(약 120 억원)이 드는데 비해 개발할 경우 비용은 기당 80 억~90 억원이 소요되기 때문에 비용 측면에서도 매우 유리함을 알 수 있다.
- 축적된 기술력을 바탕으로 기존 발전소 중 시뮬레이터가 설치되어 있지 않은 곳이나 향후 건설되는 발전소에 대한 시뮬레이터의 공급이 쉬워진다
- 적용 범위를 훈련용에서 신규 개발에 적용할 기술용으로 확대할 수 있어 차세대 원자로 설계시 이에 대한 안정성 및 경제성 제고에도 기여하리라 기대된다.

(3) 원자력 발전소 운전원 훈련용 시뮬레이터의 성능 기준

원자력 발전소 운전원 훈련용 시뮬레이터(이하 발전소 시뮬레이터)는 기존 발전소와 같은 환경의 중앙제어실과 시뮬레이터 운전 상황을 감시하고 제어

하는 instructor console 및 컴퓨터 시스템으로 구성된다. 중앙제어실의 각 panel 과 컴퓨터간 입출력 신호수는 10,000~15,000 개이다.

시뮬레이터의 주요 기능으로는 발전소 정상 운전 및 사고로 인한 비상 훈전시 발전소 운전원의 운전 절차 숙달 및 비상 대응 능력을 배양시키는 훈련 목적에 주로 사용된다. 그 외 발전소의 각종 운전 절차서(Operation procedure)를 확인하며 발전소 과도 현상(Transient analsysis)을 분석 평가하는 도구로 활용된다. 정상 운전 및 비정상 운전을 비롯 각종 설비의 성능 저하, 고장 등으로 인한 비상사태까지도 포함되며 이러한 상황 부여 및 결과 분석을 위한 주요 기능으로는 초기 조건, 설정, 이상, 고속/저속 시간, 역추적, 재현, 동결 등이 포함된다.

발전소 시뮬레이터의 주요 기능은 다음과 같다.

- initial condition: 실시간 시뮬레이터의 시작 조건
- backtrack: 어느 이전 시간으로 되돌아감
- fast/slow time: 실시간 대비 1/10 ~ 10 배의 속도
- malfunction: 발전소 부속 장비의 고장 및 성능 저하 상황의 부여
- replay: 시뮬레이션 운전 상황을 15 초에서 1 분 단위로 기록, 동일한 시뮬레이션 환경 재생
- override: 모델 및 계기의 데이타 수정
- snapshot : 어느 순간의 환경을 저장
- freeze: 시뮬레이션의 의도적 정지
- LOA(Local operator action): : 현장기기의 조작

- parameter monitoring: 다양한 형태의 운전 변수 감시
- automatic exercise: 훈련 시나리오의 자동 훈련
- trainee evaluation and monitoring : 운전원을 객관적으로 상세히 평가할 수 있도록 기록

(4) 개발 환경

시뮬레이션용 컴퓨터 기종은 대형 호스트 중심에서 workstation 으로 다운사이징되고 있다. 이러한 경우 UNIX 에 실시간 기능들이 추가된 운영체계를 채택하고 있다. 사용언어의 경우 수행시간의 최적화를 고려하여 Assembler, C, C++ 등이 많이 사용되고 있다. 발전소 시뮬레이터에서는 수식 계산은 FORTRAN, 제어계통은 Assembler, 일반 프로그램은 C 로 주로 이루어지며 사용자 편이를 강화하기 위해 X-Window 등의 환경에서 개발한다.

(5) 발전소 운전원 훈련용 시뮬레이터의 내용

운전원 훈련용 시뮬레이터의 구조 및 개발 내용 등은 원자력연구소에서 개발한 KAERI-CNS(Compact Nuclear Simulator) 시스템을 대상으로 살펴보기로 한다.

(가) 개발 과정

1989 년에서 1988 년에 걸쳐 스웨덴 STUDSVIK 사와 공동으로 개발하여 KAERI 연수원에 설치하였으며 CNS 하드웨어 제작은 국내에서 수행되었고, 소프트웨어 개발은 STUDSVIK 사와 공동으로 스웨덴에서 수행하여 국내에서

보완과정을 거쳐 완성하였다.

(나) 연도별 업무 수행 내용

1985 년부터 1988 년까지 수행한 업무들의 내용은 다음의 [표 3-3]와 같다.

(다) 목 적

CNS 는 발전소의 제러반 전체를 모의화한 full scope simulator 와는 달리 원자력 발전소 기초 교육, 계통 설계 및 안전성 연구에 필요한 주요 계통 또는 일부만을 모의화하여 제작한 모의 장치이다. 이것은 주로 운전 요원의 초기 및 재교육 훈련을 제공하기 위한 훈련 도구이다.

(라) 사용 분야

현재 개발된 시스템이 사용되고 있는 분야는 다음과 같다.

- 발전소 핵증기 공급 계통(nuclear steam supply system) 설계 요원 훈련
- 원자력 연구 개발을 위한 연구원 교육 훈련
- 한전 소속 원자로 조종사 보수 교육
- 계측 제어 연구시 제어 알고리즘 시험

[표 3-3] 연도별 CNS 개발 내용

년도	주요 수행 내용
1985 년 7 월	. 자료 수집 및 검토 . 개념 설계

1986	. CNS 공동 개발 계약 체결 . Malfunction 선정 . Console panel 설계 및 보완 작업 . Measuring instrument list 작성
1987	. Console panel 제작 . Interface card 제작 . Graphic display system . Meth. Modeling . Software module 개발 . S/W, H/W integration . Functional test procedure manual 제작
1988	. Factory acceptance test . 현장 설치 . Site acceptance test . 시운전

5) 시스템 구성

다음 [그림 3-6]은 CNN 의 전체 시스템 블록 구조를 보여주고 있다.

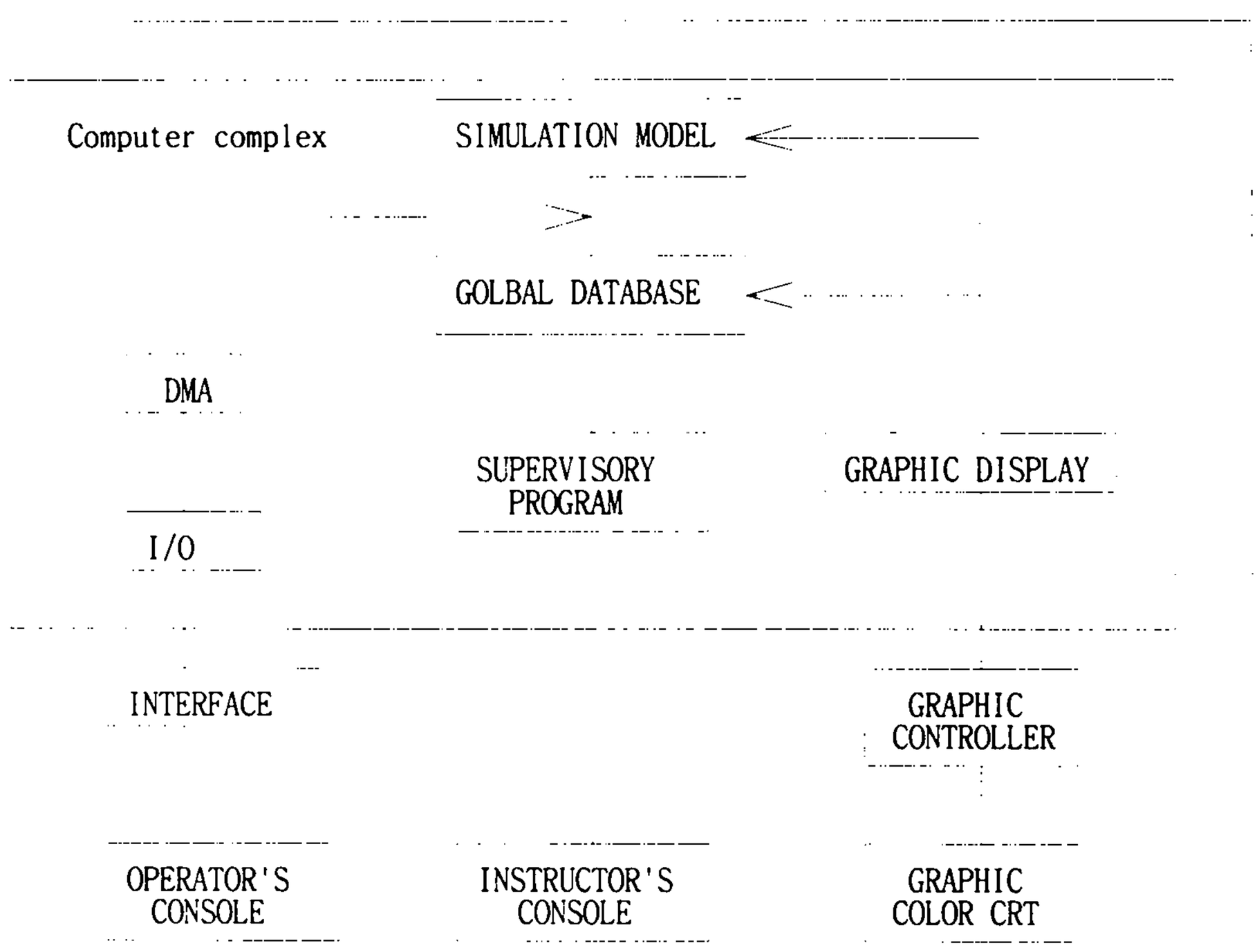
CNS 는 크게 하드웨어와 소프트웨어로 나뉜다.

하드웨어는 주제어실의 제어반을 축소한 operator console, 실제 발전소와 유사한 상황을 모의화하는데 필요한 모든 데이타를 발생시키는 컴퓨터 조직, 컴퓨터와 console 사이의 입출력 정보를 전달하는 interface 카드, 3 개의 컬러 CRT 에 표시되는 full graphic display 계통, 훈련원에게 각종 명령을 내리고 훈련원의 훈련 내용을 감시할 수 있는 강사 조작반으로 구성된다. [그림 3-7]

CNS 의 두뇌인 컴퓨터는 Digital Equipment Co 의 MicroVAX II 두 대를 사용하고 있고, 주변 장치는 다음과 같다.

- MicroVAX II(BA 123 enclosure, 5MB memory)
- DRV11-J parallel interface : 컴퓨터에 설치되어 cable 을 통해 interface 카드

와 판넬에 관한 정보를 교환한다.



[그림 3-6] CNS 의 전체 블록 구성

- DHV11 serial interface: terminal, printer 등 주변 장치를 컴퓨터에 연결 사용하는 장치이다.
- DECNET interface
- LA210 operator console
- RD53 71Mbyte 5.25 inch Winchester disk
- TK50 95Mbyte cartridge tape

- QP24M 200 cps dot matrix printer
- Doosan 220 CRT terminal
- C3920 color graphic terminal

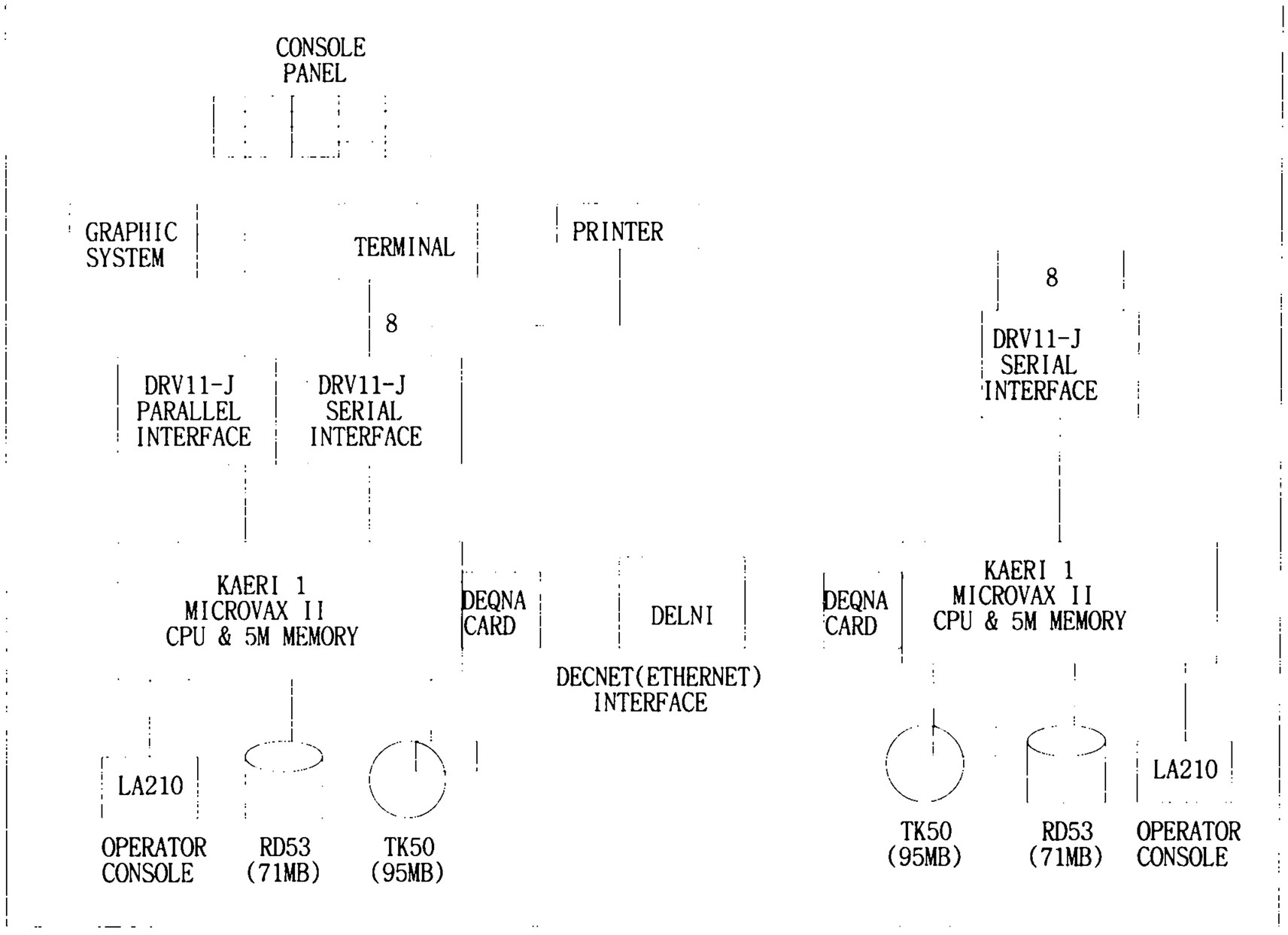
두 컴퓨터 사이는 Ethernet interface 로 연결하여 필요한 데이타의 전송을 가능하게 한다.

소프트웨어는 발전소 각 계통의 동작을 표시한 수학적 모델링 프로그램, 강사 조작반에서의 각종 명령을 처리해 주는 supervisor 프로그램, 공용 데이타베이스를 관리하는 공통 데이타 화일 프로그램, 훈련 시작 전 판넬 상의 램프, 기록계, 스위치 등 하드웨어의 동작 여부를 점검할 수 있는 프로그램으로 구성된다.

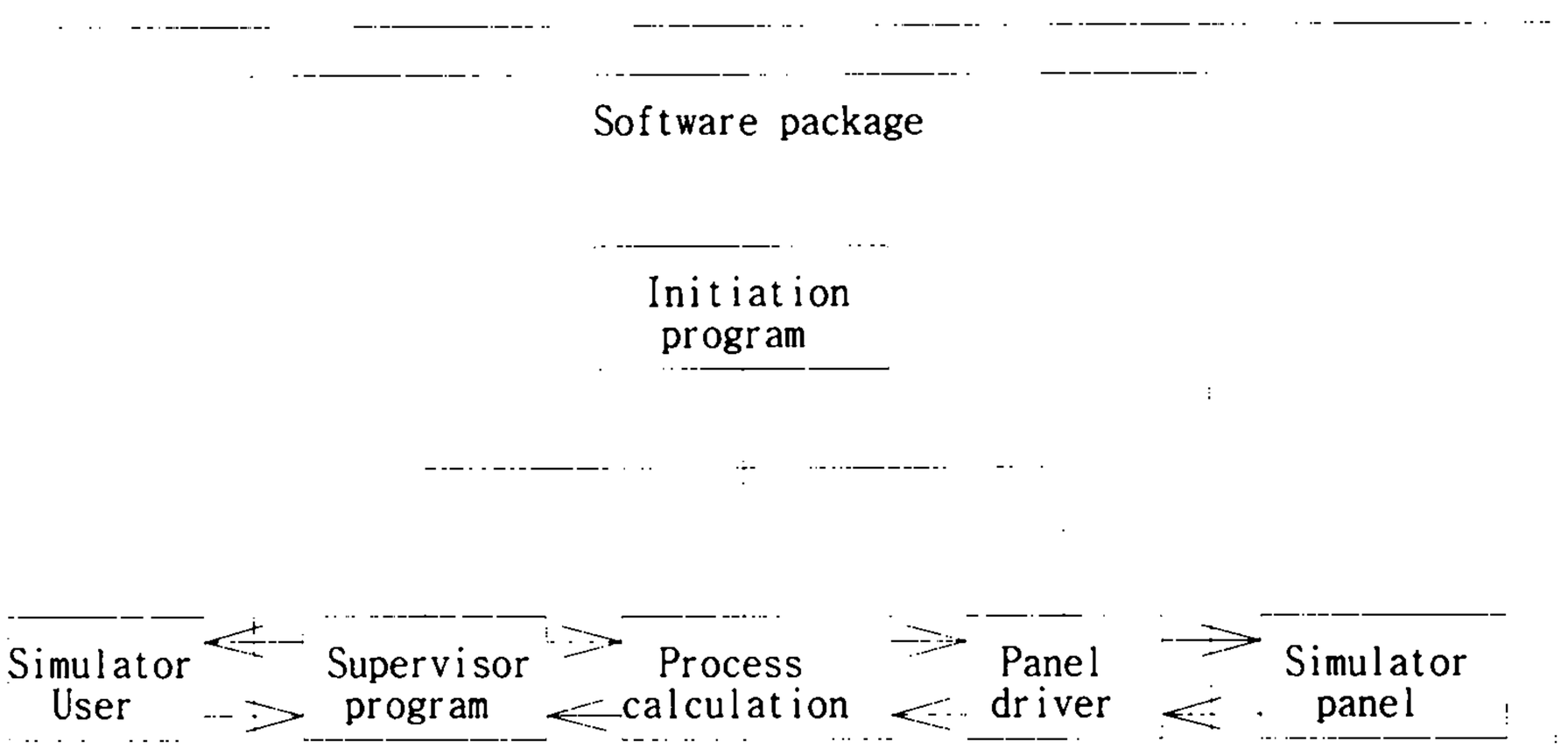
CNS 소프트웨어는 우선 순위에 따라 실시각 프로그램과 background 프로그램으로 나눌 수 있으며, 실시각 프로그램은 고정된 우선 순위를 가지고, background 프로그램은 시분할 시스템에 연결되어 변하는 우선 순위를 갖는다. 동적 계통 모형화 프로그램은 실시각 프로그램이며, 데이타베이스 관리 프로그램인 CDF 프로그램은 background 프로그램이다.

- CNS 소프트웨어는 계통 모의화 프로그램, Supervisor 프로그램으로 나뉘어지며, 모의화 소프트웨어는 기능에 따라 초기화 부분, 사용자 interface 부분, 실시각 실현 부분, 계통 모의화 부분 및 판넬 구동 부분으로 나누어지며 CNS 소프트웨어의 블록 구조는 다음 [그림 3-8]과 같다.

- 초기화 부분: 판넬 초기화, 계통 모의화 변수의 초기값 부여, Supervisor 프로그램 초기화 등 다른 부분의 소프트웨어를 초기화한다.
- Supervisor 부분: backtrack, replay, snapshot, display service, 계통 모의화값 표시등 강사 조작반에 관련된 역할을 수행한다.
- 강사 조작반: MicroVAX II 컴퓨터와 연결된 VT220 터미날로 구성되며 simulator 동작중에는 supervisor 명령을 처리하는 곳이다. Supervisor 명령에는 기본명령어와 종속 명령어가 있으며, 기본 명령어는 simulator 운영에 관련되는 명열어로 구성되어 있고, 종속명령어는 운전상황을 저장하거나, 재현하고 또 변수값을 표시하는 기능 등의 simulator 운전에 필요한 기능들이 포함된다.
- 계통 모의화 부분: 정적 프로그램과 동정 프로그램으로 나누어지고 정적 프로그램은 다시 초기화 루틴과 indata 화일로 나누어지며, 계통별 하위 모듈로 구분된다. 전체 가압 경수로 계통은 원자로 냉각로을 포함하는 SMABRE code, 원자로 용기, 가압기 압력 및 수위 제어, 증기 발생기 수위 제어, 화학 및 체적 제어 계통, 보조 계통 (instrument air, 기기 냉각수 계통과 격납용기 계통), 주증기 계통, 터빈 계통, 전기 계통, 급수 및 복수 계통 그리고 원자로 보호 계통으로 구분하였다.



[그림 3-7] CNS 의 하드웨어 구성



[그림 3-8] CNS 의 소프트웨어 블록 구성

- 정적 프로그램은 100% 출력 상태시의 초기 조건을 입력하는 초기화 루틴과 모의화 과정에서 필요한 각종 데이타를 저장하는 indata 화일로 구성된다. Indata 화일은 동적 루틴에서 사용되는 데이타를 가지고 있으며, simulator 데시트시 값을 변경할 때 동적 루틴을 변경하여 compile 및 link 할 필요없이 indata 화일만 변경하고, Supervisor 모드에서 초기화할 때 데이타가 common 변수로 입력된다.

6) 발전소 시뮬레이터의 개발 효과

축적된 원자력 발전소의 안전 해석 및 과도 현상의 모의화 소프트웨어 기술 및 컴퓨

터 interface 설계 기술은 다른 원자력 발전소 시뮬레이터 개발에 활용할 수 있으며, 또한 원자력 발전소 비상 대응 설비의 개발에도 이용될 수 있다.

다. 생산 라인 제어용 시뮬레이터

현재 포항 제철에서 사용되고 있는 대부분의 시뮬레이터는 공정 개발 프로젝트를 수행한 결과의 부가 기능으로 개발되어 시스템의 기능 확인을 위한 테스트용으로 사용되고 있다. 많은 시뮬레이터들이 생산 라인과 맞물여 있어면서 작동되도록 되어 있는데 가상으로 생산에 관련된 데이타를 발생시켜 처리하여 마치 실제 생산이 일어나는 것처럼 보이게 하는 것이 대부분이다. 시스템의 향상, 설계 등을 위해 독립적으로 사용되는 시뮬레이터를 발견하기는 매우 어렵고, 일단 사용자에게 인도하면 더이상 시뮬레이터는 사용되지 않는 경우가 대부분이다.

이러한 시뮬레이터 중의 하나로 1988 년 "포항제철 2 열연 P/C System 개발" 프로젝트 결과에서 나타나 있는 지원기능으로서의 시뮬레이터에 관해 간단히 설명한다.

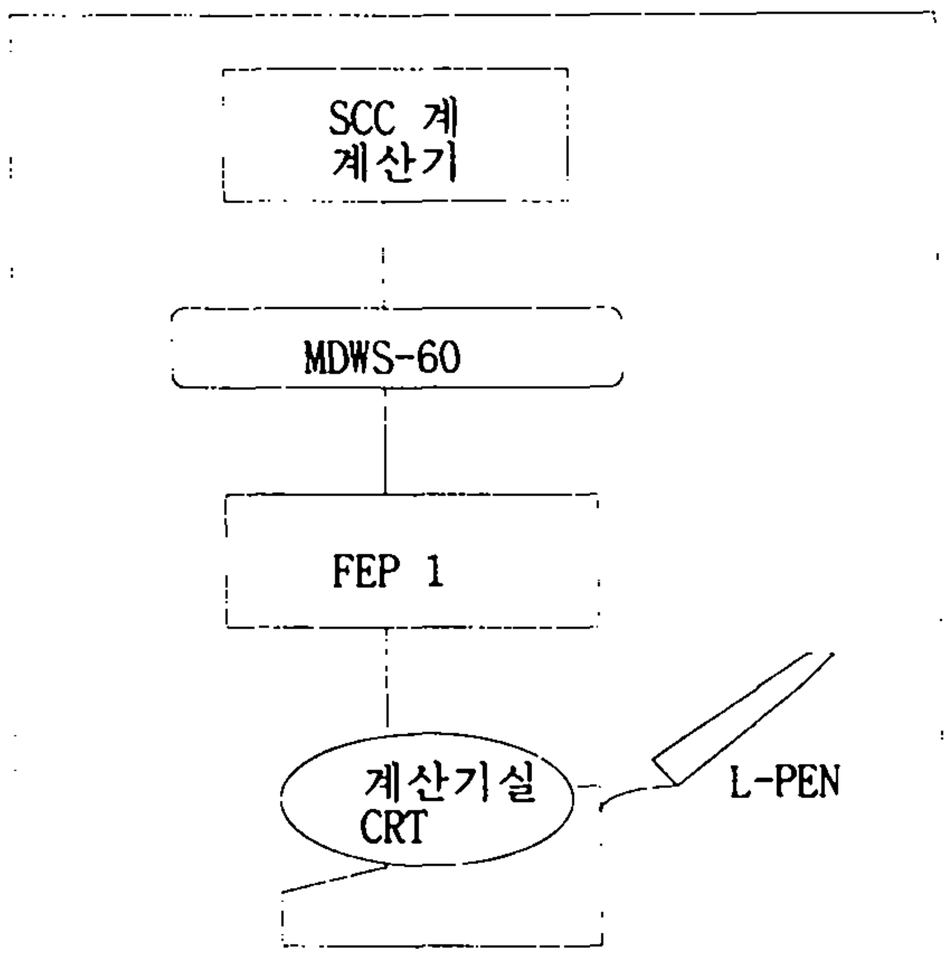
이 지원기능은 계산기 소프트웨어 조정을 주목적으로 한 지원 소프트웨어 도구이다. 그중, 계산기 단독으로 기능을 check 하는 TRACKING SIMULATOR, MELPLAC 의 COLD-RUN 에 대응하는 COLD-RUN SIMULATOR, 실제 조업할 경우 B-UP 계산기를 사용하는 ON-LINE SIMULATOR 로 구성되어 있다. 주목적은 계산기 소프트웨어 조정이지만, COLD-RUN 등과 같이 실제 조업에서 기계 설비의 조정 확인 등에 사용 가능한 것도 있다. 이 중 TRACKING SIMULATOR 와 COLD-RUN SIMULATOR 를 중심으로 살펴보기로 한다.

(1) TRACKING SIMULATOR

계산기 단독으로 MILL-LINE TRACKING 을 중심으로 진행하는 소프트웨어 기능 확인 지원 도구이다. 계산기실의 CRT 를 사용하여 라인의 개요도를 나타내고, 외부 I/O

를 받아들이지 않고, L-PEN 을 사용하여 TRACKING 용의 각종 SENSOR EVENT 를 발생시킨다. 이 도구는 센서 신호에 대응한 재료의 유무 등은 모두 체크하고 있지는 않다.

하드웨어 구성은 다음과 같이 이루어져 있다.



[그림 3-9] Tracking Simulator 의 하드웨어 구성

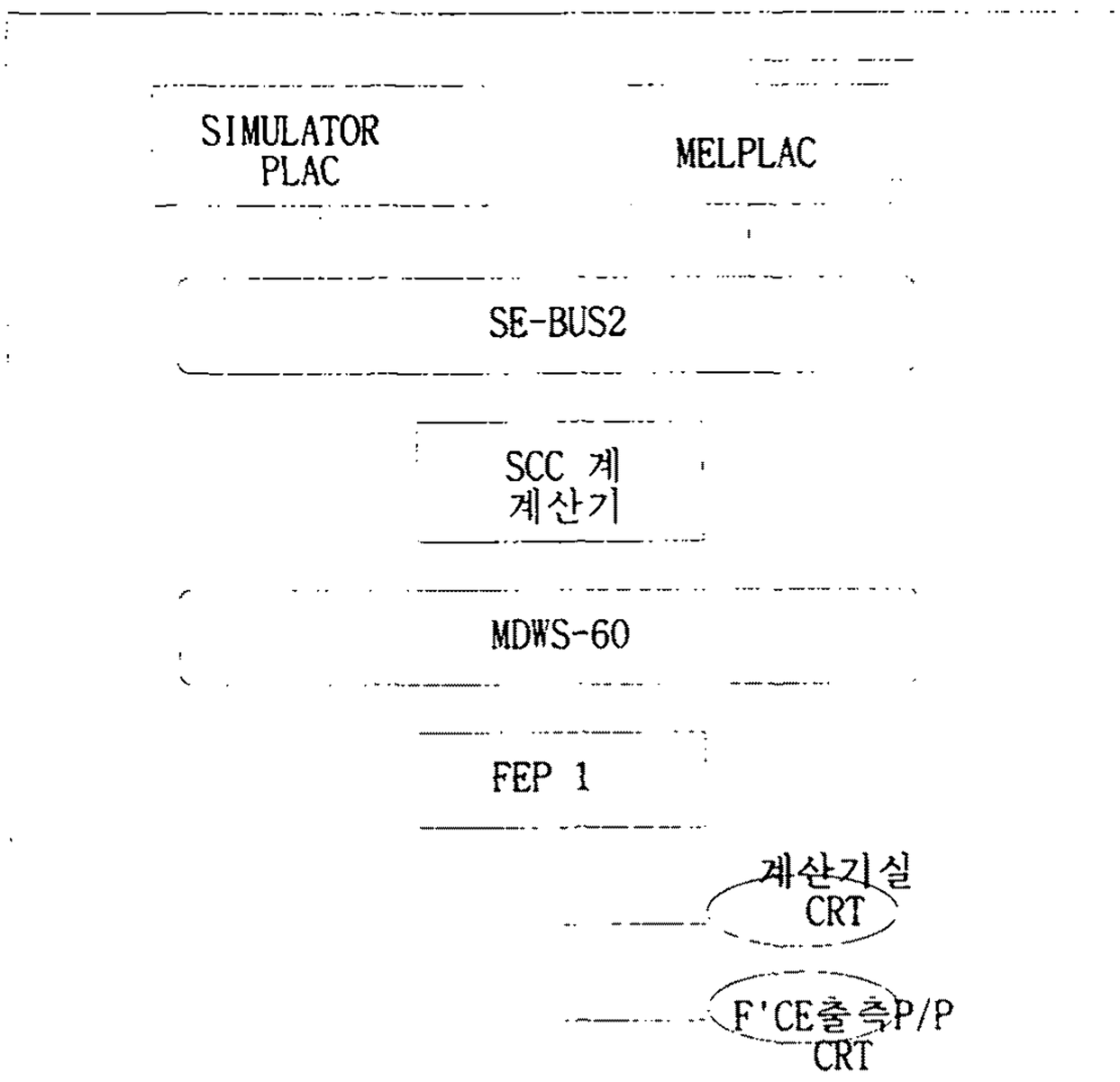
(2) COLD-RUN SIMULATOR

MELPLAC 의 COLD-RUN 에 대응하여 SCC 계 계산기에서 SIMULATION 환경을 설정하기 위한 지원도구이다. 시뮬레이션 환경의 설정은 시뮬레이션용의 테스트 데이타를 준비하는 것 뿐이다. 시뮬레이션용의 테스트 데이타는 실제 압연 명령 또는 테스트용으로 준비된 데이타 중에서 사용한다.

COLD-RUN SIMULATOR 는 SIMULATOR MELPLAC 에 의해 각종 TRACKING SENSOR 신호가 모의되기 때문에 SCC 계 계산기에서는 MILL-LINE 의 제어 처리는 통

상의 압연처리와 같이 처리한다. 단 압연재가 실제로는 없고 모의재가 있다는 것을 주의해야 한다.

하드웨어의 구성은 다음과 같다.



[그림 3-10] Cold-run Simulator 의 하드웨어 구성

이 시뮬레이터를 수행하면 각종 tracking sensor 신호가 모의된다. SCC 계 계산기에서는 통상 압연과 같이 처리를 하지만 FCC 계 계산기는 추출 신호만을 가지고 가상적으로 수행하게 된다.

3. 국내 상용 시뮬레이션 소프트웨어의 현황

가. 상용 시뮬레이션 소프트웨어의 개괄

시뮬레이션 시스템은 크게 시뮬레이션 언어(Simulation languages)와 시뮬레이터

(Simulators)로 구분하여 생각할 수 있다. 시뮬레이션 언어는 하나의 컴퓨터 패키지로 일반적인 용도로 사용되는 것이지만 특정 응용 분야에도 쉽게 적용하여 사용할 수 있도록 여러가지 특수한 구성 요소를 가지고 있는 경우가 많다. 예를 들면, SIMAN 과 SLAM II 와 같은 시뮬레이션 언어는 일반적인 응용 분야에서 사용되지만 특히 제조 과정을 모형화하기 위해 컨베이어, AGV(automatic guided vehicle) 등을 위한 constructors 을 가지고 있다. 시뮬레이션 언어는 프로그래밍 기술이 요구되고 복잡한 시스템을 모형화하기 위해서는 많은 노력이 든다.

시뮬레이터는 특수한 분야에만 사용할 수 있도록 특별히 제작된 컴퓨터 패키지로 해당 응용 분야에 특수한 기능들을 쉽게 선택적으로 명세하여 사용할 수 있다. 예를 들어 특수한 유형의 생산 시스템, 컴퓨터 시스템, 통신 시스템, 또는 발전소 시스템 등 특수 목적에 적합하게 주문제작된 것이 많다. 대부분의 시뮬레이터는 대상 시스템의 구성 요소에 관련된 modeling constructor 를 가지고 있기 때문에 프로그래밍 경험이 없는 사람도 쉽게 사용할 수 있다. 그러나 범용성이 떨어지고 표준화된 특성만을 모형화할 수 있다는 단점이 있다.

본 상용 시뮬레이션 소프트웨어에 관한 조사에서는 시뮬레이션 언어와 시뮬레이터를 구분하지 않지만 대부분의 상용 소프트웨어는 그 특성상 시뮬레이션 언어에 가깝다고 할 수 있다.

국내에서 사용되고 있는 대부분의 상용 시뮬레이션 소프트웨어는 외국에서 만들어진 것으로 국내에서 제작된 것은 거의 없다. 현재 국내에서 개발 중인 것으로는 통산 산업부의 G7 과제로 FMS, 물류 등에 응용가능한 시뮬레이터가 있다.

지금까지 많은 시스템들이 개발되어 왔으며 1993 년 12 월 OR/MS Today 에 실린 조사에서는 55 개의 제품을 서로 비교하여 소개하고 있다.

시뮬레이션 소프트웨어는 여러가지 기준을 가지고 분류할 수 있다. 응용 분야에 따

라 구별하거나 시뮬레이션이 수행되는 방식에 따라 구별할 수도 있다. 그러나 시뮬레이션 언어는 범용적으로 사용되기 위해 개발된 것이 많아 응용 분야에 따른 분류나 수행 방식에 따른 분류를 명확히 하기는 어렵다.

시뮬레이션 언어를 시뮬레이션이 수행되는 방식에 따라 분류하면 연속형, 이산 사건형, 그리고 혼합형으로 구분할 수 있다. 연속형 시뮬레이션 언어는 시스템이 연속적으로 변화하는 상황을 모형화하여 시뮬레이션 하는 것으로 주로 미분 방정식을 많이 이용한다. ASCL 및 CSSL-IV 는 미분 방정식의 기술 및 수치 계산 방법으로 해를 구하는 과정을 용이하게 하기 위해 개발된 것이다. 이산 사건 시뮬레이션 언어는 대부분 사건 중심 또는 프로세스 중심으로 문제를 기술하는데 GPSS, SIMSCRIPT, SLAM 등은 고수준의 시뮬레이션 언어들로서 모형을 쉽게 작성할 수 있다는 장점이 있다. 그 중 SLAM은 연속형 시뮬레이션도 가능하다.

나. 시뮬레이션 소프트웨어의 발전 경향

컴퓨터 기술이 발전됨에 따라 현재까지 개발되었거나 개발중인 시뮬레이션 소프트웨어들은 사용자의 편의를 증가시키고 문제 해결 능력을 향상시키기 위해 많은 노력을 기울이고 있다. 그 발전 경향들을 살펴보면 다음과 같다.

(1) 일반적인 특성

- 모형화의 유연성(modeling flexibility): 시뮬레이션 소프트웨어는 다향한 형태의 문제를 효과적이고 효율적으로 모형화하여 문제를 해결할 수 있도록 지원할 수 있어야 한다. 현재 많은 시뮬레이션 소프트웨어들이 객체 지향 기법(Object oriented approach)을 이용하여 개발되고 있다. 이 접근 방법은 모형 개발이 용이성 뿐 아니라 모형의 재사용도 쉽게 가능하게 하는 장점이 있다.
- 모형 개발의 용이성(ease of model development) : 사용자의 입장에서 볼 때 문제 해

결을 위해 작성해야 하는 모형을 쉽게 개발할 수 있도록 지원하여야 한다. 시뮬레이션 소프트웨어가 비록 뛰어난 모형화 능력을 가지고 있더라도 사용하기가 어렵다면 좋은 소프트웨어라고 할 수 없다. 사용자을 지원할 수 있는 다양한 기능을 가져야 한다. 이러한 경향은 소프트웨어들이 그래픽 처리를 지원하면서 많은 발전을 가져오고 있다.

- 모형 수행의 신속성(fast model execution speed) : 시뮬레이션은 실세계에서 발생하는 사건들을 컴퓨터에 모사하여 구현한 것이다. 따라서 실세계에서 복잡하게 진행되는 내용을 빠른 시간에 가상으로 실행할 수 있어야 한다. 이를 위해서는 컴퓨터의 속도가 빨라야 할 뿐 아니라 사건의 내용을 빠른 시간에 처리할 수 있는 모형화 방법론이 필요하다.
- 사용 환경의 변화 : 개인용 컴퓨터의 용량 및 기능이 급속도로 향상되면서 많은 시뮬레이션 소프트웨어들이 데스크탑 컴퓨터에서도 쉽게 운용될 수 있게 되었다. 이러한 경향은 예전에 많은 시뮬레이션 소프트웨어들이 용량 등의 문제 때문에 특정 하드웨어 환경에서만 작동되도록 만들어진 사실과 비교된다.

(2) 애니매이션

1980 년대 후반부터는 시뮬레이션 모형을 애니매이션화 하는 시도로 CINEMA, WITNESS, SIMFACTORY, ProModel 등의 많은 패키지들이 발표되었다. 이러한 접근 방법은 시뮬레이션 모형을 쉽게 작성할 수 있고 실행을 직접 보여줌으로써 실제 시뮬레이션 모형 및 프로그램의 타당성을 검정할 수 있다는 장점이 있다. 대부분의 시뮬레이션 소프트웨어는 그래픽 출력과 시뮬레이션 애니매이션을 기본으로 사양화하여 장착하고 있다. 기존에 애니매이션이 지원되지 않았던 시뮬레이션 소프트웨어들은 애니매이션 기능을 추가하여 새로운 시스템으로 기능을 강화하고 있다. 이러한 시스템의 예

를 들면, System Modeling 사의 ARENA 는 SIMAN 과 CINEMA 를 결합하여 모형화 환경을 제공하고 있다. 또 Advanced Systems Technologies 사의 visual interactive GPSS 와 GPSS/vi 도 그러한 예이다.

(3) 통합화 경향

문제가 복잡해짐에 따라 시뮬레이션은 단일 문제를 해결하기 위해 개발되기보다는 여러 단계에 걸쳐 사용되는 경우가 늘고 있는 것도 새로운 경향이라고 할 수 있다. 예를 들면 하나의 컴퓨터 통합 생산 시스템(CIM: Computer Intergrated manufacturing)에서 시뮬레이션의 사용은 설계, 생산, 품질 관리 등의 전 공정을 걸쳐 이루어지고 있다. 시뮬레이션 응용 가능 분야가 확장되고 있다는 사실은 시뮬레이션 소프트웨어가 시스템 설계용, 재무 모형, 데이타 베이스 등과 같은 다른 분야의 소프트웨어와 연결되어 사용될 수 있어야 한다는 것을 뜻한다.

(4) 분석 능력

시뮬레이션은 대부분 많은 종류의 데이타를 사용하고 있다. 그 데이타의 특성을 확인하여 모형을 개발하는데 정확히 반영하는 것은 매우 어려운 일이다. 일부 시뮬레이션 소프트웨어에서 이러한 분석 기능을 지원한다. SIMAN 의 Input Processor, SimStat, UniFit II 등의 프로그램들은 확률 분포를 확인하거나 매개변수를 확정짓는데 사용된다.

시뮬레이션은 또한 많은 출력 데이타를 발생시킨다. 시뮬레이션 출력을 분석하는 것은 통계 등의 전문지식을 필요로 하는 경우가 많다. AutoStat, SimStat 등은 steady-state 분석이라던가 통계 분석을 통하여 출력 데이타의 분석을 지원하고 있다.

다. 국내 딜러의 현황

국내에서 사용되고 있는 시뮬레이션 소프트웨어는 크게 학교 및 연구소에서 교육 및

연구 목적으로 사용되는 경우와 실제 문제 해결에 사용되는 경우로 구분하여 생각할 수 있다. 공장 자동화 등의 문제에 많이 이용되고 있는 상용 시뮬레이션 소프트웨어들은 국내의 여러 딜러들을 통해 공급되는데 이러한 딜러들로는 동일 CIM, 아이오텍, SimTech, 삼성 데이타 시스템, 대우 정보 시스템 등이 있다. 다음 [표 2-]은 각 회사에서 취급하고 있는 시뮬레이션 소프트웨어들 중 대표적인 것을 간단히 기술하고 있다

국내의 시뮬레이션 시장은 그 규모로 볼 때 성숙되어 있지 않고 매우 작기 때문에 많은 상용 시뮬레이션 소프트웨어들이 사용되고 있지는 않다. 다음의 [표 3-4]에서 알 수 있는 바와 같이 다양한 종류의 시뮬레이션 소프트웨어를 취급하고 있지는 않고 주로 생산 시스템 및 물류에 관련된 소프트웨어에 중점이 주어져 있다.

[표 3-4] 국내 딜러의 현황 및 소프트웨어

회사	취급 시뮬레이터	용도	사용환경
동일 CIM	Factor/ AIM	생산일정계획 설비투자 및 공장 layout 분석 생산성 분석 및 평가 공장 자동화 사전 분석, 평가 생산 일정 계획	PC, WS
	FI-2	생산계획, Shop Floor Control	WS
	UniFitII	통계 분석	PC
	SLAMII	범용 시스템	PC, WS
아이오텍	AutoMod	공장자동화, 배송 시스템 물류 시스템	PC,WS
SimTech	PCModel	생산, 물류, 수송 시스템	PC

	CADmotion	생산, CAD/CAM	PC
한국 CADD 엔지니어링	Quest	범용	PC,WS
	I-Grib	용접, 가공 simulation down load	전용 hardware
삼성데이타 시스템	ARENA Simulation System	생산, logistics, 일정계획, Process reengineering	PC, WS
대우정보시스템	AutoMod	공장자동화, 배송 시스템 물류 시스템	PC, WS
	FACTOR/AIM	생산일정계획 설비투자 및 공장 layout 분석 생산성 분석 및 평가 공장 자동화 사전 분석, 평가 생산 일정 계획	PC, WS

위에서 언급한 국내의 딜러들은 단순히 시뮬레이션 소프트웨어를 판매하기보다는 종합적인 시뮬레이션 전문 사업을 수행하면서 소프트웨어 판매도 겸하고 있는 경우가 많다. 즉 시뮬레이션에 관한 자문 , 교육 등의 기능도 함께 수행하고 있다.

라. 대표적인 상용 시뮬레이션 시스템들

다음은 대표적인 상용 시뮬레이션을 그 응용 분야에 따라 간단히 조사한 것이다. 국내에서 가장 많이 사용되고 있는 생산 분야의 소프트웨어와 요즘 컴퓨터 통신망 기술이 발전함에 따라 망의 성능 평가를 위한 시뮬레이션 소프트웨어를 살펴본다.

(1) 생산시스템을 위한 시뮬레이션 모델링 도구

생산 시스템을 시뮬레이션 하기 위한 소프트웨어는 대부분의 일반적인 시뮬레이션의 기본 기능에 추가하여 생산 시스템 응용 분야에 적용할 수 있도록 여러가지 constructor 들을 도입한 것들이 많다. 자동 생산 시스템을 특징짓는 AGV, 컨베이어, automatic storage and retrieval systems(AS/RS), 크레인, 로보트 등을 쉽게 모형화할 수 있는 기능이 그러한 것이다. 대표적인 것으로 SIMAN IV 와 SLAM II 는 생산 시스템에 적용 가능한 building block 을 가지고 있어 부품, 기계 위치, 사용 자원, 컨베이어, 운송수단 등을 쉽게 모형으로 작성할 수 있도록 지원한다.

(가) AutoMod II

AutoSimulation Inc.에 의해 개발된 제품으로 제조 및 물류 시스템을 시뮬레이션하는데 주로 사용된다. 개인용 컴퓨터 및 워크스테이션에서 사용할 수 있으며, 대화형, 3차원 애니매이션을 지원하여 제조 및 배송 시스템을 설계, 분석, 운용에 이용할 수 있다. 통계 분석 기능이 강화되었고 실험 계획, 디버깅, 보고서 작성, 스프레드쉬트 등과도 연결할 수 있다. 가격은 $15,000 이상이다.

(나) ProModel

PROMODEL Co.의 제품으로 에셈블리, 물류, JIT 등의 분야의 시뮬레이션에 사용된다. 초보자의 경우에도 사용하기 쉽게 되어 있으며 훌륭한 통계 처리 기능(입력 확률 분포 등)을 포함하고 있다. 개인용 컴퓨터에서 운용되며 도스용과 윈도우용이 있으며 윈도우용은 객체지향 기술을 이용하여 개발되었다. 가격은 도스용이 $4,950, 윈도우용은 $10,000-$13,000 이다.

(다) SIMFACTORY II.5

CACI 사의 제품으로 공장 및 생산 분석에 사용된다. 공장 설계를 연구하기 위해서는

별도의 프로그램이 필요하지 않으며 사용자 인터페이스를 제공하여 사용이 쉽다. 훌륭한 통계 처리 능력이 있고 소형, 중형 컴퓨터에서 수행된다. 가격은 $9,500 에서 $15,000 이다.

(라) WITNESS

AT&T ISTEL 사의 제품으로 제조, 공정, 서비스, 의료 등의 다양한 분야에 적용될 수 있다. 강력한 시뮬레이터로서 프로그래밍 같은 입력/출력 규칙과 동작을 지원하고 애니매이션을 보여준다. 개인용 및 위크스테이션에서 수행된다.

(마) XCELL+

Scientific Management System 의 제품으로 메뉴 방식으로 동작되는 그래픽 중심의 소프트웨어로 복잡한 제조 공정을 쉽게 모형화할 수 있다. 그러나 운송과 집적 컨베이어에 관한 모형이 부족하여 적용이 적절하지 않은 분야가 있다. 개인용 컴퓨터에서 수행되며 가격은 $1,500 으로 비교적 저렴하다.

(2) 통신망 시뮬레이션 모델링 도구

컴퓨터 및 통신망의 기술 발달과 더불어 LAN, WAN, B-ISDN 등이 나타나게 되었고, 그에 따라 통신망의 성능을 평가할 필요성이 부각되었다. 통신망 관련 시뮬레이션 소프트웨어는 통신망의 기능을 쉽게 모형화할 수 있게 도와 주어 시뮬레이션의 기능을 성공적으로 수행할 수 있도록 한다.

(가) NETWORK II.5

CACI 사의 제품으로 컴퓨터 통신망과 내장 컴퓨터 시스템(embedded computer system)의 시뮬레이션에 사용된다. 애니매이션이 지원되며 개인용 컴퓨터와 워크스테이션에

서 사용할 수 있다. 가격은 $23,500-$29,500 이다.

(나) OPNET(OPtimized Network Engineering Tools)

미국의 MIL3 Inc.에서 개발된 제품으로 통신망 및 분산 시스템을 시뮬레이션할 수 있다. 상세한 프로토콜 모델링과 성능 분석 기능을 갖추고 있고 애니매이션과 그래픽 객체 지향 에디터를 지원한다. 워크스테이션에서 사용되며 가격은 $14,000 이다.

(다) BoNes(Block Oriented Network Simulator)

미국 Comdisco 사에서 개발되어 계층적, 블럭 중심의 모형화를 통해 통신망과 분산 처리 시스템의 성능분석에 사용된다. 즉, 블럭을 통한 시스템의 모듈화 및 계층화가 지원되고 그래픽을 통해 입력된 모형을 자동으로 실행 가능한 C 프로그램으로 변환시킨다. 주요 적용 분야로는 LAN, PSDN, 패킷 통신망 등이 있다.

(라) COMNET III

미국 CACI 사에서 개발한 객체 지향 통신망 시뮬레이션을 위한 시스템으로 LAN, MAN, WAN 을 분석할 수 있으며, ATM, X25, ISDN, SS7, SNA, Frame Relay 등의 최신 기술의 분석도 제공한다. 애니매이션이 지원되며 개인용 컴퓨터와 워크스테이션에서 수행된다. 가격은 $23,500-$29,500 이다.

(3) 기타 분야

생산과 통신 이외의 분야로 시뮬레이션 소프트웨어가 적용되는 분야를 살펴보면 비행, 생물학, 화학 공정, 의료, 경영, 군사, 교육 분야를 생각할 수 있다. 분야에 따라 연속적인 시뮬레이션 소프트웨어가 필요한 곳도 있고 비지니스 리엔지니어링과 같은 특수한 목적을 위한 시스템들도 있다.

마. 국내 시뮬레이션 사용 환경의 문제점

시뮬레이션 소프트웨어가 국내에 많이 도입되지 못한 이유는 우선 시뮬레이션 소프트웨어의 가격이 높다는데 그 이유가 있다. 많은 시뮬레이션 소프트웨어는 가격이 수천만원에 이르는 것이 많고, 하드웨어를 포함하게 되면 억대에 이르는 것도 있어 대기업이 아니면 구입하기가 매우 힘들다.

사용자가 시뮬레이션을 대하는 마음 자세도 시뮬레이션 기술이 널리 확산되지 못하는 하나의 장애 요인이 된다. 즉 많은 사용자가 시뮬레이션 소프트웨어만 구입하면 문제가 해결될 것이라고 기대하지만, 시뮬레이션의 개념, 모델 정립 등의 전문가를 양성하기 위한 교육이 필수적이어서 단시간에 그 효과를 얻지는 못한다. 사용자는 시스템 특성을 모두 고려하여 시뮬레이션을 적용할 수 있는 능력이 사전에 배양되어야 한다. 따라서, 도입자에게 있어서 시뮬레이션 도구의 구입은 기술확보 과정의 시작 단계에 불과하며, 지속적인 적용 기술의 축적이 필요하다는 것을 충분히 인식하고 있어야 한다.

또한 시뮬레이션 소프트웨어는 대부분 외국에서 개발되었기 때문에 국내 실정에 맞지 않는 경우가 많다. 공정의 표준화와 자동화 정도, 엔지니어나 작업자의 시스템적 사고 수준, 그리고 시스템 운영 철학 등 산업 문화의 기반 구성 요인을 고려할 때 외국의 소프트웨어를 국내 환경에 그대로 적용하는 것은 무리인 경우가 있다.

4 절 실시간 시뮬레이션 모델

1. 실시간 객체 지향 모델

가. 개요

객체 지향 구조의 개발 초기부터 실시간 응용에 관심을 가진 많은 연구가들이 기본

적인 객체 모델에 마감 시간(deadline)을 도입하려는 생각을 가지고 있었다. 그러나 시스템 개발자들이 간단히 확장한 구조적인 접근방법으로 경성(hard) 실시간 분산 컴퓨터 시스템을 개발하는 것에 있어서는 의미있는 진전을 보지 못하였다. 경성 실시간 분산 컴퓨터 시스템을 개발하는데 있어서 가장 중요하고 바람직한 것은 시스템에서 적시에 서비스하는 능력이 시스템을 개발하는데 있어서 포괄적이며 실제적으로 쉽게 달성될 수 있게 하는 것이다.

기본적인 객체 모델의 적절한 확장을 발견하려는 동기는 실시간 분산 컴퓨터 시스템을 개발하는데 있어서 통합적인 방법론을 정립하기 위한 필요성에서부터 발생되었다. 실시간 시스템을 개발하는데 있어서 기존의 방법은 다음과 같은 문제점이 있었다.

(1) 요구명세의 엄격성 부족, 특히 시간적 행위의 요구사항 명세 및 신뢰성 요구사항

(2) 설계, 구현, 확인 및 평가시 명세와 시스템 모델간의 추적성 부족

(3) 신뢰성 및 시스템 응답 지연의 보장 결여로 설계 기법들의 통합 불능

통합 개발 방법론에서 한가지 간과했던 점은 실시간 컴퓨터 시스템을 설계, 추상화 및 단계적 개선 방법 및 응용 환경 해석의 기초를 제시하는 시스템 모델이다. 이 모델은 응용 환경의 표현뿐만 아니라 시뮬레이션도 제어 컴퓨터 시스템을 확인하는 통합적 도구로 사용된다. 바람직한 모델은 시스템 개발 주기동안 진화하는 다른 수준들에서 응용 환경, 환경 시뮬레이터, 제어 컴퓨터 시스템 설계?등을 표현하는데 있어서 일관성(uniformity)과 높은 정확성(accuracy)을 지원하는 것이다.

RTO.k 모델은 시간 구동 실시간 객체(Time-Triggerd Real-Time Object, TT-RTO)로도 불리는데 이는 Kane Kim 과 Hermann Kopetz 가 임베디드 컴퓨터 시스템 및 응용 환경을 일관성있고 정확하게 표현할 수 있는 기본적인 객체 모델을 찾으려는 시도의 결과 산물이다. 객체의 적시성(timeliness) 서비스 능력의 설계 시각 보증(Design-Time Guarantee) 구현의 용이성은 RTO.k 객체 모델을 구체화하는데 기본적인 원리로 사용되었다. 기본

적인 객체 모델의 다른 확장과 가장 특징적으로 구분되는 두가지는 다음과 같다.

(1) 자발적 메소드 (spontaneous method)로 불리는 시간 구동 객체 메소드와 서비스 메소드로 불리는 메시지 구동 객체 메소드간의 명확한 분리

(2) 기본 동시성 제한조건(basic concurrency constraint)이라 불리는 실행 규칙

RTO.k 객체를 기반으로 하는 일관성 있는 구조적 접근방법하에서는 제어 시스템 설계와 환경 시뮬레이터의 조합은 RTO.k 객체들의 네트워크(network) 형태를 갖는다.

나. RTO.k 모델의 주요 특성

다음과 같은 두가지 목표는 현재의 RTO.k 객체 모델을 형식화하는데 기본적인 원리로 채택되었다.

(1) 실시간 분산 컴퓨터 시스템(Distributed Computing System, DCS) 및 그의 응용 환경 시뮬레이터의 일관성 있는 구성

(2) 객체의 적시성 서비스 능력의 설계 시각 보증의 구현

(1) 기본적인 객체 모델의 주요 확장

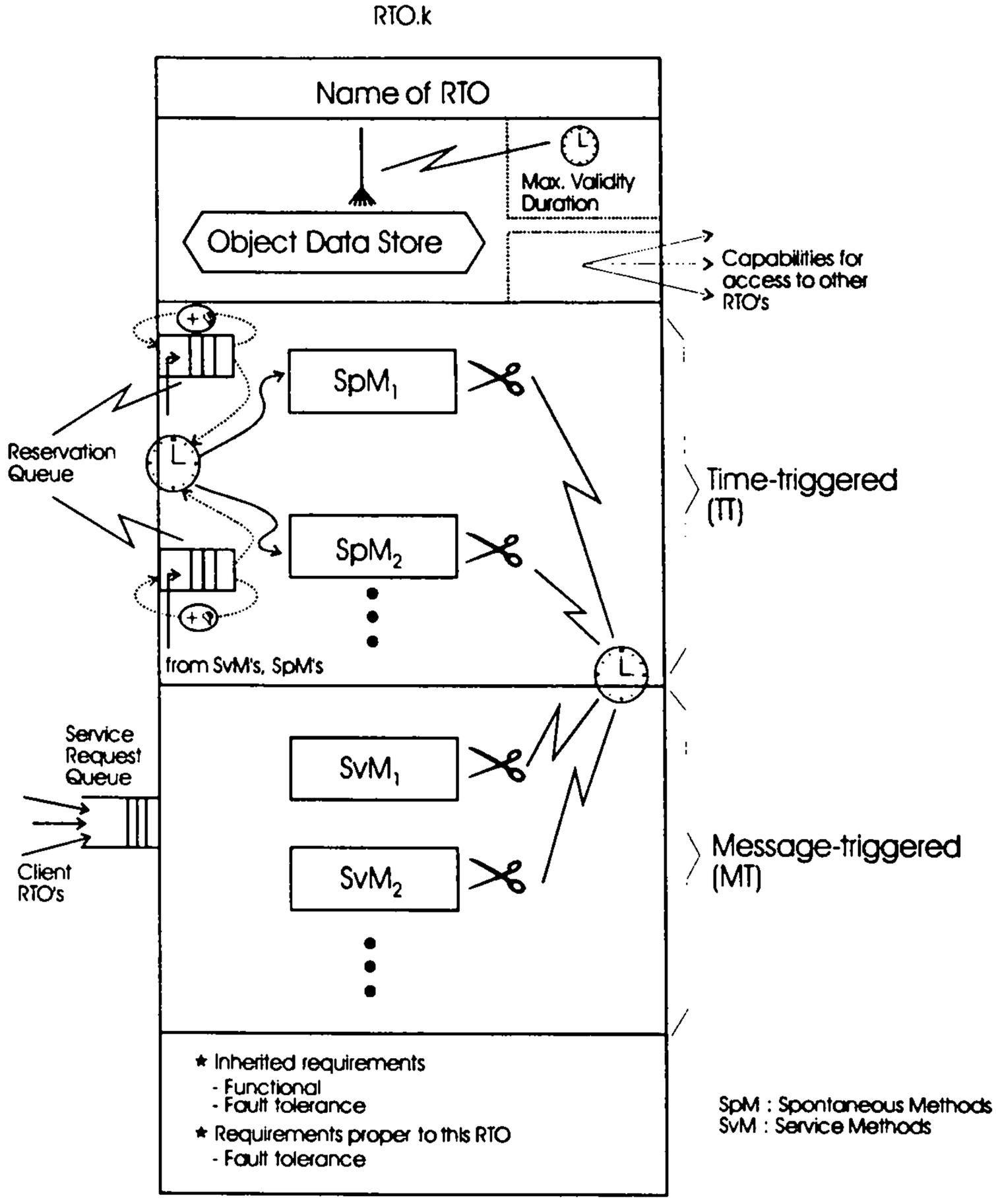
고전적인 객체 모델의 확장으로서 RTO.k 객체 모델은

(1) 프로그램을 하거나 객체 설계를 규정하는데 사용되는 언어와 종속되지 않고

(2) 상속(inheritance)이 구현된 방법에 종속되지 않는다.

현재의 RTO.k 객체 모델에 기반한 다수의 구체적인 언어 구조의 출현을 상상할 수 있으므로 이 모델은 발전하는 모델군에 대한 틀로서 볼 수 있다.

RTO.k 객체 모델의 기본 구조는 [그림 2-1]에 보여 진다. RTO.k 객체 모델은 다음과 같은 네가지 중요한 방법에 있어서 고전적인 객체 모델의 확장이다.



[그림 2-1] RTO.k 객체 모델의 구조

(1) RTO.k 객체의 일부 메소드에 대하여 실시간 클럭은 클럭이 설계 시각에 규정된 어떤 값에 도달할 때 메소드 실행을 구동하는 장치가 되며, 이러한 메소드를 시간 구동(TT) 메소드 또는 자발적 메소드(SpM's)라고 하며 클라이언트(client)로 부터 메시지

(message)에 의해 구동되는 고전적인 서비스 메소드(SvM's)와 명확히 구분된다. 실시간 클럭이 설계 시간에 결정될 수 있는 어떤 값에 도달할 때 취해지는 행동은 SpM 에만 나타난다.

(2) TT-메소드와 메시지 구동 메소드간의 충돌을 방지하는 동시성 제한조건이 포함된다. 기본적으로 외부 클라이언트로 부터 발생되는 메시지에 의해 구동되는 서비스 메소드는 충돌 가능한 TT 메소드의 수행이 일어나지 않을 때에만 활동할 수 있다. 정확하게 표현하면 메시지 구동 서비스 메소드가 TT 메소드와 객체 데이터 공간(Object Data Space, ODS)의 동일한 부분을 접근하는데 충돌이 배제되지 않는다면 메시지 구동 메소드는 자발적 메소드의 TT 구동에 대해 규정된 시간 영역내에서는 허용되지 않는다. 이러한 제한조건을 기본 동시성 제한조건(BCC)이라고 부른다. 따라서 TT 메소드는 메시지 구동 메시지에 비해 더 높은 우선순위가 부여된다. 이러한 BCC 는 TT 메소드간의 동시적 수행이나 메시지 구동 메소드간의 동시적 수행에는 어떠한 제한조건도 부과되지 않는다는 것에 주의해야 한다.

(3) RTO.k 객체의 각각의 메소드 수행에 대하여 마감시간이 부과된다.

(4) RTO.k 객체내에 포함된 실시간 데이터는 최대 유효 기간(Maximum Validity Duration)이라 불리는 시간간격 이후에는 유효성을 상실한다.

[그림 2-1]에서 객체 데이터 공간 (객체 데이터 저장소라고도 불림)상에서 작동되는 두개의 구별되는 유형의 메소드 즉, SpM 과 SvM 이 표시되어 있다. SvM 으로 부터 SpM 을 명확하게 분리한 것은 RTO.k 객체 모델의 중요한 특성이다. 두 유형의 메소드는 그들의 수행이 구동되는 방법에 있어서 뿐만아니라 "at constant-clock-value do S" 또는 "sleep-until constant-clock-value"와 같은 유형의 활동이 SpM 에서만 나타난다는 점에서 구별된다. 예를 들어 SvM 에 의해 계산된 값이 어떤 시간에 정확하게 출력될 필요가 있다면 RTO.k 객체는 생산자(producer) SvM 에 의해 제기된 요구가 SpM 에 전달되어서

출력을 수행하는 방식으로 설계된다.

(2) 자율적 활동 조건

SpM 에 대한 구동시간은 설계시각 동안에 상수로 완전하게 규정되어야 한다. 이러한 실시간 상수는 자율 활동 조건(Autonomous Activation Aondition, AAC) 섹션이라고 불리는 SpM 명세의 첫번째절에 나타난다. AAC 는 다음과 같은 형태로 규정된다.

```
ab        "AAC-begin"
{ [AAC name:]
    for <time-var> = from <activation-time>
                to <deactivation-time>
                [every <period>]
    start-during
            (<earliest-start-time>, <latest-start-time>)
    finish by <deadline>
}*
ae        "AAC-end"
```

여기서 별 표시 x* 또는 {x}*는 집합 {NULL, x, xx, xxx, ...}에 대한 정규 표현(Regular Expression)이다.

예를 들면 다음과 같은 경우를 생각할 수 있다.

```
ab
. AAC-1:
    for T = from 10:00:00.000 am
```

to 11:00:00.000 am

every 0.005 sec

start-during (T, T+0.001 sec)

finish-by T+0.003 sec

ae

위의 AAC-1 은 다음을 규정한다. "이 SpM 은 10 am 에 시작하여 11 am 까지 매 5 msec 마다 수행되고 각각의 수행은 1 millisecond 간격 (T, T + 0.001 sec)내에서 시작되어야 하고 T + 0.003 sec 내에 완료되어야 한다.

"for t = from 10am to 10am start-during (t, t + 5 min)" 는 "start-during (10am, 10:05am)"과 동일한 효과를 가진다는 것에 주의하라. AAC 섹션은 이름이 없거나 고유한 이름을 가지는 다수의 AAC 절(clause)을 가질 수 있다.

실행 엔진은 SpM 의 AAC 를 사용하여 적절한 시간에 SpM 을 구동하는 구동 스케쥴을 생성하며 이는 SpM 의 예약 큐(Reservation Queue) ([그림 2-1]에 표시됨)에 삽입된다. 각 예약 큐는 가장 최근에 구동될 스케쥴이 큐의 맨 앞에 나오게하는 저장 큐이다.

SpM 의 AAC 섹션이 실제 구동시간이 아닌 구동시간의 후보만을 갖게할 수도 있다. 따라서 AAC 섹션내에 정해진 후보 구동시간의 부분집합은 실제구동을 위하여 동적으로 선택될 수 있다. 이런 동적 선택은 동일한 RTO.k 객체내의 SvM 이 특정 SpM 의 미래의 실행을 요구할 때 발생한다. 다시 후보 구동 시간은 설계 시각동안에 상수로서 완전하게 규정되어야 한다. SvM 은 SpM 에 관련된 예약 큐에 예약함으로써 SpM 의 미래의 구동을 요구한다. 예약 큐는 현재까지 결정된 SpM 에 대한 미래에 수행할 구동 스케쥴을 갖는다. 컴파일러의 일을 간단히 하기 위해서 SpM 을 수행하기 위해 SvM 에 의한 예약은 SpM 의 정의에 나타나는 AAC 절의 이름을 포함하여야 한다. 실제 구동시간보다 후보 구동시간을 규정하는 AAC 절을 포함하는 AAC 섹션의 부분은 "if-

demanded" 선언으로 시작한다.

예를 들면

ab if-demanded

a1: start-during (10am, 10:05am)

finish-by 10:15am;

a2: start-during (11am, 11:05am)

finish-by 11:15am

ae

이것은 두개의 후보 구동 시간을 정의한다. SvM 은 다음과 같이 SpM 즉 SpM-1 을 호출한다.

if current-time < 9:30am

then request SpM-1(AAC=a1)

else request SpM-1(AAC=a2)

따라서 SpM 에 대한 구동 시간을 결정하는 두가지 모드가 존재한다.

(1) 시스템 설계과정에서 완전히 결정됨. 이런 경우 정적으로 스케쥴된다고 한다.

(2) SvM 이 SpM 의 수행을 요구하고 실제 구동시간으로서 설계시각동안 준비된 후보 구동시간의 부분집합을 결정함. 이런 경우 SpM 은 부분적으로 동적 스케쥴된다고 한다.

다른 한편 실시간 클럭이 설계시에 결정될 수 없는 값에 도달했을 때 취해지는 행동은 SvM 에 나타날 수 있는데 행동이 주기적으로 수행될 때 나타난다. 그러므로 SvM 은 "for X seconds every Y seconds do S"형태의 표현을 포함할 수 있다. 그러나 이 표현식 수행의 시작시간은 SvM 이 클라이언트에 의해 호출될 때까지 알 수 없다.

(3) RTO.k 객체간의 상호작용

RTO.k 객체 모델의 기반을 이루는 설계 개념은 실시간 DCS 는 항상 RTO.k 객체의 네트워크의 형태를 갖는다는 것이다. RTO.k 객체는 서버 객체내의 SvM 에 대한 클라이언트 객체의 호출(call)을 통해 상호작용한다. 호출자는 SpM 또는 클라이언트 객체내의 SvM 이 될 수 있다. 클라이언트와 서버 객체의 고도의 동시 연산(concurrent operation)을 가능하게 하기위하여 고전적인 블로킹 유형의 호출 뿐만 아니라 비블로킹(비동기로도 불림) 유형의 호출(서비스 요구)도 SvM 에 가해질 수 있다.

따라서 서버 RTO.k 객체내의 SvM 에 다음과 같은 두가지 기본적인 유형의 호출이 가해질 수 있다.

(1) 블로킹 호출(Blocking Call) : SvM 을 호출한 후 클라이언트는 SvM 으로 부터 되돌아 올 때까지 기다린다. 구문구조는 다음의 형태를 갖는다.

```
Obj-name.SvM-name(parameter-1, parameter-2, ...)
```

클라이언트와 서버 객체는 두개의 다른 처리 노드에 상주할 수 있기 때문에 이런 호출은 일반적으로 원격 프로시져 호출(Remote Procedure Call)의 형태로 구현된다. SvM 내에 결과 매개변수가 없다고 할지라도 실행 완료 신호(signal)는 클라이언트에 되돌아 온다.

(2) 비블로킹 호출(Non-Blocking Call) : SvM 을 호출한 후 클라이언트는 다음 스텝(표현식과 명령들)으로 진행할 수 있고 그리고 SvM 으로 부터 결과 메시지를 기다린다. 구문구조는 다음의 형태를 갖는다.

```
Obj-name.SvM-name(parameter-1, parameter-2, ..., mode NWFR, timestamp TS);
---- statements ----
```

get-result Obj-name.SvM-name(TS) by deadline;

모드 명세 "NWFR"은 "No-Wait-For-Return" 의 약자로서 이것은 비블로킹 호출이라는 것을 나타낸다. 클라이언트가 SvM 을 호출할 때 클라이언트는 TS 라는 변수에 타임스탬프를 기록한다. 타임스탬프는 클라이언트로 부터 동일한 SvM 에 대한 다른(과거 또는 미래) 호출과 구분되는 SvM 에 대한 이 특정 호출을 고유하게 인식한다. 따라서 후에 클라이언트가 "get-result" 표현식을 수행하여 SvM 에 대한 앞서의 비블로킹 호출로부터 되돌아온 결과의 도착을 확인할 필요가 있을 때, SvM 이름 뿐만 아니라 종속된 호출과 관련된 타임스탬프를 포함하는 TS 변수가 지정되어야 한다. 클라이언트가 "get-result" 표현식을 수행하기 전에 SvM 에 대한 다수의 비블로킹 호출을 발생시킬 때 타임스탬프는 비블로킹 호출이 참조되는 실행 엔진을 명확하게 지정한다. 결과가 "get-result" 표현식을 수행할 때 돌아오지 않는다면 클라이언트는 실행 엔진이 결과의 도착을 인식할 때까지 기다린다. 이와 같이 비블로킹 호출은 대응되는 "get-result" 표현식이 수행될 때까지 지속되는 클라이언트(SpM 또는 SvM)와 서버 SvM 간의 동시성을 생성한다. 어떤 상황에서는 클라이언트는 SvM 에 대한 비블로킹 호출로 부터의 어떠한 결과도 필요로 하지 않는다. 이러한 클라이언트는 "get-result" 표현식을 사용하지 않는다.

(4) 동시성과 실행 ODS 명세

다음과 같은 유형의 동시성(concurrency)은 RTO.k 객체내의 객체 메소드를 실행하는데 사용된다.

(1) SpM 실행자간의 동시성

(2) SvM 실행자간의 동시성

(3) SpM 실행자와 SvM 실행자간의 동시성

SpM 간의 동시성은 묵시적으로 그러나 자연스럽게 규정된다. 예를 들면 두 SpM 은

10am 에 구동 되도록 설계된다. 일반적으로 두 SpM 의 규정된 실행주기가 중복된다면 두 SpM 명세는 명확하게 동시성이 내포되어 있다.

유형 2 와 유형 3 의 동시성을 이용하기 위해서 RTO.k 객체 모델내에서 채택되는 접근 방법은 각각의 메소드에 의해 사용되는 객체 데이터 공간(ODS)의 부분을 명백히 선언하는 것과 데이터의 충돌이 없을 때 마다 메소드의 동시적 수행을 허용하는 것이다. 따라서 RTO.k 클래스(class)내의 ODS 섹션은 ODS 세그먼트(ODSS)의 선언들로 구성된다. 각 ODS 세그먼트는 명확하게 명명된다. 그러므로 각 객체 메소드의 명세는 메소드의 실행중에 동작할 수 있는 ODS 세그먼트들의 집합을 지정하는 실행 ODS 세그먼트를 포함한다. 이 명세는 각각의 ODS 세그먼트가 “read-only” 또는 “read-&-write” 목적으로 접근될 수 있는지도 지정한다. 이것은 실행될 필요가 있는 객체 메소드들 간의 가능한 데이터 충돌을 감지할 수 있게 한다. 이와 같이 각 ODS 세그먼트는 설계자에 의해 명확하게 규정되는 저장 단위이다. 이러한 ODS 세그먼트의 크기는 객체 메소드가 수행될 때 이용될 수 있는 병행성의 정도를 결정한다.

설계시에 실행 ODS 명세는 SpM 명세내의 오류 발견을 위해서 사용된다. 만약 두 SpM 이 동일한 ODS 세그먼트에 대하여 “read-&-write” 권리를 가지고 동시에 수행되도록 규정되었다면 SpM 명세는 옳지않은 것이다.

실행시에 실행 ODS 명세는 클라이언트에 의해 요구되는 SvM 이 즉시 구동될 수 있는지 또는 충돌가능한 어떤 메소드가 완료될 때까지 지연되는지를 결정하기 위하여 실행 엔진에 의하여 사용된다. 예를 들어 SvM, SvMx 의 실행 ODS 명세가 {(ODSS-1,RW)}로 구성되며 RW 와 RO 는 각각 “read-&-write” 권리 및 “read-only” 권리라고 가정하자. 클라이언트가 SvMx 를 호출하고 실행 엔진이 SvMx 실행 구동의 가능성 평가를 시작한다면 엔진은 현재 실행중인 어떠한 SpM 또는 SvM 이 ODSS-1 에 대한 접근권한(RO 또는 RW)을 갖는지를 검사한다. 접근권한을 갖는다면 SvMx 는 아직 구동될

수 없다. 그렇지 않다면 엔진은 ODSS-1 에 대한 접근권한을 갖는 어떠한 SpM 이 t + MET(SvMx)전에 구동될 것인지를 검사한다. 여기서 t 는 현재의 시간을 나타내고 MET(SvMx)는 SvMx 에 대한 최대 수행 시간 예측치를 나타낸다. 다시, 그렇다면 SvMx 는 아직 구동될 수 없다. 그렇지 않다면 SvMx 는 구동된다.

최악의 경우의 서비스 시간을 결정하기 위한 객체 설계자의 증가된 부담 비용을 가지면서 SvM 실행간의 추가된 동시성을 이용하려면 SvM 의 실행 ODS 세그먼트에 대한 접근 모드규정이 않될 수도 있고 대신 각 ODS 세그먼트는 CREW(concurrent-read-&-exclusive-write) 모니터내에 포함(package)될 수 있다. CREW 모니터는 readers-writers 의 미론(semantics)을 갖는다. 이런 경우 클라이언트에 의해 호출된 SvM 은 SpM 과의 데이터 충돌의 가능성이 없는 한 구동될 수 있다. 공유된 ODS 세그먼트는 CREW 모니터내에 안전하게 보호되기 때문에 다른 SvM 과의 데이터 충돌 가능성은 무시될 수 있다.

동일한 RTO.k 객체내의 세개의 객체 메소드가 다음과 같은 실행 ODS 명세를 갖는다고 가정하자.

Working ODS of SpM1: {(ODSS1, RO), (ODSS2, RW)}

Working ODS of SvM1: {(ODSS1, unspecified)}

Working ODS of SvM2: {(ODSS1, unspecified)}

또한 SvM1 은 클라이언트에 의해 호출되고 현재의 시간은 10am, SpM1 은 10:05am 에 구동되며 MET(SvM1)은 10 분이라고 가정한다. SvM1 은 이 시간에 구동될 수 없다. 왜냐하면 ODSS1 에 대한 SvM1 의 접근 모드는 규정되어 있지 않기 때문에 RW 로 될 수 있고 또한 SvM1 이 지금 구동된다면 SpM1 이 구동되는 시간인 10:05am 에 끝나지 않는 SvM1 의 수행 가능성이 있기 때문이다. SpM1 이 10:10am 에 완료된다고 가정하자. 이 시간에는 SvM1 이 구동된다. SvM2 가 10:12am 에 클라이언트로 부터 호출되고

SvM1 이 시간에 계속 수행 중 이라면 SvM2 와 SpM 간의 데이터 충돌의 가능성이 없는 한 SvM2 는 구동 될 수 있다. 이것은 ODSS1 이 CREW 모니터내에 포함되었기 때문이며 이와 같이 SvM1 과 SvM2 에 의해 조화롭게 공유될 수 밖에 없다.

마. 적시 서비스 능력의 설계 시각 보증 및 최대 서비스 시간의 명세

RTO.k 객체 모델의 중요한 기본적인 설계 개념은 실시간 DCS 는 항상 RTO.k 객체의 네트워크의 형태를 가지며 각각의 RTO 객체는 클라이언트 RTO.k 객체들에 신뢰성 있는 서비스를 제공하는 동안 자신의 활동을 시간 조정(timing)하는데 있어서 최대한의 자치성(autonomy)을 가져야 한다는 것이다. 그러므로 RTO.k 객체의 설계자는 SpM 을 내부적인 능력으로, SvM 을 모든 가능한 클라이언트에게 알려진 서비스로 볼 수 있다.

각각의 RTO.k 객체의 설계자는 가능한 클라이언트 객체의 설계자에게 알려진 SvM(그리고 관련되는 SpM)의 명세내에 각 SvM(그리고 SvM 으로 부터 요구되어 수행될 수 있는 각각의 SpM)에 의해 제공되는 모든 출력의 마감시간을 지정하여 객체의 적시성 서비스 능력의 보증을 제공한다. 여기에서의 출력은 다음이 될 수 있다.

(1) ODS 일부분의 수정

(2) 다른 RTO.k 객체 (클라이언트가 될 수도 되지않을 수도 있음) 또는 다수의 객체에 의해 공유되는 장치로의 메시지 전달

(3) 특정 SpM 에 대한 예약 큐에 예약을 등록

가능한 클라이언트 RTO.k 객체의 설계자에게 제공되는 각각의 SvM 의 명세는 적어도 다음을 포함해야 한다.

(1) 다음들로 구성되는 입력 명세

(가) 서버 객체가 받아 들일 수 있는 입력 매개변수의 유형

(나) 최대 요구 수용 비율 즉, 서버 객체가 클라이언트 객체로 부터 서비스 요구를

받을 수 있는 최대 비율

(2) 최대 지연(정확한 출력 시간은 아님) 및 SvM 에 의해 제공되는 모든 출력에 대한 출력값의 특성을 지정하는 출력 명세. SvM 으로 부터 기대되는 출력 행동에 관련되는 모든 최대 지연중의 최대치는 SvM 의 최대 서비스 시각이다.

클라이언트 객체로 부터의 서비스 요구가 서버 객체에 대한 입력 명세내에 지정되는 최대 수용 비율을 초과하여 서버 객체에 도달한다면 서버는 클라이언트 객체에 예외 신호를 돌려 보낼 수 있다. 시스템 설계자는 조심스럽게 설계하여 이런 "overflow"를 방지할 수 있다. 그렇지 않으면 예외 관리자가 응용의 목표를 만족스럽게 달성하는 것을 확신하는 전제하에 예외 관리자를 제공할 수 있다.

SvM 으로 부터의 출력에 대한 최대 지연의 명세는 서버 객체 설계자에서 부터 가능한 클라이언트 객체의 설계에 까지 중요한 역할이 된다. 이것은 적시성 서비스의 보증이 된다. 최대 지연 명세를 결정하기 전에 서버 객체 설계자는 객체 실행 엔진(하드웨어와 운영체제)이 사용가능하고 서버 객체는 출력 활동이 최대 지연내에서 수행될 수 있도록 SvM 을 항상 수행할 수 있도록 구현될 수 있어야 한다. 이것은 서버 객체 설계자가 다음을 고려해야 한다는 것을 의미한다.

(1) 서비스 요구의 클라이언트 객체에 의한 발생으로 부터 서버 객체에 의해 관련된 SvM 을 구동하기까지의 최악의 경우의 지연

(2) SvM 을 구동하는데서 부터 이들 각각의 출력 활동을 하기까지의 최악의 경우의 실행 시각

한편 클라이언트 RTO.k 객체는 서버 RTO.k 객체에게 모든 의도적인 계산 효과(모든 의도적인 출력 활동)를 생성하는데 대해 마감시간을 부과한다. 클라이언트에 의해 부과된 마감시간은 고유한 마감시간 보다 앞설 수 없고 서버 객체의 설계자에 의해 관련된 SvM 의 완료를 알려주어야 한다.

SvM 으로 부터의 요구에 의해 실행될 수 있는 SpM 의 명세는 또한 SvM 을 호출할 수 있는 클라이언트 객체의 설계자에게 제공되어야 한다. 이러한 SpM 의 명세는 적어도 다음을 포함하여야 한다.

(1) 자율 활동 조건(AAC)

(2) 출력 명세

SpM 명세내에 입력 명세는 없지만 SpM 에 대한 출력 명세는 SpM 의 수행으로 부터 기대되는 모든 출력에 대해 이것이 생성되는 정확한 시각과 출력 활동내에서 가져오는 모든 값의 특성을 지정한다. 예를 들면 SpM 은 다음과 같이 설계될 수 있다. "SpM 이 10:10am 과 10:15am 사이에 수행을 시작한다면 10:30am(출력의 시간)에 객체 데이터 저장소내의 특정 데이터 항목을 수정한다(출력의 특성)"

적시 서비스 능력의 설계 시간 보증을 쉽게하기 위해서 RTO.k 객체 모델에 기본 동시성 제한조건(basic concurrency constraint, BCC)이 포함되었다. 적어도 이것은 SpM 의 실행 시간 행위 분석을 쉽게 하여 준다. SpM 의 실행은 SvM 의 실행에 의해 방해 받지 않고 SpM 의 구동 시간은 설계시에 정해진다. 예를 들면 "at 10am do S"와 같은 유형의 표현식이 SpM 내에 나타났다면 이것의 신뢰성 있는 수행은 쉽게 보증된다. 따라서 대부분의 계산이 SpM 에 의해서 수행되고 단지 간단한 클라이언트 인터페이스 기능이 SvM 에 의해서 행해진다면 대부분의 계산이 SvM 에 의해서 수행되고 동일한 ODS 부분을 접근하기 위해 SvM 간에 잦은 경쟁이 일어나는 객체의 경우 보다 적시 서비스 능력을 보증하기가 쉽다.

(5) 최대 유효 기간

ODS 섹션은 ODS 세그먼트의 선언들로 구성되고 각 ODS 세그먼트 선언은 변수 선언들로 구성된다. 여기 최대 유효 기간(Maximun Validity Duration, MVD)들은 다음과 같

이 각 변수들과 관련될 수 있다.

<type-specifier> <identifier> [during <effective-period> | by <expiration-time>]

예를 들면

"int K during 5 msec"

이것은 정수형 변수 K로 저장된 새로운 값은 단지 5 millisecond만 유효하다는 것을 나타낸다. MVD 명세는 실행 엔진에 구현되어 있는 오류 검출자에 의해 사용될 수 있다.

(6) 객체 메소드 명세의 전체 구조

RTO.k 클래스 정의는 4개의 주요 섹션으로 구성되어 있다.

```
RTO_class = class
begin
    "connectivity_section :"
        list of RTO_name;
                "that can be called upon"
    "object_data_space_section :"
        list of object_data_space_segment;
"spontaneous_method_section :"
        list of spontaneous_method;
"service_method_section :"
```

list of service_method

end;

RTO.k 클래스내의 SpM 명세의 전체 구조는 다음과 같다.

SpM-name *<name>*

using-SpM *<SpM-name>**

using-data (*<ODS-segment-name>*, *<access-mode>*)*

<AAC section>

finish-by *<deadline>*

begin

<method-body>

end;

여기서 "using SpM" 섹션에는 이 SpM 에 의해 보내지는 활동 요구들을 받는 동일한 RTO.k 객체내의 SpM 의 이름들이 나열된다.

RTO.k 클래스내의 SvM 명세의 전체 구조는 다음과 같다.

SvM-name *<name>* (*<parameter specification>*)

using-services *<RTO-name.SvM-name>**

using-SpM *<SpM-name>**

using-data (*<ODS-segment-name>*, *<access-mode>*)*

[finish-within *<duration-limit>*|

finish-by *<deadline>*]

begin

<method-body>

end;

여기서 적시 완료 요구조건은 두가지 다른 방법으로 규정될 수 있으나 메소드 완료 마감시간 명세는 SvM 의 수행 시작 시간이 결정될 때까지 완전히 평가될 수 없는 표현이다.

(7) 동일한 객체내의 RTO.k 객체간 및 객체 메소드간의 확장된 모드 상호작용

SvM 에 대한 블로킹 호출과 SvM 에 대한 비블로킹 호출등의 RTO.k 객체간의 두가지 기본적인 유형의 상호작용에 부가하여 특정 상황에서 통신 오버헤드(Communication Overhead)를 줄이기 위하여 각각의 두가지 유형의 주요 변화가 RTO.k 객체 모델에 채택된다. 이러한 확장의 골간은 클라이언트가 SvM 을 호출하고 다른 SvM 으로 부터 결과를 받는 장치를 허용하는 것이다. 이것은 어떠한 SvM 의 수행도 대기(stand-by)를 기반으로 하게 하는 기본 동시성 제한조건을 통해서 가능한 것이다. 즉 SpM 수행들 간에 충분히 큰 시간창(Time Window)이 열려있을 때 수행하게 하는 것이다.

따라서 SvM 즉 SvMx 의 최대 수행 시간 예측치가 MET(x) second 이고 SpM 들이 빈번히 수행되어 MET(x) 보다 큰 시간창이 열릴 수 없다면 SvM 은 실행될 수 없다. 이 문제를 피할 수 있는 한 방법은 이런 SvM, SvMx 를 다수의 작은 SvM 들, SvMx1, SvMx2,...,SvMxk 로 나누는 것이다. 클라이언트는 동시에 각각의 작은 SvM 들을 호출하여야 한다. 각각의 더작은 SvM 을 호출하는 것은 클라이언트로 부터 SvM 으로 호출 메시지를 전달하며 SvM 으로 부터 클라이언트로 결과 메시지를 전달하는데 대한 통신 오버헤드가 발생한다. 일반적으로 이러한 객체간 메시지들은 객체내 메시지들 보다 훨씬 큰 지연을 발생 시킬 수 있다. 자연스럽게 다음과 같은 순서를 생각할 수 있다. 클라이언트는 첫번째 SvM(SvMx1)을 호출한다. 실행을 완료한 후 다음 차례로 클라이언트를 두번째 SvM(SvMx2)으로 넘겨준다. 이것은 마지막 SvM(SvMxk)이 완료될때 까지 계속되며 최초의 SvM 을 호출한 외부 클라이언트인 상속 클라이언트에게 결과를 돌려

준다. SvM 으로 부터 클라이언트를 다음 SvM 으로 전달하는 것은 내부 객체 메시지들과 관계되는데 이러한 클라이언트 전달 호출 메시지는 외부 클라이언트로 부터의 서비스 요구가 통과하는 것과 동일한 서비스 요구 큐를 통과한다.

클라이언트 전달 호출은 외부 클라이언트에 의한 호출의 경우 또는 ODS 내의 공유 데이터 구조를 통한 정보 전달등에서 행해진 것과 같은 명확한 방법으로 매개변수를 전달하는 것과 관련된다. SvM 에 대한 이러한 클라이언트 전달 호출의 문맥 구조는 다음 형태를 갖는다.

SvM-name(parameter1, parameter2, ..., mode CT, external parameter-x1, parameter-x2, ...)

모드 명세 "CT"는 "Client Transfer"의 약자이며 이는 클라이언트 전달 호출임을 나타낸다. "external"절내에 나열된 매개변수들은 호출자와 호출자의 호출자와의 인터페이스로 사용된다. 정상적으로 이러한 매개변수들은 외부 클라이언트(즉, 종속 서버 객체밖에 위치한)와 클라이언트에 의해 호출된 최초의 SvM 간의 인터페이스로 부터 상속된다.

SvM 에 대한 이 클라이언트 전달 호출을 실행하는 일환으로 실행 엔진은 호출자 SvM 을 종료 시키고 호출된 SvM 의 실행을 위한 요구를 서비스 요구 큐에 담고, 호출된 SvM 으로 부터 방금 끝난 호출자 SvM 으로의 반송 연결(return connection, 즉 클라이언트로의 연결)을 확립 시킨다. 호출된 SvM 의 "return" 표현식이 실행될 때 결과는 반송 연결이 확립되면서 돌려진다. 첫번째 SvM 을 호출한 외부 클라이언트는 어떤 SvM 으로 부터 결과를 받을지를 예측할 수 없으므로 어떠한 SvM 으로 부터라도 결과를 받을 수 있도록 구현되어야 한다.

이 클라이언트 전달 호출이 다른 RTO.k 객체의 SvM 을 호출하는 경우까지 확대될 수 없는 이유는 없다. 외부 SvM 을 호출하는 이러한 클라이언트 전달 호출의 문맥 구조는 다음과 같다.

Obj-name.SvM-name(parameter1, parameter2, ..., mode CT, external parameter-x1, parameter-x2, ...)

외부 SvM에 대한 위의 클라이언트 전달 호출외에 RTO.k 객체간의 확장 모드 상호 작용의 채택을 위한 다른 언어 구조를 도입할 필요는 없다. 이것은 호출된 SvM으로 부터 받은 결과는 시스템내의 어떠한 SvM으로 부터 받은 결과의 특수한 경우이기 때문이다. 더우기 클라이언트의 설계자와는 투명하게 서버 객체의 상세를 만드는 것이 우리의 철학이므로 외부 SvM을 호출하는 클라이언트는 호출된 SvM으로 부터 결과를 받는 경우와 다른 SvM으로부터 결과를 받는 경우 간의 구별을 할 필요가 없다.

2. 시뮬레이션 구조 모델

가.개요

RTO.k 객체 모델을 개발하는데 있어서 최종 목표는 실시간 제어 시스템 설계의 가변적인 추상화 뿐만 아니라 응용환경을 정확하게 표현하는데 효과적인 표현 구조를 확립하는 것이다. 이러한 모델 구조는 완전한 상세 기능 명세로 부터 상위 수준의 구조적 표현에 이르기 까지 가변적인 정확성 표현을 가능하게 하는데 있다. 특히 이것은 다양한 수준의 정확성으로 시간 특성을 다룰 수 있어야 한다. RTO.k 모델이 이러한 모델링 구조를 가지고 시뮬레이션 모델로서의 가능성을 시험하기 위하여 국방 C3 시스템에 RTO.k 모델을 이용하여 요구명세에서 부터 구현 및 테스트 단계에 이르기 까지의 개발에 적용하였다.

나. RTO.k 에 기반한 실시간 시뮬레이션의 적용

객체 모델은 원래 시뮬레이션 응용에 적용하기 위하여 사용되었다. 그러므로 환경 객체를 모델링하는데 있어서 객체 모델, 즉 시간이 변하는 내부 상태를 가진 환경내의 모듈 개체를 사용하는 것은 오랫동안 활용되었다. 그러나 RTO.k 객체 모델은 환경 객체를 더욱 정확하고 상세하게 모델링 할 수 있게 하여 준다. 이러한 측면과 RTO.k 객체 모델이 요구 명세 과정 및 상위 수준의 설계 단계에서 어떻게 사용되는지를 국방 C3 분야의 예를 통해 보여주고자 한다.

[그림 2-2]에 나타난 미사일 격퇴 국방 시나리오를 보자. 이 구조에 나타난 환경은 "전장(theater)"이라고 불리는 육해공 단편이며 이 전장내에는 전함과 같은 값비싼 방어 목표물과 공격적인 적군 운항체(hostile reentry vehicle, RV) 및 위협적이지 않은 느린 이동 물체등의 비행체들이 포함된다.

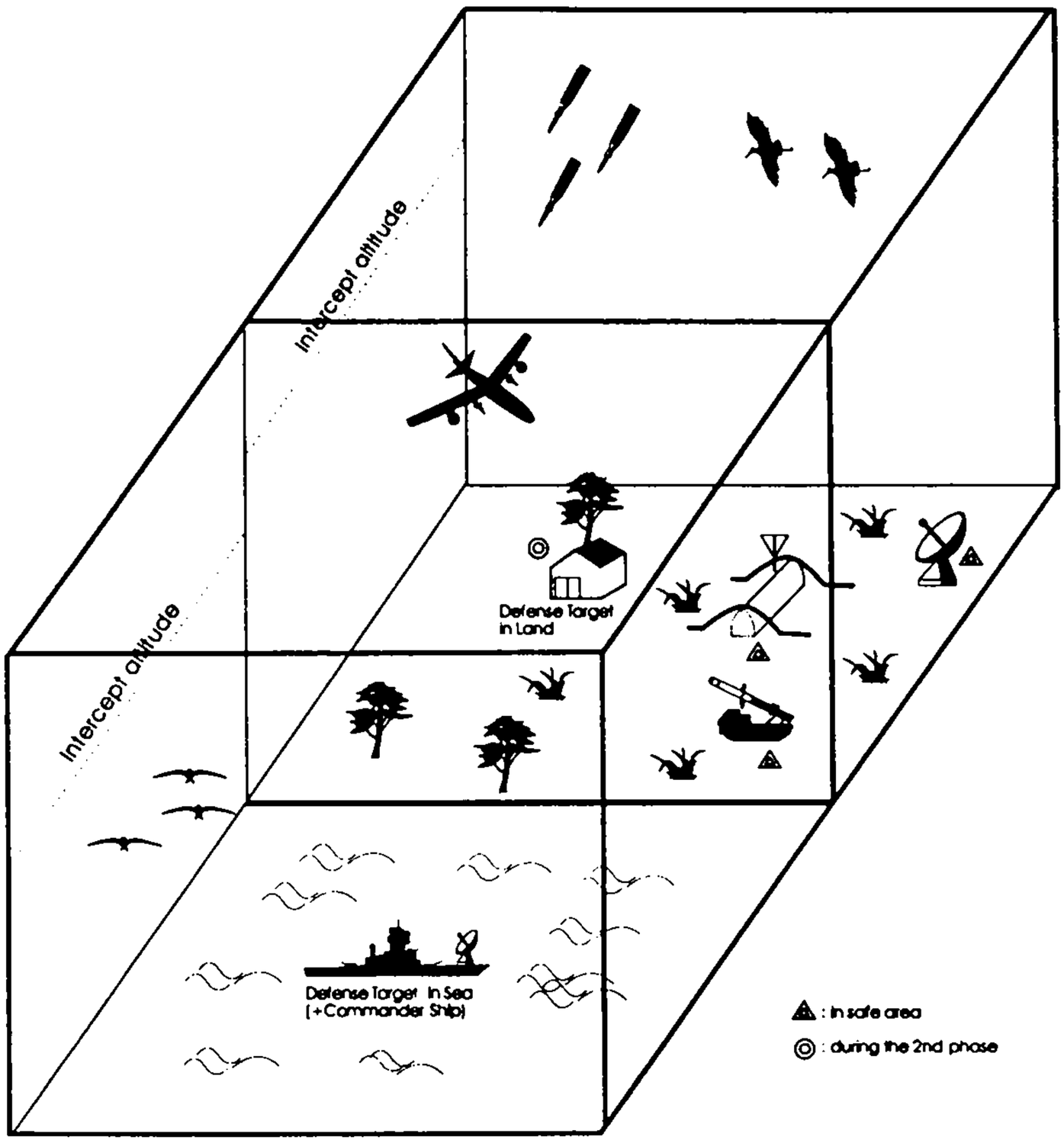
방위 시스템을 지휘하는 지휘관에 의해 주어진 상위 수준의 요구사항은 다음과 같다.

(1) 각 RV 는 이것이 위험하다면 요격되어야 한다.

(2) 요격기 보다 더 위험한 RV 가 있다면 먼저 도달한 RV 는 요격되어야 하고 방어 목표는 안전한 곳으로 이동되어야 한다.

시스템 설계자는 우선 센서들과 액츄에이터(요격기)들의 배치를 결정하여야 한다. 그리고 컴퓨터 제어 시스템의 기능은 채택된 제어 논리(Control Logic)에 기반하여 결정된다. 당분간 센서와 액츄에이터들은 결정되지 않았고 육지에 아무것도 없다고 가정한다. 또한 전장의 하늘 부분에 어떠한 비위협적인 비행 물체도 없다고 가정한다. 이러한 축소한 전장의 RTO.k 표현이 [그림 2-3]에 표현되었다.

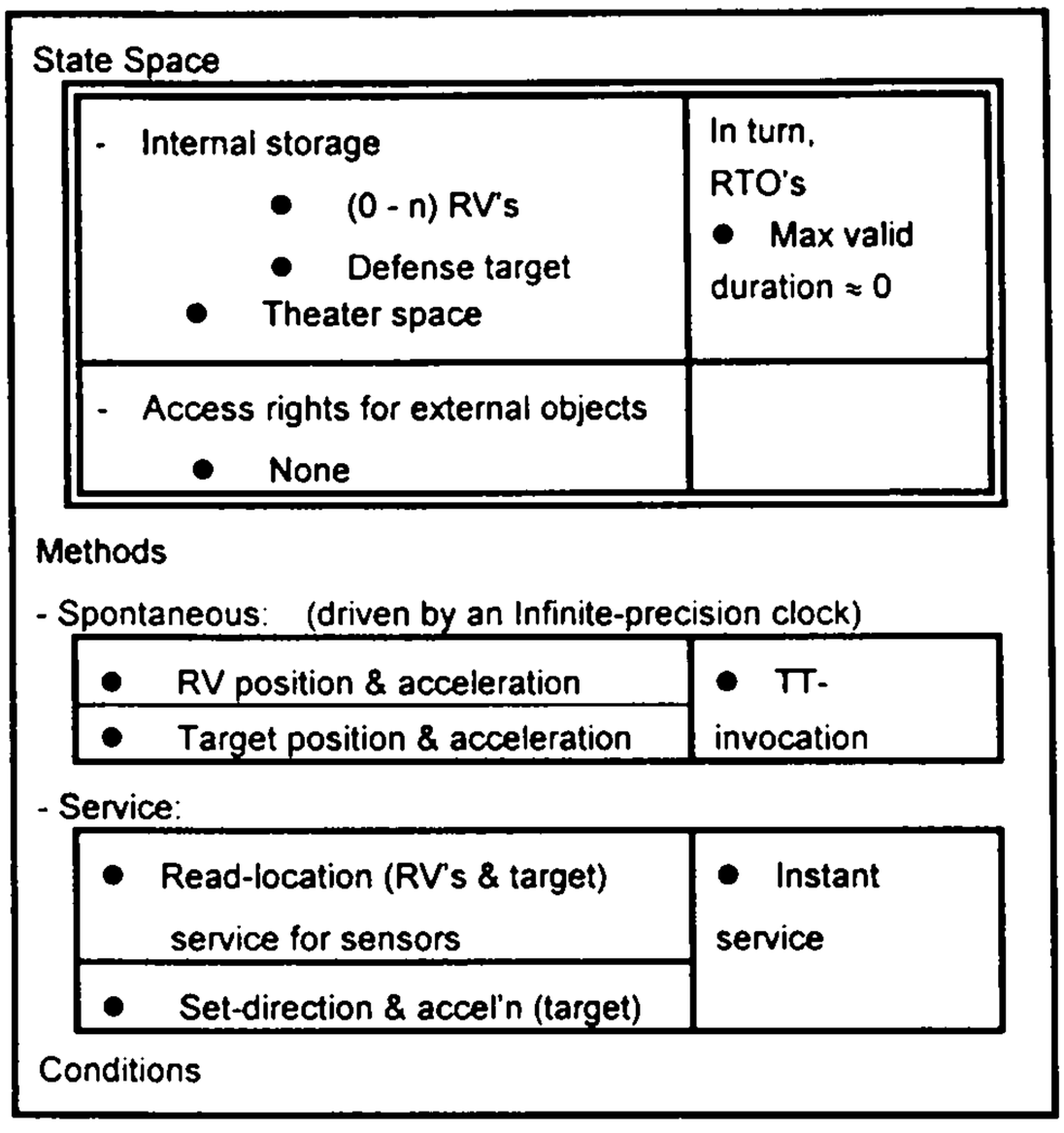
이 상위 수준의 RTO.k 객체의 내부 저장소(객체 데이터 저장소)는 기본적으로 전장내의 공간과 방어 목표(전함) 및 동적으로 변화하는 갯수의 RV 들이다. 그러므로 이 RTO.k 객체내에 보유한 정보는 방어 목표, RV 및 전장 공간내의 어떠한 다른 객체들이 보유한 정보의 조합이다. 여기서 주목할 만한 특성은 이러한 각각의 요소들은 방어 목표물 및 RV 자체등의 객체 데이터 저장소의 요소들로 취급되며 이것들은 RTO.k 객체로 표현될 수 있다는 것이다.



[그림 2-2] 시뮬레이션 응용 환경 : 전장

개념적으로 전장에 대한 이 상위 수준 RTO.k 객체 모델은 연속적으로 구동되고 그들 실행의 각각은 즉시 완료된다. 시간 영역이 이산 영역이며 영역내에 인접한 두 순간간의 시간간격이 시계 바늘 움직임(clock tick)으로 불리어지는 덜 정확한 모델의 버전을 채택하면 환경에 대한 덜 정확한 표현이 나타나게 된다. 즉, 이러한 관점은 어떠한 자발적 메소드의 구동 빈도도 시계 바늘 움직임당 한번을 넘지 않으며 반면 각각의 실행은 동일 메소드의 이어지는 구동 시간 이전 또는 이에 맞추어 완료됨을 나타낸다. 따라서 자율적 메소드는 환경 객체내에서 자연적으로 발생하는 연속적인 상태 변화를

표현(시뮬레이트)하는 장치이다. 환경 객체간에 존재하는 자연적인 병렬성은 동시적으로 구동될 수 있는 다중의 TT 메소드를 사용하여 정확하게 표현된다. 일반적으로 환경의 RTO.k 객체 표현의 정확성은 TT 메소드의 구동 빈도의 직접 함수이다.



[그림 2-3] 축소 전장의 RTO.k 명세

[그림 2-3]의 서비스 메소드는 센서와 액츄에이터는 전장밖에 위치한다고 가정하고 정의되었다. 이러한 측면에서는 [그림 2-3]에서 가정한 전장은 [그림 2-2]에 보여진 전장과는 다르다. [그림 2-3]에서 서비스 메소드를 호출하는 클라이언트는 환경 객체들과 인터페이스하는 센서와 액츄에이터들이다. 센서는 환경 객체들의 상태에 관한 정보를 얻기위해 작용하는데 주로 위치, 운동 방향과 RV 및 방어 목표물의 특성 정보들을 구한다.

센서와 환경 객체간의 이 관계는 [그림 2-3]의 "Read-location(환경 객체)"과 같은 서비스 메소드에 의해 부분적으로 표현될 수 있는데 이러한 메소드는 센서가 환경 상태를 관측한 순간에 실행된다.

이와 유사하게 액츄에이터는 환경 객체의 조건 및 향후 진행과정에 영향을 주도록 작용한다. 이 예에서는 설계된 시스템에 의한 RV에 가해지는 유일한 가능한 영향은 적군의 RV에 대한 요격체의 충돌이다. 이러한 요격체들는 시스템 설계의 초기 단계에 생성되는 액츄에이터들이며 이들은 한번에 생성되며 전장내에서 또한 환경 객체로서 취급되어야 한다. [그림 2-3]이 이러한 액츄에이터를 도입하기 전에 전장을 표현함에도 불구하고 컴퓨터 시스템으로 부터 접근 가능한 방어 목표물내에 제어점(control point)들이 있으며 이것들은 무선 통신 장비(radio communication device)등의 통신 채널을 통한 제어 컴퓨터 네트워크(control computer network, CCN)로서 주로 구성된다. [그림 2-3]의 "Set-direction & acceleration (target)" 서비스 메소드가 이러한 가능성을 표현한다.

[그림 2-2]에 표현된 전장은 이미 센서를 포함하고 있는데 즉, 육지의 레이더, 바다의 전함상의 레이더가 있고 액츄에이터로서는 육지의 요격체, 전함의 요격체가 있다. CCN은 역시 전함내에 구축되어 있는데 이것들은 모두 전장내에 있다. [그림 2-2]의 이 전장의 보다 상세한 RTO.k 표현이 [그림 2-4]에 표현된다.

전장의 하나의 RTO.k 객체 명세는 RTO.k 객체 명세들의 네트워크로 분할 되며 각각은 각기 다른 환경 객체 또는 컴퓨팅 객체에 대응된다. [그림 2-5]는 이러한 분할과정을 보여준다. 하나의 RTO.k 명세의 조건 섹션내에 포함된 모든 지식은 보존되어야 하며 대부분은 하위 수준의 객체 명세의 네트워크로 분산된 형태를 갖는다. 추가되는 지식은 또한 분할 과정중에 도입될 수 있다. 여기서 본것과 같이 환경 객체의 모든 RTO.k 객체 명세는 환경 객체에 대한 RTO.k 객체 구조 시뮬레이터로 변환될 수 있다. 시뮬레이션 정확도의 탄력적인 제어 및 시뮬레이터의 효율적이고 구조적인 생성은 여기서 본

바와 같이 이 모델링 및 시뮬레이션 접근방법의 중요한 장점이다.

Theater

Max. validity duration= ??

None

Sacn-search schedules received
Spot-check schedules received
Current angle position
Width of pulse
Radar returns received
Radar report prepared

SpM

"Update the states of the physical objects in the theater."

Recognize newly entered objects.

Update the states of Comand Post In Land.

Update the states of Defense Target In Sea.

Update the states of Radar In Land.

Update the states of Land-lanched Interceptor 1.

Update the states of Land-launced Interceptor f.

Update the states of Reentry Vehicle 1.

Update the states of Reentry Vehicle n.

Update the states of Non-threatening Object 1.

Update the states of Non-threatening Object m

SvM

None.

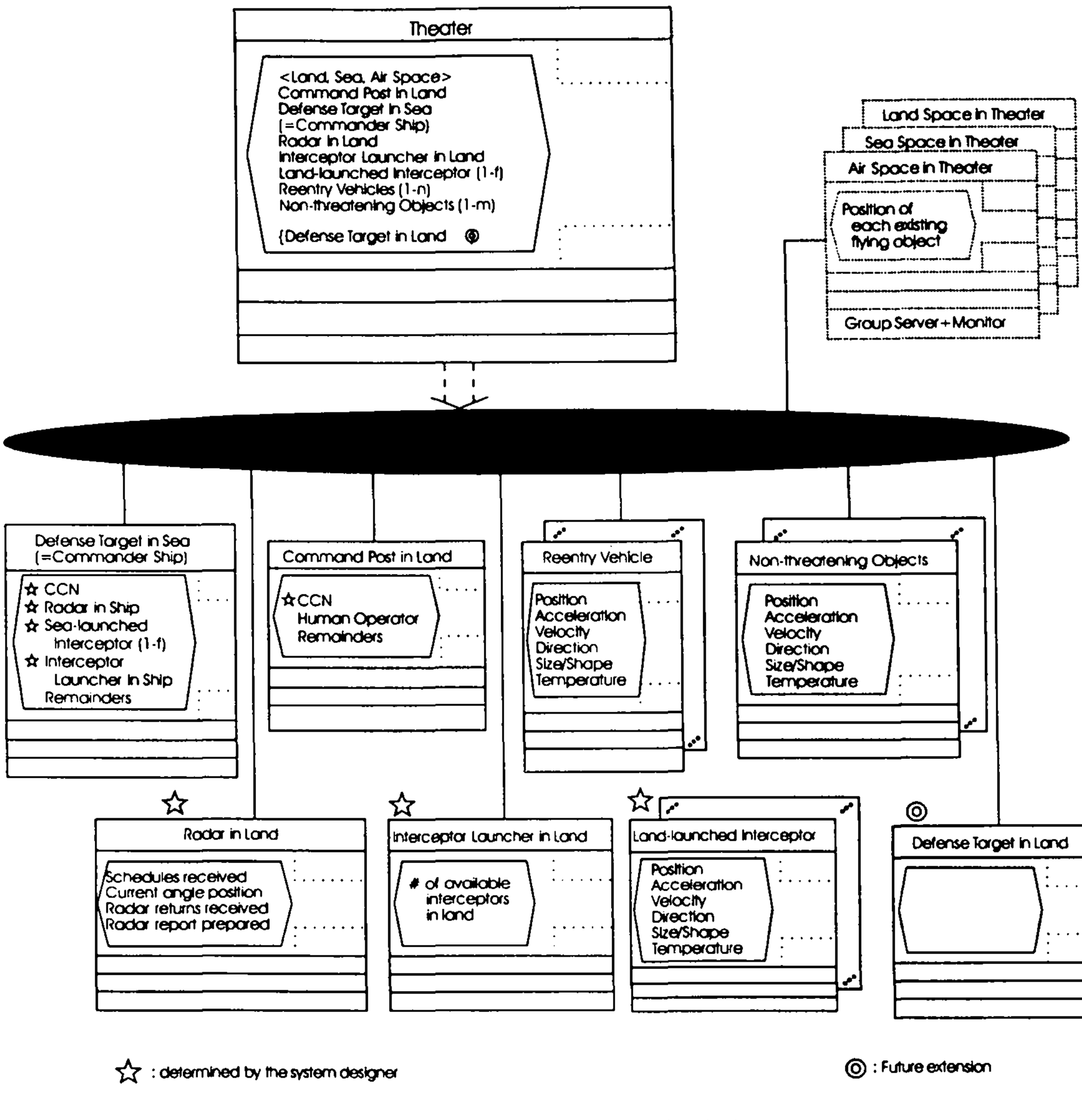
Conditions

1. Req's from customer

- By operating the remote sensor and the remote actuator In Land, the command post in land must attempt to track any RV in the theater and have an interceptor destroy a threatening RV.
- By operating the sensor and the actuator within itself, the commander ship must attempt to track any RV in the theater and have an interceptor destroy a threatening RV.
- Target movemet must occur to avoid attacking RV's when iterceptors are exhausted.

2. Physics law

[그림 2-4] 전장의 RTO.k 명세



[그림 2-5] 전장 RTO.k 의 분할

5 절 미시적 실시간 시뮬레이션의 적용

1. 대상 시스템의 개요

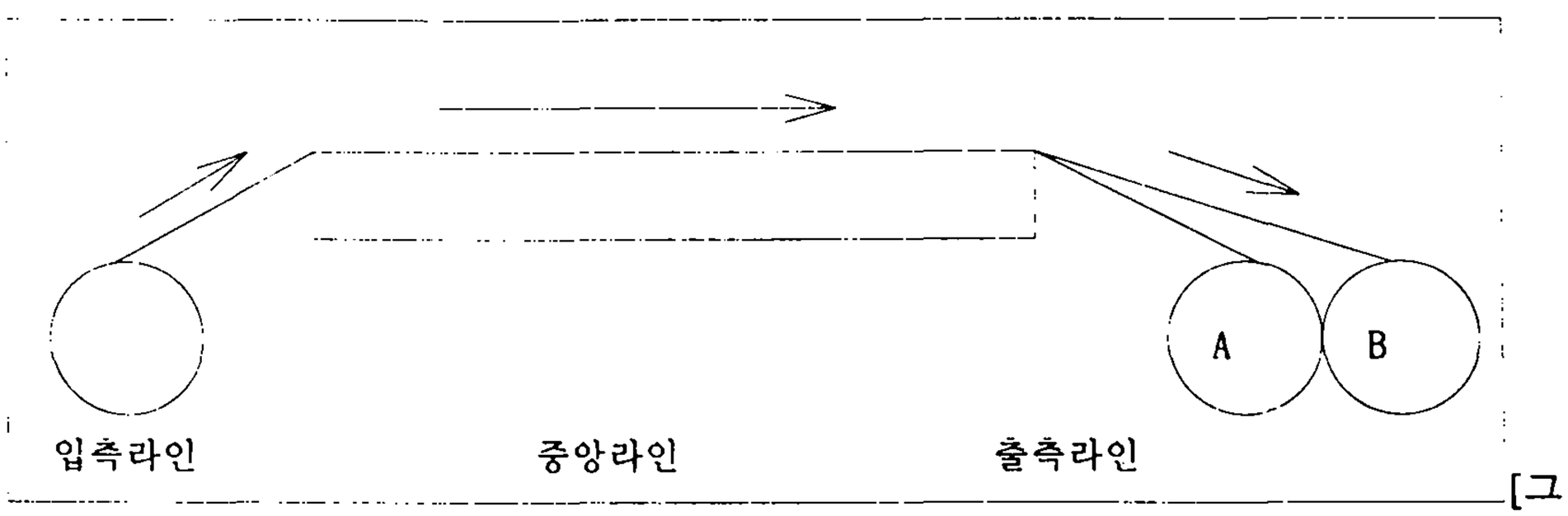
가. 냉간 압연 공정 제어 시스템의 개요

철강 제조 공정은 크게 제철, 제강 및 압연 공정으로 나누어지며 그중에서도 최종 제품의 품질을 직접 좌우하는 압연 공정이 특히 중요하다. 선진국에서는 1960 년대 이후부터 컴퓨터에 의한 자동제어 시스템의 도입이 적극 추진하여 압연 공정에서의 생산성을 향상시키고, 원료를 절감하고, 품질을 향상시키기 위한 노력을 효과적으로 수행하여 왔다. 최근에는 압연 제품에 대해 소비자들은 고품질화, 다품종 소량화, 납기일 단축 등을 절실히 요구하고 있어, 압연 공정에서의 제어 기술의 발전이 시급히 요구되고 있다. 압연 공정의 효과적인 제어를 위해서는 공정 모델링, 제어 이론 및 컴퓨터의 사용이 필수적이며 그외 센서 및 액추에이터의 성능도 영향 요인이 된다.

압연 공정은 열간 압연 공정 및 냉간 압연 공정으로 나뉜다. 열간 압연 공정이 강괴(슬라브)에 영결점 이상(1300 도 C 이상)의 열을 가해 압연기를 통해 열연 코일을 만들어 내는 것이라면, 냉간 압연 공정은 영결점 이하(700 도 C) 의 열을 가하여 하나의 열연 코일을 미리 정해진 두께의 다른 코일로 만들어 내는 공정이다. 이러한 작업을 수행할 수 있게 하는 압연기는 여러 개의 압연 로울로 구성되어 있다. 냉간 압연기에 입력된 열연 코일은 여러 단계의 압연 로울을 지나면서 원하는 두께의 다른 코일로 만들어져 최종 제품화된다.

냉간 압연의 공정 라인은 [그림 5-1]과 같이 크게 입측 라인, 중앙 라인, 출측 라인으로 구성된다. 입측 라인은 열연 코일을 중앙 라인의 시작인 POR(pay of reel)로 이동시키며 각종 기기의 자동 설정을 수행한다. 중앙 라인은 입측 라인에서 받아들인 열연 코일을 연속적인 작업 수행을 위해 영접(두개의 코일의 끝을 연결하는 것)하여 이어진 형태의 스트립(strip)으로 만든 후 연속적으로 압연을 수행한다. 출측 라인은 압연이 완료 된

스트립을 냉연 코일로 만든 후 각 코일의 무게를 측정하는 등 제품과 실적 데이터를 수집한다.



[그림 5-1] 냉간 압연 공정의 개요

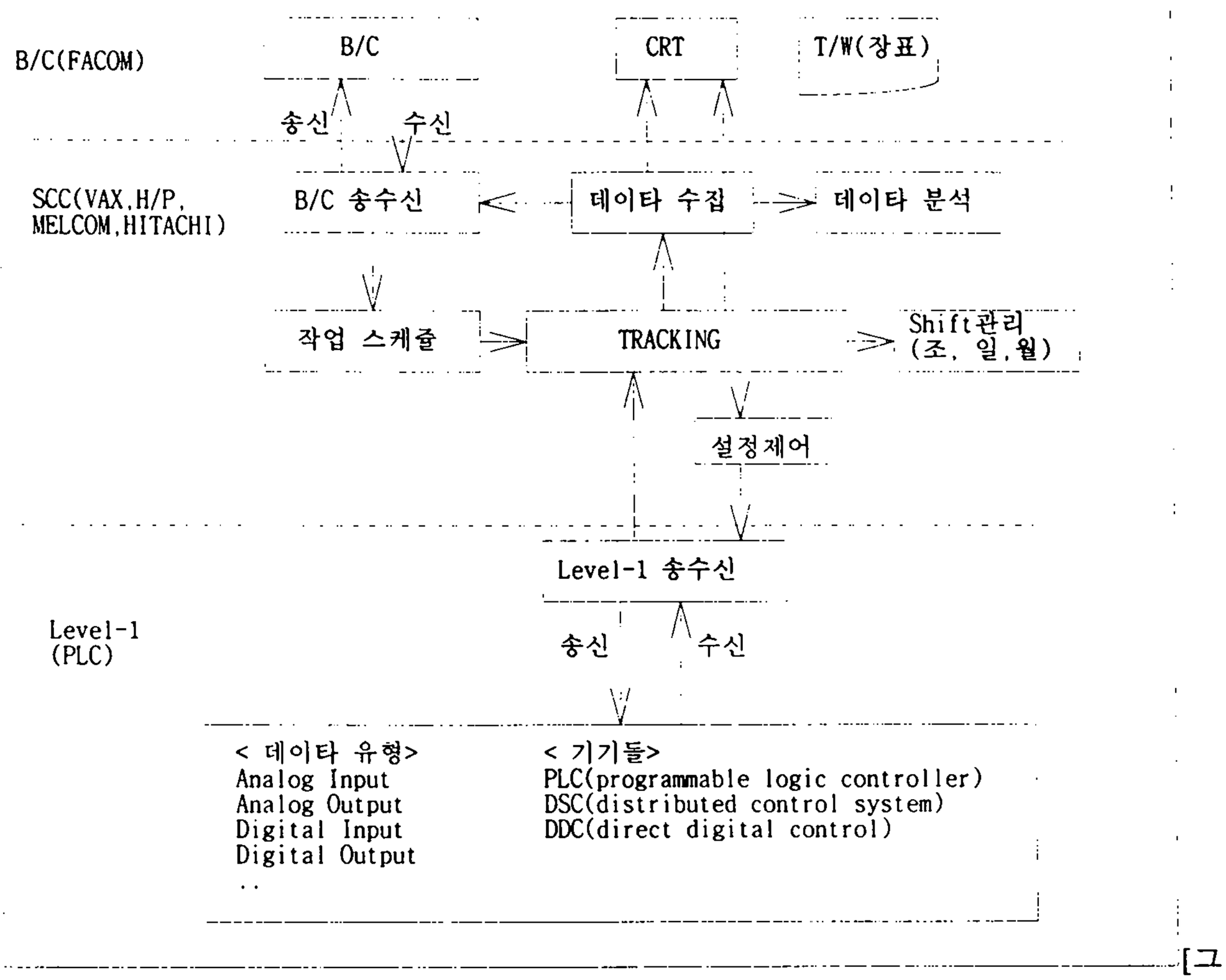
나. 전체 시스템의 제어 구조도

공정 라인 전체의 관점에서 바라보면, 공정들의 기능들은 다음과 같이 여러가지 수준으로 구별하여 판별할 수 있다. 우선 전체 라인은 3 가지의 수준으로 나뉜다.

- B/C(business computer) 수준: 고객의 주문, 생산 계획 등을 받아들여 SCC 수준에서 작업을 수행할 것을 지시하고 생산 전반에 걸친 데이타 흐름을 관장한다. 즉 이 수준에서는 B/C link 를 통하여 작업 지시 및 작업 실적을 관리한다. 또한 CRT 를 통해 operator 의 작업을 인도한다.
- SCC(supervisory control system) 수준: B/C 수준에서의 작업 수행 명을 받아들여 실제 생산을 하기 위한 제반 기능들을 수행한다. 이 수준의 주된 기능으로는 입측, 중앙, 출측라인에서의 소재를 tracking 하고, level-1 link 를 통해 각종 생산 기기들의 설정 제어를 하며, 생산 관련 데이타를 수집하고 분석하여 B/C 수준으로 전달하는 등의 기능을 한다.

● level-1 제어 수준: PLC, DDC 등과 같은 여러가지 제어 기기들을 직접 동작시켜 라인을 운용하는 기능을 담당한다.

각 수준별 기능들과 데이타 흐름을 살펴보면 다음 그림과 같다.



[그림 5-2] 냉간 압연 제어 공정

다. 응용 대상 시스템의 구성 요소

본 연구에서 대상으로 하고 있는 시스템은 냉간 압연 공정 제어 시스템의 일부로서

중앙 라인 및 출측 라인의 일부를 중심으로 하고 있다. 즉 압연로울을 지난 코일을 두개의 가용한 축(그림 5-1 에서 A 축 또는 B 축)에서 감아들이면서 원하는 길이로 자른 후에 무게를 측정하고 그것을 포장하기까지의 공정만을 대상으로 한다.

본 대상 공정이 수행될 때 고려되어야 하는 사항들은 다음과 같다.

● 입력되는 열연코일들은 공정이 연속성을 가질수 있도록 입력되는 열연코일들의 끝부분들은 용접 기계에 의해 용접되어 연결되어야 한다. 이렇게 연결된 부위는 다른 부분과는 재질 특성이 다르기 때문에 제품 생산시 cutter 에 의해 잘라주어야 한다. 따라서 이 부분은 구멍을 뚫어 놓아 센서에 의해 감지되도록 하여 cutter 가 자를수 있게 하고 있다.

● 산출물로 나오는 냉연 코일은 두개의 축에 교대로 걸리게 한다. 코일이 정해진 길이만큼 생산되면 현재 작동중인 축은 정지시켜 제품을 포장할 수 있게 한다. 이 때 다른 축을 준비시켜 다음 생산에 대비한다.

● 모터의 속도는 축에 작용하는 장력(tension)에 의해 결정되는데 장력이 커지면 속도를 늦추어야 하고 장력이 줄어들면 속도를 빠르게 한다. 이때, 감는 축의 반지름이 커질 경우 장력이 커지게 되므로 모터의 속도는 느리게 하여야 한다.

● 평량(저울)에 만들어진 코일이 얹혀지면 그 무게를 측정한 후 그 값을 표시하고 포장단계로 넘어간다.

● 현재 라인에서 생산되는 코일의 길이는 PLG(pulse generator)를 이용하여 계산한다. 코일의 길이는 모터의 속도*시간에 의해 계산될 수 있다.

라. 대상 시스템의 실시간 특성 및 고려 사항

하나의 대상 시스템이 실시간 특성을 가진다는 것은 그 시스템이 계산의 정확도를 요구할 뿐 아니라, 반응 시간에 제약이 있다는 것을 의미한다. 즉 실시간 시스템은 정

상 상태에서 긴급한 사건들에 대해 예측 가능한 빠른 응답이 가능해야 하며, 비정상 상태에서 시스템의 안정성을 보장할 수 있어야 한다.

정밀한 냉연 공정은 많은 데이타 소스(data source)에서 발생한 데이타를 즉시, 또는 시간의 제약을 가지고 다양한 처리절차를 거쳐 처리해 주어야 한다는 점에서 실시간 시스템의 특성을 잘 반영하고 있다고 할 수 있다.

냉연 공정을 하나의 실시간 시스템의 응용이라고 보았을 때, 다음의 사항들은 각 공정이 진행되어 갈 때 고려하여야 하는 것들로 실시간 시스템의 구현을 위해 꼭 표현되고 고려되어야 한다.

● 주기적인 처리에 대한 시간 제약 사항 : 다음 사건이 발생하기 전에 모든 주기적인 작업을 마쳐야 한다. 이를 위해서는 각 주기적인 사건의 마감 시간을 명시하고 이에 따라 필요한 타스크를 주기적으로 기동시킨다. 예를 들어 cutting 작업을 할 경우 cutting 준비, cutting, cutting 완료 등이 하나의 주기를 이루므로 cutting 작업의 수행은 다음 cutting 이 발생하기 이전에 완료되어야 한다.

● 공정상에 차례대로 발생하는 사건들: 사건의 특성에 따라 각 사건들에 부과되는 시간 제약 조건에 맞게 사건을 처리해야 한다.

● 사용자에게 보여지는 화면에 대한 시간의 제약 조건: 일정한 시간 내에 화면이 갱신되어야 한다.

이러한 실시간 시스템을 분석하는 방법은 이론적인 방법, 프로토타입 방법, 시뮬레이션 방법이 있을 수 있다. 그러나 이론적으로 분석하는 방법은 실시간 시스템이 가지는 복잡성으로 말미암아 현실성이 떨어지고, 프로토타입을 이용한 방법은 다양한 시스템 환경에서 성능 평가가 어렵기 때문에 시뮬레이션을 통한 분석이 바람직하다고 할 수 있다.

본 연구에서의 대상 시스템으로서 냉연 공정 전체가 아니라 일부만을 선택한 것은 일부만으로도 실시간 시스템이 가지고 있는 다양한 특성들이 거의 다 포함되어 있으며, 냉연 공정 전체로 쉽게 확장될 수 있기 때문이다.

2. 실시간 시뮬레이션을 위한 모형화

가. 모형화 접근 방법

(1) 냉연 공정 시스템의 제어 과정

냉연 공정 시스템의 제어 시스템(컴퓨터)은 공정 여러 곳의 센서에서 발생하는 데이타를 받아들여 처리한 후 목적에 맞도록 운영하는 기능을 수행한다. 이 때 정상적인 데이타의 처리는 물론이고 시스템의 응급 상황등에도 대처할 수 있어야 한다. 물론 이 시스템은 실시간성 시스템이므로 시스템의 작업 수행에 시간 제약이 따라야 한다. 이를 간단히 표시하면 다음 [그림 5-3]과 같다.

(2) 실시간 시뮬레이션 시스템의 구현시 고려할 점

대상 시스템을 실시간 시스템으로 구현하기 위해서는 실제 응용 환경에서 발생하는 여러 사건들의 상대적인 시간성이 정확하게 시뮬레이터에 의해 모방되어야 한다. 따라서 사건들의 순서가 실제 환경에서나 시뮬레이터 상에서 같아야 함은 물론 실제 환경에서 일어난 두 사건간의 시간 간격과 상응하는 시뮬레이터 상에서의 시간 간격과 비율이 어느 두 사건을 고려하더라도 항상 일정하거나 미세한 변동폭을 보여야 한다. 또한 시뮬레이션 속도는 실제 응용 시스템이나 환경이 진행하는 속도만큼 빨라야 한다.

대상 시스템의 실시간성의 분석은 공정상의 사건들을 분석함으로써 시작된다. 사건은 공정 라인에서 올 수도 있고, 작업 지시 명령과 같은 B/C(business computer) 에서 올 수도 있으며, 사용자 오퍼레이터에 의해 발생할 수 있다. 이러한 각 사건들별로 그 특성을 분류하여 처리 절차 등을 정의하는 것이 중요하다.

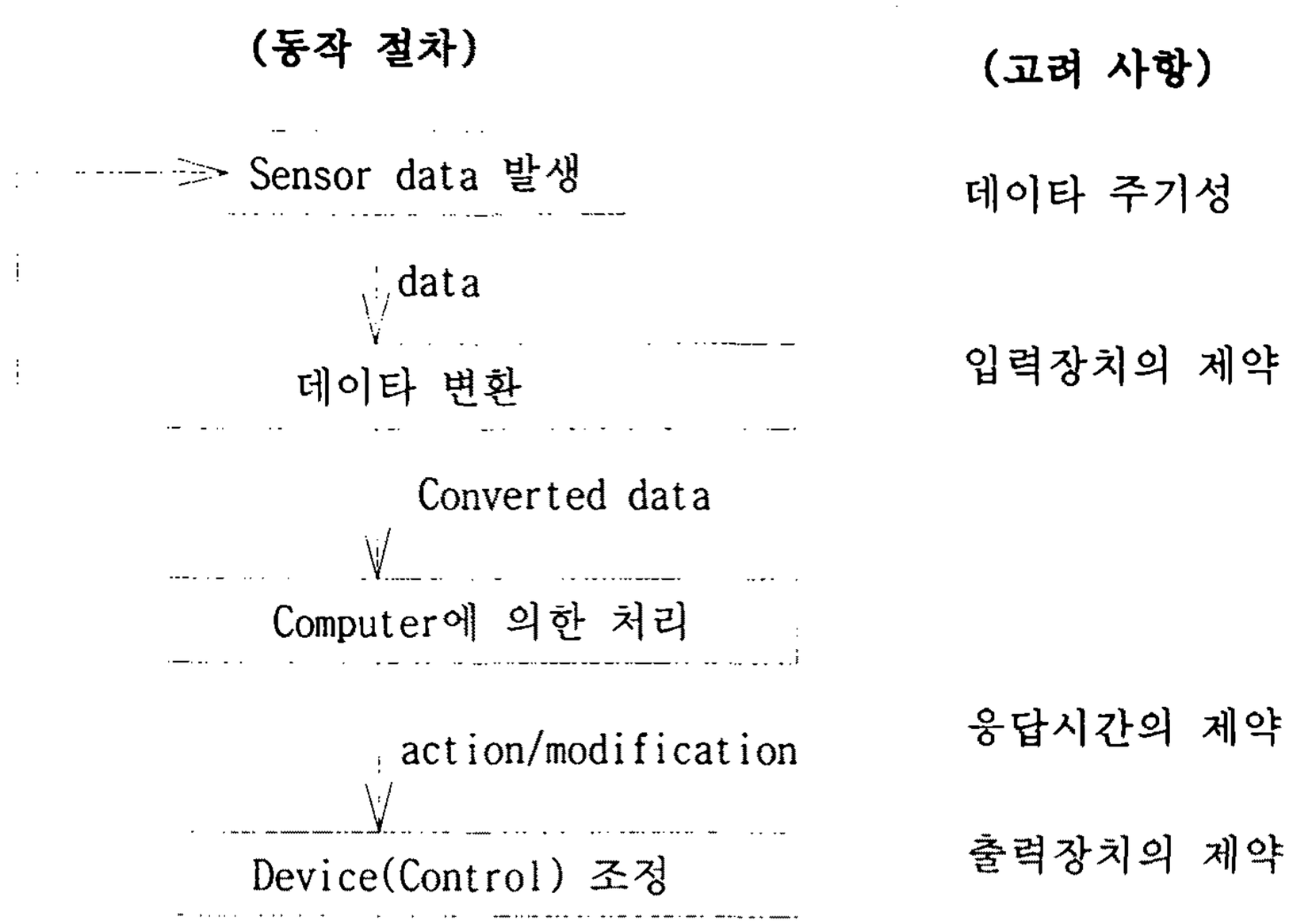


그림 5-3] 공정 제어 절차

(3) 태스크 특성

태스크(task)는 실시간 시스템에서 사용되는 제어의 단위로서 각 태스크는 독립적으로 수행될 수 있어야 한다. 실시간 시스템을 수행하는 소프트웨어는 병행 처리를 할 수 있는 태스크로 구성된다. 실시간 시스템을 구성하는 태스크는 여러가지 방법으로 나뉠 수 있다. 다음은 공정을 구성하는 기능이 태스크로 분할되는 하나의 기준을 제시하고 있다.

- dependency on I/O: I/O device 에 의존하는 기능들
- User interface dependency: 사용자 I/F 에 의존하는 기능들

- Periodic execution: 주기적으로 실행되는 기능들
- Periodic I/O execution: 주기적으로 I/O 가 실행되는 기능들
- Time critical function: 시간상으로 중요한 기능들
- Computational requirement:계산이 요구되는 기능들
- Functional cohesion: 기능상의 결합도가 강한 기능들, 즉,상호 밀접한 기능상의 관련

성을 가지고 기능들간의 데이타 송수신이 빈번할 경우

- Sequential cohesion: 연속적인 수행상의 결합도가 강한 기능들
- Temporal cohesion: 시간상의 결합도가 강한 기능들, 동일한 사건에 의해 trigger 되는 기능
- 화일 관리를 담당하는 기능들은 library function 으로 구축한다.

태스크가 정해진 경우 각 태스크들은 서로간의 통신을 통해 필요한 정보를 주고 받으면서 동작한다. 이러한 태스크간의 인터페이스는 메시지를 이용하거나, 데이타 저장소와 같은 정보 은닉 모듈을 사용하거나, 두개 이상의 태스크의 동기화에 사용하는 사건 시그널을 이용할 수 있다.

태스크는 우선 순위를 가지고 여러가지 방법으로 발생되고 수행된다. 태스크는 주로 주기적으로 기동되거나, 정시에 기동되거나, 사건에 의해 기동되는 경우가 많고, 그외에 시스템 태스크, 간헐적 기동, 태스트 기동 태스크 등에 의해 기동된다.

각 태스크는 제어장치의 메모리에 계속 상주하는 태스크와 기동될 때만 적재되는 비상주 태스크로 나눌 수 있다. 이러한 태스크 스케줄링은 우선 순위 선점 방식을 따른다. 또 각 태스크 수행전에 필요한 데이타가 모두 준비되어 있다고 가정한다.

(4) 데이타 특성: 입력/출력 데이타의 유형 설명

대상 시스템의 각 부분에서 발생하는 여러가지 종류의 데이타는 여러가지 태스크를

구동시키는 자극제의 역할을 하게 된다. 각 데이타의 특성에 따라 그 처리방법도 달라지게 된다.

규칙적으로 발생하는 데이타는 적절한 시간에 시스템이 적절한 처리를 하면 되지만 불규칙적으로 발생하는 인터럽트와 같은 데이타는 시스템으로 하여금 빠른 시간내에 처리할 것을 요구하는 경우가 많다. 따라서 대부분 하드웨어에서 발생한 인터럽트를 처리 가능한 digital input based OS 가 있어야 한다.

실시간 시스템이 아닌 기존의 PLC(programmable logic controller)를 이용한 제어 시스템의 경우 1 scan(예: 40 msec) 단위로 주기적으로 작업이 이루어지고 있기 때문에 인터럽트를 적시에 처리하기가 어렵다. 만일 어느 사건이 그 scan 주기 시간 중에 발생하여 프로세스가 그것을 인식하지 못하면 적시에 처리가 이루어지기 어렵다.

(5) 냉연 공정 실시간 시뮬레이터 개발의 기대 효과

냉연 공정 실시간 시뮬레이터의 개발을 통하여 효율적인 공정이 될 수 있도록 공정 제어 컴퓨터의 소프트웨어를 조정, 제어 변수들을 설정하는데 사용될 수 있다. 이러한 시뮬레이터를 이용하면 실제 공정을 동작시키지 않고 가상적으로 신입 사원들의 교육에 활용하여 기본 동작 원리를 이해시키는데 사용할 수 있다.

따라서 냉연 공정에 대한 실시간 시뮬레이션 시스템의 개발을 통하여 우리는 대상 시스템을 계획하고, 개발할 수 있으며 운영 기간 중의 상세한 결정 사항에 대하여 정밀한 정보를 얻을 수 있다.

그러나 실시간 시뮬레이터는 응용분야마다 상당한 설계 노력이 들기 때문에 시뮬레이터의 설치 비용이 크다는 단점이 있다.

나. 대상 시스템의 모델링

(1) 입출력 데이타의 설명

냉연 공정 제어 시스템은 피제어 시스템(controlled system)과 제어 시스템(control system)으로 나눌 수 있다. 피제어 시스템은 여러가지 센서를 포함하고 있으면서 시스템에 데이타를 발생시켜 시스템으로 하여금 적절한 행동을 취하도록 하는 것들을 말한다. 제어 시스템은 제어를 담당하는 컴퓨터를 말한다. 다음은 피제어 시스템과 제어 시스템으로 사용되는 장치들을 보여주고 있다.

[표 5-1] 데이타 발생 장치

장치	기 능
WPD(welding point detector)	코일이 연결된 부분을 감지한다.
cutter	두 코일의 연결 부분을 잘라준다.
motor(A,B)	A 또는 B 축을 속도조절을 하면서 감아준다.
장력측정기	벨트에 걸리는 장력을 측정한다.
평량	코일의 무게를 측정한다.
제어(컴퓨터)	전체 시스템의 제어 기능을 수행한다.

위의 각 장치에서 발생하는 데이타들에 관한 자세한 사항은 다음의 [표 5-2]에 나타나 있다. 또한 제어기가 입력 데이타를 처리하여 실제 시스템을 제어하기 위해 발생시킨 출력 데이타는 다음 [표 5-3]와 같다.

[표 5-2] 입력 데이타의 특성

번호	data	유형	내용	센서위치
I.1	point 여부	digital	Welding 점을 발견함	WPD

I.2	cut 완료여부	digital	cut 작업이 완료되었음	cutter
I.3	사용축	digital	A,B 중 현 사용축 표시	축
I.4	장력	analog	라인에 걸리는 장력을 표시	라인
I.5	코일 무게	digital (analog)	현재 감기고 있는 코일의 무게	축
I.6	평량준비	digital	코일이 평량 위에 위치하여 무게를 측정할수 있는가의 여부	평량

[표 5-3] 출력 데이타의 특성

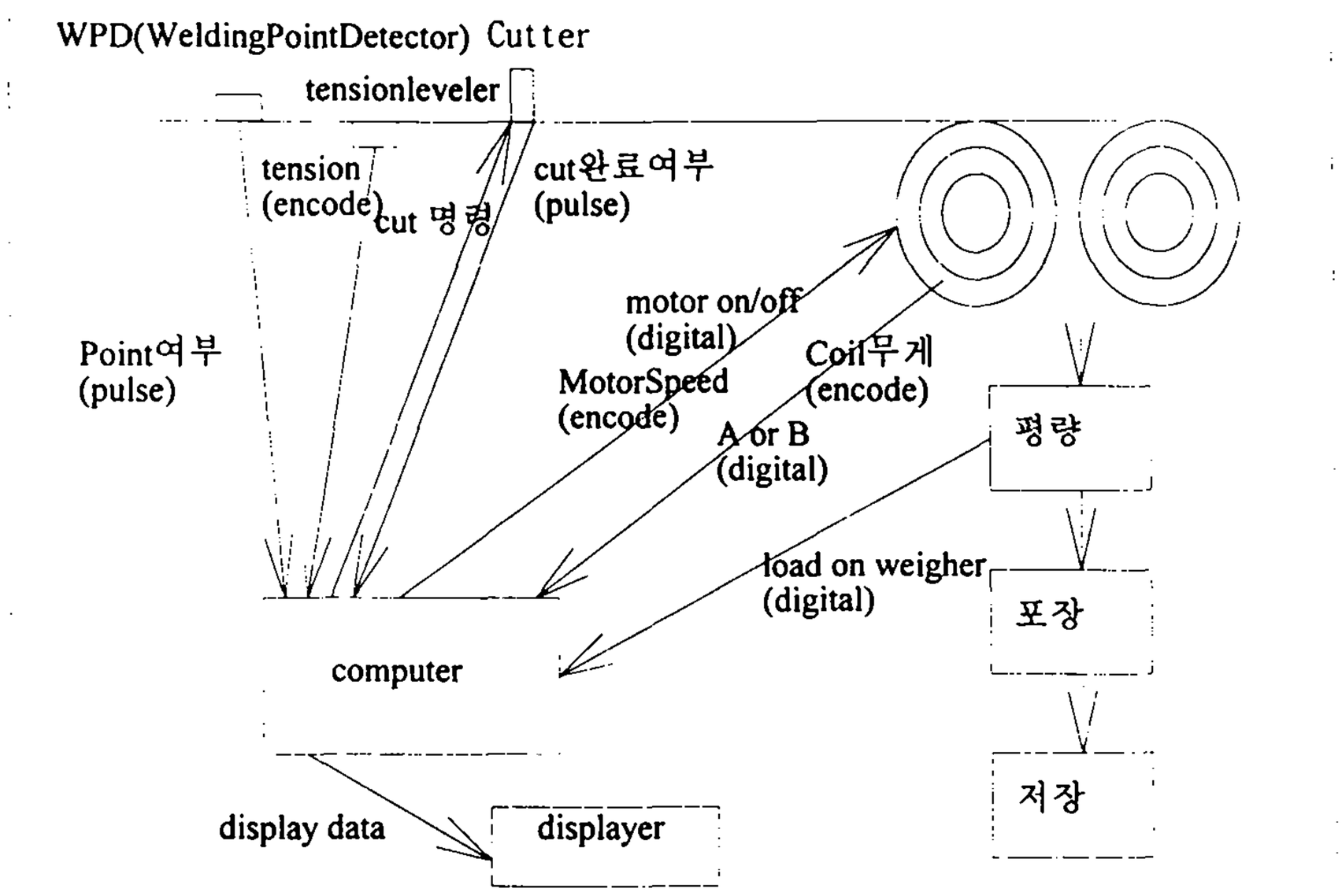
번호	데이타	유형	내용	전달장소
O.1	cut 명령	digital	welding 점 발견시 cut 명령	Cutter
O.2	motor 작동	digital	motor 를 on/off	Motor
O.3	motor 준비	digital	motor 를 준비시킨다.	Motor
O.4	motor 감/가속	analog	motor 의 속도 조정	Motor
O.5	display	analog	코일의 길이 또는 무게 표시	Display

(2) 태스크의 구성

다음 [그림 5-4]은 대상 공정 시스템의 여러가지 하드웨어 구성 요소 및 센서에서 발생하는 여러가지 다양한 데이타들을 나타낸 것이다. 이를 바탕으로 실시간 시스템의 태스크 중심 모형화가 이루어질 수 있다.

냉연 공정 태스크는 다음과 같이 6 개의 태스크로 구성한다. 각 태스크는 다음과 같은

특성 및 처리 절차를 가지고 있다.



[그림 5-4] 대상 시스템의 입출력 특성

(가) CoilingTask (not real-time)

o 정의 : 현재 축에 감겨 있는 코일의 길이를 측정하여 그 데이타를 얻어내는 기능을 수행한다.

o 수행 절차

- PLG 를 통해 입력 데이타(pulse)를 받아들인다.(주기적으로 발생)
- 주어진 데이타를 길이 데이타로 변환시킨다.
- 현재 코일의 길이를 display 한다.
- 일정한 길이가 되면 cut 명령을 내린다.

(나) AiTask(analog task) (not real-time)

o 정의 : 평량에서 보내온 신호(평량에 코일이 loaded/unloaded 상태)를 받아들여 만일 loaded 되었다는 신호가 들어오면 코일의 무게를 측정하는 기능을 수행한다.

o 수행 절차

- 평량에서 신호를 받아들인다.
- load 신호이면 코일의 무게를 측정한다.
- 무게를 display 한다.

(다) DiTask(digital task)(대부분 real-time)

o 정의 : pulse 형태의 여러가지 interrupt 처리를 담당한다.

o 수행 절차

- 여러가지 interrupt 를 받아들인다.

예) welding point detect

cut 완료 여부

- 상황에 맞는 동작을 수행한다.

(라) MotorTask (real-time)

o 정의 : 모터에 관련된 기능을 수행한다. 즉 모터의 구동 준비, 모터의 감가속 등의 기능을 수행한다.

o 수행 절차

- 축의 교환이 이루어진 경우 해당 모터를 정지시키고 다른 모터를 준비시킨다.

- 장력이 큰 경우(코일의 반지름이 커진 경우) 모터의 속도을 감소시킨다.

(마) CutTask (real-time)

o 정의 : welding point 를 감지한 후 cutting 작업을 수행한다.

o 수행 절차

- WPD 에 의해 welding point 가 감지되었다는 데이타를 받아들인다.(real-time)
- cutter 에 cutting 명령을 내린다.

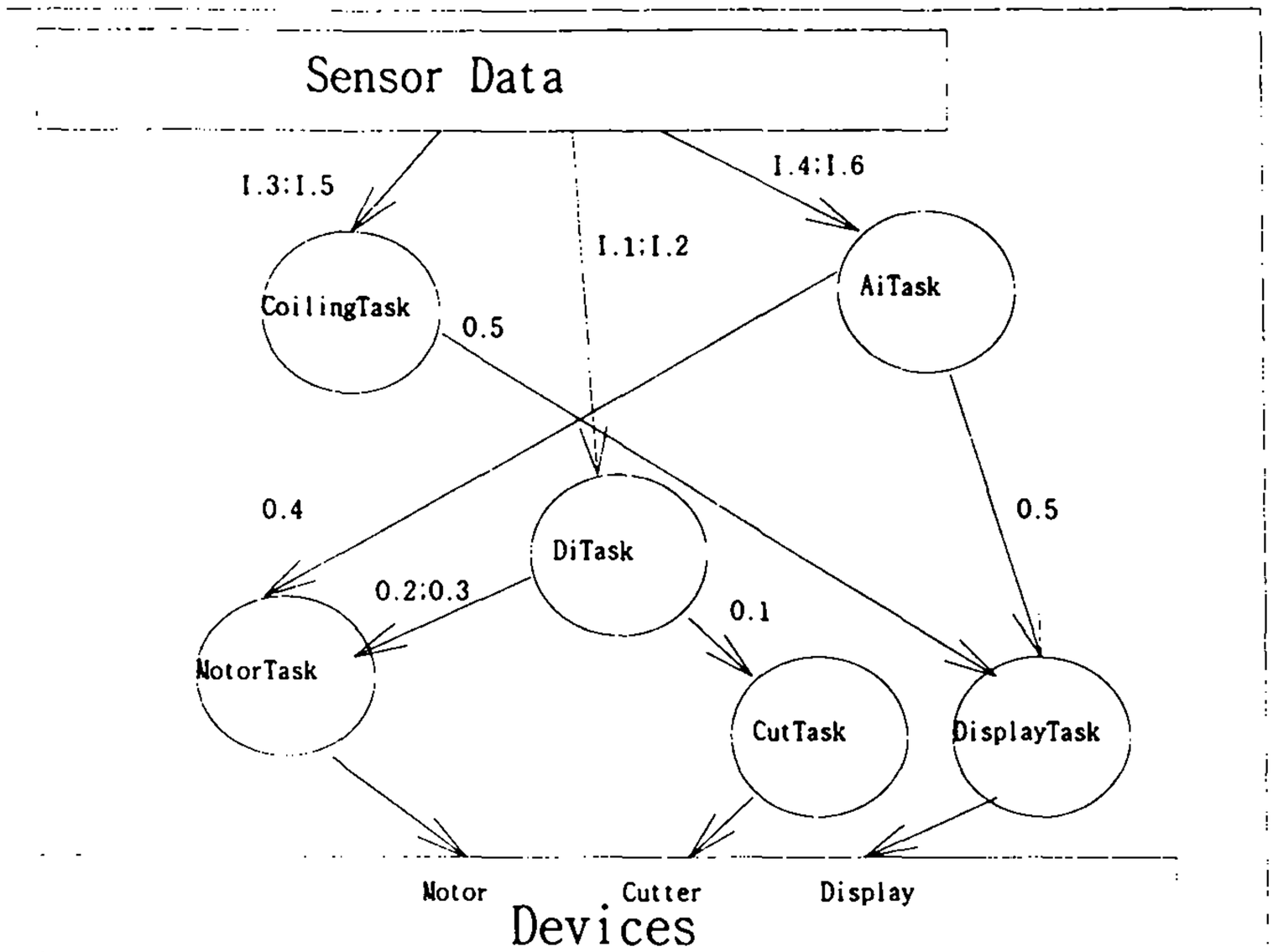
(바) DisplayTask (real-time)

o 정의 : 다양한 정보를 화면에 표시한다.

o 수행 절차

- 여러곳에서 다양한 display 관련 데이타를 받아들인다.
- 데이타를 처리한 후 실시간으로 displayer 에 표시한다.

다음의 [그림 5-5]은 태스크들간의 상호 작용을 나타낸다.

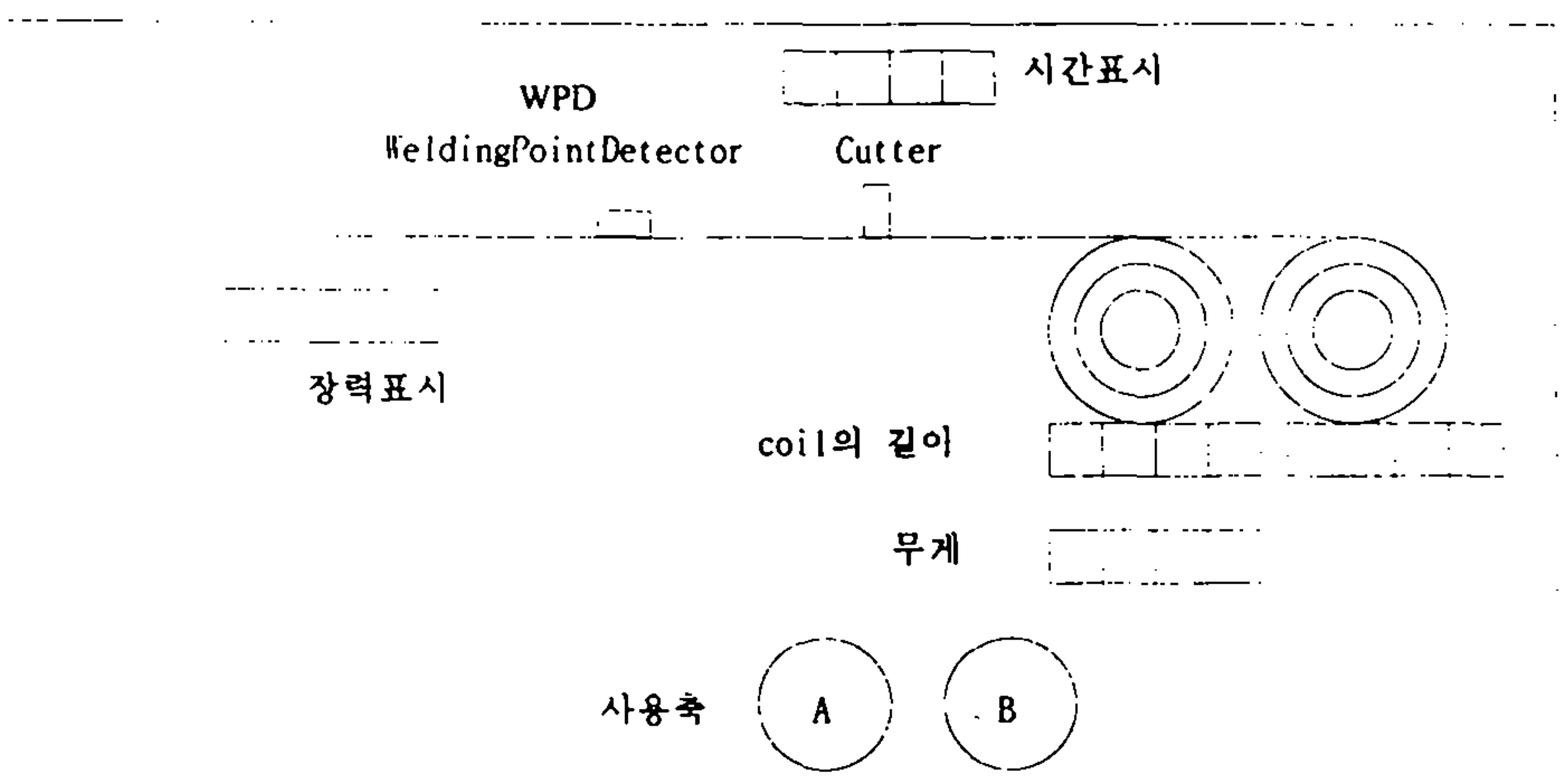


[그림 5-5] 각 태스크의 상호 작용

3. 실시간 시뮬레이션 시스템의 설계

가. 제어 패널(control panel)의 설계

실제로 냉연 공정 실시간 시뮬레이터를 개발할 경우 제어 패널(control panel)에 표시할 부분은 다음과 [그림 5-6]와 같다. 이 화면은을 통하여 사용자는 시스템이 어떻게 작동하는지를 알 수 있고 여러가지 정보를 얻을 수 있다.



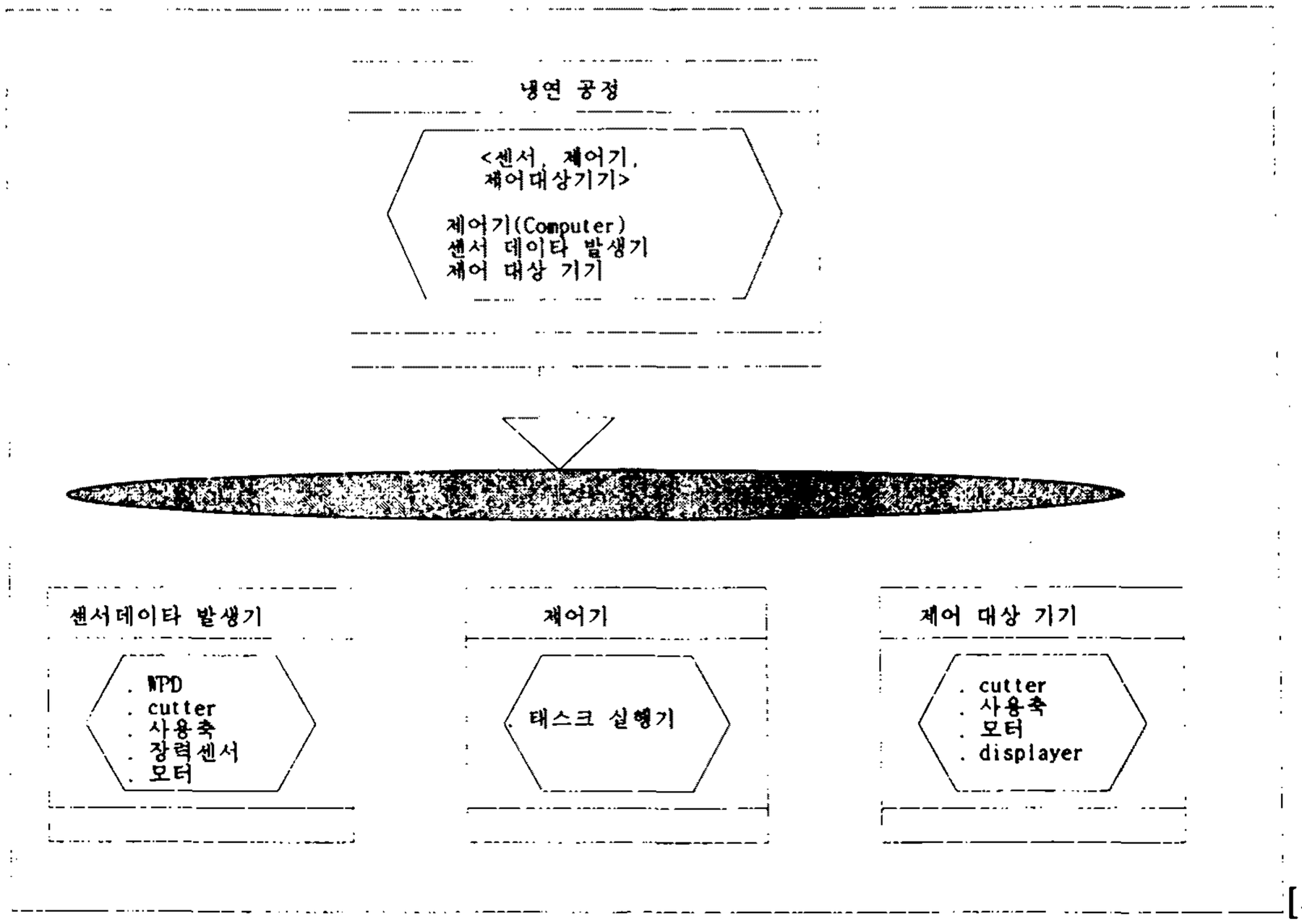
[그림 5-6] 냉연 공정 시뮬레이터의 컨트롤 패널

Control panel 은 공정을 있는 그대로 표현하면서 중점을 두어 관리할 사항들에 촛점을 두어 설계되는 것이 바람직하다. 위의 그림에서는 각 센서들과 관련 데이타, 각 축에 걸린 코일의 길이, 평량에 실린 코일의 무게, 현재 사용축 등에 관한 정보를 보여주도록 설계되었다.

나. RTO.k 로의 모형화

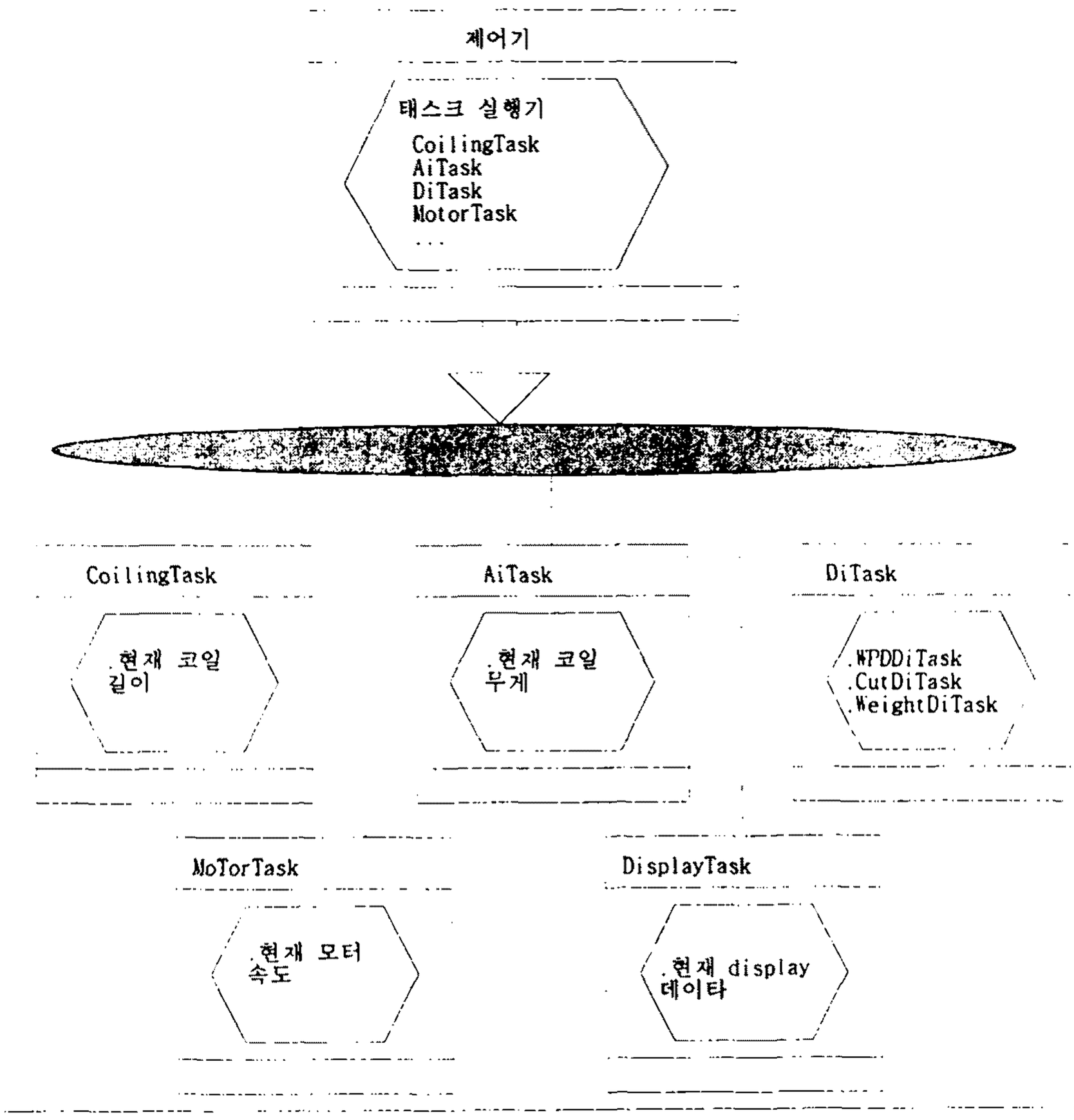
RTO.k 의 모형을 이용하여 냉연 공정 시뮬레이터를 구축할 수 있다. 이 때, 각 RTO

객체는 다양하게 구성할 수 있다. 여기에서는 WPD, Cutter, 사용축, 장력센서, 모터, displayer 등을 중심으로 수행하는 태스크를 중심으로 설계한다. RTO.k 를 이용한 모형화의 장점은 Top-down 방식으로 분할(decompose)하여 점차적으로 구축할 수 있다는 것이다. 즉, 대상 시스템 전체를 하나의 RTO 객체로 보고 모형화한 후 세부 객체들로 만들어가는 방식이다. 이러한 관점을 가지고 냉연 공정을 모형화 하여 보면 다음 [그림 5-7]과 같다. 냉연 공정은 센서 부분, 제어기 부분, 제어 대상 기기 부분으로 나누어 생각할 수 있다. 즉 냉연 공정 객체가 세개의 객체들로 분할되어 모형화되었다는 것을 보여 준다. 이렇게 분할된 객체는 계속적으로 분할할 수 있는데 분할 수준은 시뮬레이터의 목적에 맞게 결정되어야 한다.



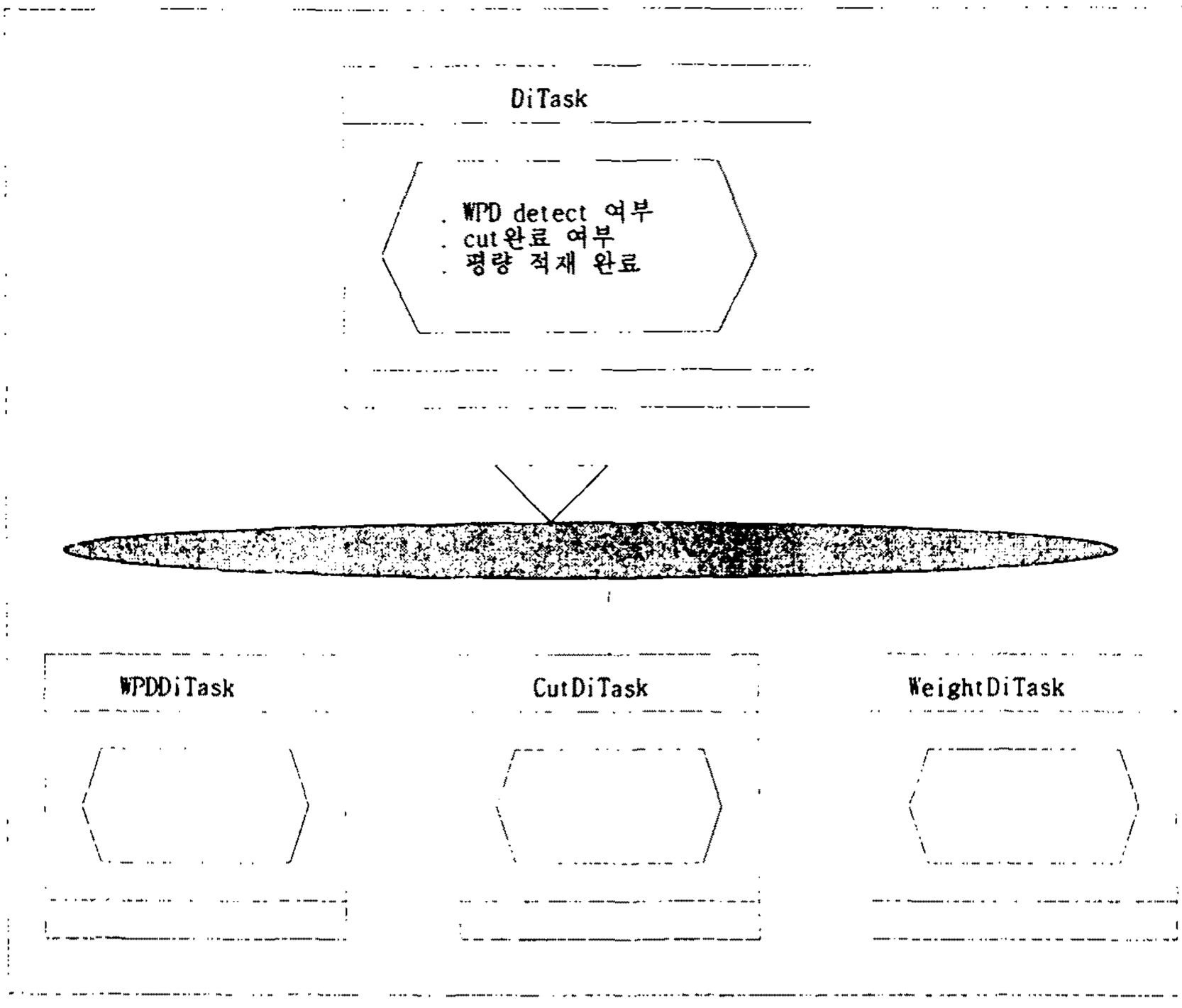
[그림 5-7] 냉연 공정의 전체 RTO.k 모형

제어기는 다음과 같은 태스크를 수행하는 기능들의 합으로 나타낼 수 있다.



[그림 5-8] 제어기의 RTO.k 구조

DiTask는 센서에서 데이타를 받아들여 처리하는 방식에 따라 여러개의 하부 태스크로 나눌 수 있다. 따라서 이 태스크는 다시 분할하여 다음 [그림 5-9]과 같이 여러개의 RTO 객체로 모형화할 수 있다.



[그림 5-9] DiTask의 분할

다음 [그림 5-10]는 위의 대상 시스템 중 CoilingTask를 실시간 시스템의 모델링 방법인 RTO.k의 형식으로 기술해 본 것이다. 그러나, 각 구현의 세부 사항은 포함하지 않고 있다.

CoilingTask

Use (other RTO's)
- DiTask
- DisplayTask

Object Data Store

```
real  time;//  현재의 시간 표시
real  length;// 현재 코일의 길이를
표시
int  axis; // 현재 사용중인 축
```

SpM

SpM 1 <deadline> read_PLG_data
//주기적으로 입력되는 PLG의 값을 읽는다.
AC: every t msec within a usec

SpM 2 <deadline> catch_current_time
// 화면에 표시할 시간을 가져온다.
AC: every t msec within a usec

SpM 3 <deadline> display_length
// 길이값을 화면에 display 한다.
AC : every t msec within a usec

SvM

SvM 1 <deadline> SvM-name calculate_length
// PLG값에서 길이를 계산한다.

Conditions
1. Conditin theory
- PLG data는 각 t msec마다 읽혀져야 한다.
- 코일의 길이값도 매 t msec마다 쓰여져야 한다.
2. Physical Law

[그림 5-10] CoilingTask의 RTO.k 모형

1. 개요

RTO.k 모델을 명확하게 정의하기 위하여 RTO.k 객체를 구성하는 실용적인 언어 구조를 제시한다. RTO.k 클래스의 언어 구조는 직관적인 이해를 용이하게 하기위하여 파스칼(pascal) 언어 형식으로 정의 했으며 궁극적인 개발 대상 언어는 C++ 의 확장이다. 따라서 파스칼 형식의 언어 정의를 먼저 설명하고 이후에 C++ 형태의 확장 언어인 C++T 의 BNF(Backus-Naur Form) 형태의 정의를 제시한다. 아래와 같이 RTO.k 클래스의 정의는 4 개의 실행(executable) 섹션과 한개의 주석(comment) 섹션으로 구성된다.

RTO_class = class

begin

use_section : *list of RTO_name;*

object_data_space_section : *list of object_data_space_segment;*

spontaneous_method_section : *list of spontaneous_method;*

service_method_section : *list of service_method;*

{condition_section : *list of condition;* }

end;

2. 실시간 객체 클래스의 구조 표현

가. Use 섹션

이 섹션은 이 RTO.k 객체내의 메소드에 의해 호출되는 다른 RTO.k 서비스 메소드의 이름들의 리스트를 갖는다.

나. 객체 데이터 공간(ODS) 섹션

이 섹션은 아래에 보여진 바와 같이 하나 또는 그 이상의 ODS 세그먼트 정의들로 구성된다.

{ ODSS <*ODS-segment-name*>
begin
< *ODSS-body*>
end;}*

<ODSS-body>는 유형 정의와 변수 선언을 포함한다.

각각의 ODS 세그먼트(ODSS)는 RTO.k 객체내의 객체 메소드가 read-only 또는 read-&-write 접근 권한을 가지고 있는 저장소이다. ODS 세그먼트에 대해 객체 메소드가 가지고 있는 접근 권한은 객체 메소드의 동시 수행의 가능성을 결정할 수 있으므로 ODS의 세그먼트화는 객체 수행을 실현하는 병행성 정도에 직접적인 영향을 미친다. 동시 수행의 가능성을 동적으로 결정할 수 있도록 하기 위하여 각각의 ODSS의 이름은 실행 엔진내에 등록되어야 한다.

여기서 최대 유효 기간(MVD)은 다음과 같이 각각의 변수에 연관시킬 수 있다.

< type-specifier> < identifier> [during < effective-period> | by < expiration-time>]

예를 들면

"int K during 5 msec"

이것은 정수형 변수 K에 저장된 새로운 값은 단지 5 milli-second 만 유효하다는 것을 나타낸다. 반면

"ts-int K by (time-stamp$1 + 1 hour)"

이것은 타임 스탬프 정수형 변수 K에 저장된 새로운 값은 타임 스탬프의 집합과 연관 되는데 첫번째 타임 스탬프의 값에 1 시간을 더한 것은 K내의 값의 유효성이 종료되는 시간을 나타낸다.

실시간 프로그램내에서 필수적인 요소이면서도 고전적인 비실시간 프로그램에서는 거의 사용되지 않는 변수가 시간 값(timing value)을 갖는 시간(temporal) 변수이다. 다음의 두가지 시간 데이터형이 RTO.k 클래스에 채택 된다.

- 시간 간격(Time-Interval)
- 시간 값(Clock-Value)

다. 자발적 메소드 (SpM) 섹션

SpM 섹션의 전체 구조는 다음과 같다.

```
SpM-name   <name>
    using-services   <RTO-name.SvM-name>*
    using-SpM   <SpM-name>*
    using-data     (<ODS-segment-name>, <access-mode>)*
    [always | if-demanded ]
    do
    { [AAC name:]
          for <time-var> = from <activation-time> to <deactivation-time>
    [every <period>]
          start-during (<earliest-start-time>, <latest-start-time>)
          finish-by          <deadline>
    }*
   [ensure    <acceptance test>]
begin
    <method-body>
    end;
```

(1) "using-services" 섹션은 이 SvM에 의해 호출될 수 있는 다른 RTO.k 객체내의 또는 동일 RTO.k 객체내의 SvM들을 열거한다. SvM에 작용될 수 있는 호출은 두가지 유형이 있다.

● 블로킹 호출 : SvM을 호출한 후 클라리언트는 SvM으로 부터 결과 메시지가 되돌아 올 때까지 기다린다.

● 비블로킹 호출 : SvM을 호출한 후 클라인언트는 SvM으로 부터의 결과 메시지를 기다리지 않고 다음 단계로 진행할 수 있다. 그러나 클라이언트는 SvM으로 부터 결과 메시지를 받을 이 RTO.k 객체내 또는 다른 RTO.k 객체내의 SvM들의 리스트를 SvM

에 제공한다. 이 리스트는 호출 클라이언트를 포함할 수 있다. 리스트는 또한 아무 것도 포함하지 않을 수도 있다.

(2) "using-SpM" 섹션은 이 SpM에 의해 활동 요구(activation request)를 전송받을 수 있는 동일한 RTO.k 내의 SpM의 이름들을 나열한다.

(3) "using-data" 섹션은 이 SpM에 의해 접근될 수 있는 ODS 세그먼트들을 나열한다. 여기서 <*access-mode*>는 ODSS에 접근하는 유형을 규정한다. 즉, read-only 또는 read-&-write 접근등 이다.

(4) "always" 모드는 이 SpM이 AAC 섹션내에 규정된 시간 조건이 맞을 때마다 실행되는 정적 스케쥴링 모드이다. 이것은 디폴트(default) 모드이다. 한편 "if-demanded" 모드는 SpM이 다음과 같은 두가지 조건이 모두 만족할 때에만 실행되는 동적 스케쥴링 모드이다.

- 이 SpM을 활동시키기 위해 동일 RTO.k 객체 내의 다른 메소드로 부터 [t1, t2] 기간동안의 요구가 발생되었고 현재 시각 t는 이 기간 내에 있다.
- AAC 섹션내에 규정되는 시간 조건(do-절에 의해)이 만족된다.

AAC 섹션은 다수의 고유하게 명명되는 AAC 절을 포함한다. SpM을 수행하기 위해 SvM에 의해 예약하는 것에는 요구하는 활동 시간 간격 또는 SpM 정의에 나타난 AAC 절의 이름이 포함되어야 한다. SvM에 의해 요구될 수 있는 (실행시에) 활동 시간 간격이 SpM의 AAC 섹션내에 규정되는 (설계시에) 활동 시간 간격에 포함되는지는 컴파일러에 의해 검사된다. 이것은 양 시간 간격 표현이 간단한 형태가 되어야 함을 의미한다. 이것을 확고하게 하기 위한 한가지 가능한 접근 방법은 다음과 같은 유한한 수의 표현 유형을 규정하는 것이다.

<activation-time>, <deactivation-time>, <period>, <earliest-start-time>, <latest-start-time>, <deadline> 등.

대부분의 경우에 있어서 요구 SvM 은 활동 요구내에 AAC 절의 이름을 포함하여 사용함으로써 SpM 정의내의 AAC 절을 단순하게 지정할 수 있다.

(5) AAC 섹션은 하나 또는 그 이상의 AAC 정의들로 구성된다.

```
for T = from 10.00.000 am to 11.00.000 am every 5.000 msec
    start-during      (T, T+1.000 msec)
    finish-by         T+3.000 msec
```

위의 AAC 는 다음을 규정한다.

"이 SpM 은 10am 에 시작해서 11 am 까지 매 5 msec 마다 수행되어야 하고 각각의 수행은 1 msec 간격(T, T + 1 msec) 내의 어떤 시각에 시작되어야 하며 T + 3 msec 내에 완료되어야 한다."

일반적으로 <earliest-start-time>, <latest-start-time> 및 <deadline> 등은 for 절내에 선언된 <time-var> 의 간단한 산술 표현식이다. 애매함(ambiguity)을 피하기 위해서는 다음과 같은 제한 조건을 따라야 한다.

하나의 시간 간격은 둘이상의 for 절로 규정할 수 없다.

(6) "ensure" 섹션은 수락 테스트(acceptance test)를 포함하는 선택적인 것이다. 즉, 이 SpM 을 수행하여 나오는 계산 결과의 수락 정도를 결정하기 위한 실행 가능한 논리적 표현이다.

(7) <*method-body*> 내에 나타나는 각각의 출력 활동은 마감시간 절을 동반하여야 한다.

라. 서비스 메소드 (SvM) 섹션

SvM 섹션의 전체 구조는 다음과 같다.

```
SvM-name   <name>
    using-services   <RTO-name.SvM-name>*
    using-SpM   <SpM-name>*
    using-data     (<ODS-segment-name>, <access-mode>)*
    [finish-within <duration-limit> | finish-by <deadline>]
  [ensure   <acceptance test>]
    begin
    <method-body>
    end;
```

(1) "using-services" 섹션, "using-SpM" 섹션, "using-data" 섹션 및 "ensure" 섹션은 SpM 에서와 같이 사용되며 여기서도 필수적이 요소가 된다.

(2) SpM 섹션에서 언급한 것과 마찬가지로 SvM 이 수용하여야 하는 두가지의 호출이 있다. 즉, 블로킹 호출과 비블로킹 호출이다. SvM 은 동일한 RTO.k 내의 다른 SvM 에게 서비스 요구를 할 수 있다. 전형적으로 이러한 요구는 SvM 을 종료 시키는 호출로 발생된다. 실제적으로 SvM 은 자기 자신에게 서비스 요구를 발생할 수 있다.

(3) 적시 완료 요구 조건은 두가지 다른 방법으로 규정될 수 있다. SvM 의 실행 시작 시간이 결정될 때 까지는 기간 한계 명세(Duration-Limit Specification)로 부터 메소드 완료 마감시간을 유추하는 것은 가능하지 않다는 것을 인식하여여 한다.

(4) <*method-body*> 에 나타나는 각각의 출력 활동은 마감시간 절을 동반하여야 한다.

마. 조건 섹션

이 섹션은 RTO.k 객체에 부과되는 요구조건에 대한 정보를 포함하기 위한 선택적인 주석 섹션이며 다음과 같은 정보들이 포함된다.

(1) 기능적 요구조건과 신뢰성 (결함 허용, 보안) 요구조건을 포함한 상위 수준의 RTO.k 객체로 부터 상속된 요구조건

(2) 상위 수준 객체내에 규정될 수 없는 결함의 허용 같은 이 RTO.k 객체에 새로 부가된 요구조건

(3) 이 RTO.k 객체의 요소들에 의해 항상 만족되는 불변의 조건

위에서 언급된 정보는 어떠한 형식의 제한 없이 포함될 수 있다.

3. C++T 의 정의

이상에서 설명한 RTO.k 모델의 정의를 C++ 언어의 형식으로 구성하여 최종 사용자도 쉽게 실시간 응용 및 시뮬레이션을 코딩할 수 있게 하여 주는 C++ 언어의 확장으로서 C++T 언어를 제시한다. 이 언어를 이용하여 사용자는 RTO.k 객체를 쉽게 코딩할 수 있으며 설계자가 명세한 RTO 네트워크를 이 언어를 이용하여 간단하게 맵핑(mapping) 하면 실시간 응용이 생성될 수 있게 하는 것이 이 언어의 목표이다. 향후 이 언어의 전처리기(preprocessor)를 개발하여 사용자가 코딩한 객체 모듈을 C++ 코드로 변환하고 이를 컴파일하여 실행 모듈을 생성 하도록 지원할 것이다. 이러한 개발환경은 이 과제에서 추진 중인 실시간 운영체제 및 실시간 시뮬레이션 엔진의 개발과 더불어 실시간 시뮬레이션 응용의 확대에 크게 기여할 것으로 기대된다.

여기서 C++T 언어는 BNF 형태로 정의 하였다. C++T 의 정의는 부록에 나타나 있으며 이는 지속적인 수정 보완 작업을 거쳐 실시간 객체 지향 언어 전처리기를 개발하기 위한 기초로 사용될 것이다.

7 절 실시간 시뮬레이션 엔진의 설계

1. 개요

실시간 컴퓨터 시스템를 구성하기 위한 기본적 요소는 적시성을 보증하여 주는 운영체제(Timeliness-Guaranteed operating system)이다. 이것은 하드웨어 플랫폼과 함께 실행 엔진을 구성하고 있으며, 병행 분산 실시간 응용 소프트웨어에 적시성 서비스를 제공한다. 만약 운영체제가 이러한 적시성 서비스 능력을 보증하지 못하면 실시간 응용의

적시성은 보장될 수 없다.

본 과제의 국제공동 연구 파트너인 Kane Kim 은 적시성 서비스를 보증하는 실시간 프로세스를 지원하는 운영체제 커널(kernel)의 모델을 제안하였다. 이 모델은 DREAM(Distributed Real-Time Ever Available Micro-Computing) 커널이라고 불린다. 이 커널은 실시간 객체 지향 구조 모델(RTO.k)의 접근 방법을 지원하는 실행 엔진을 개발하기 위한 과정의 산물로서, RTO.k 객체 지향 구조는 차세대 실시간 컴퓨팅 환경을 구축하기 위함을 목적으로 한다.

이상적인 차세대 실시간 컴퓨팅을 실현하기 위하여는 모든 실용적이고 유용한 실시간 및 비실시간 컴퓨팅 요구사항을 처리할 수 있는 구조 모델을 구축하여야 한다. 이러한 구조적 방법은 모듈화, 일반화 및 자연스런 추상화의 장점을 고려할 때 객체 지향형이 바람직 하다고 판단된다. 지난 몇년간 전통적인 객체 지향 구조를 확장하여 실시간 컴퓨터 시스템에 적용하려는 많은 시도가 있었다. 그러나 이들 시도의 대부분은 객체의 적시 서비스 능력의 설계 시각 보증을 지원하려는 것을 목표로 하는 것이 아니었다. 그러나 이것은 차세대 실시간 컴퓨팅의 기본적인 요구사항의 하나이다.

RTO.k 객체 구조 모델은 전통적인 객체 지향 구조의 확장이지만 기본적인 구조의 여러가지 고유한 특성을 가지고 있다. RTO.k 객체의 실행기로서 DREAM 커널은 RTO.k 객체와 다양한 하드웨어 구조간에 탄력적인 연결이 가능하도록 설계되었다. 커널은 다음을 지원함으로써 이것을 가능하게 한다.

- 다양한 구동(activation)과 동기화(synchronization) 요구 조건을 갖는 실시간 프로세스
- 공유 데이터 구조 감시기(monitor)
- 실시간 다중전송 논리(Real-Time Multicast Logical, RML) 채널

이 커널은 RTO.k 객체의 구성 요소들을 차례로 수행한다. 그러므로 DREAM 커널은 프로세스 구조의 실시간 응용과 RTO.k 객체 구조 응용을 모두 지원할 수 있다. DREAM 커널을 구성하는 가장 중요한 요소는 하드웨어 사용의 손실을 최소한으로 줄이면서 적시 서비스 능력 보증을 구현하는 것이다. 따라서 DREAM 커널은 범용성 있는 적시성 보증 OS 커널 모델이라고 할 수 있다.

2. 설계 구조

가. 프로세스 구조 실시간 병행 분산 프로그램 기본 구성요소

프로세스 실행 엔진으로서의 DREAM 커널은 다음과 같은 세가지 유형의 병행 분산 프로그램(Concurrent and Distributed Program) 요소들을 지원한다.

(1) 프로세스 : 여기서는 응용 프로세스(Application Process, AP)라고도 부르며, I/O 관리 같이 이들 프로세스는 시스템 관리 프로세스의 역할을 하기도 한다. 프로세스는 그들의 다음 계산 세그먼트의 수행을 위해 커널상에 마감시간(deadline)을 부과하기도 한다.

(2) CREW (Concurrent-Read-&-Exclusive-Write) 감시기(monitor) : CREW 감시기는 공유 데이터 구조 감시기 이며 Brinch 가 정의한 감시기의 확장이다. 즉, Brinch 의 감시기는 배타적 읽기 및 배타적 쓰기 의미구조(Exclusive-Read-&-Exclusive-Write Semantics)를 가진데 비해 이것은 읽기 쓰기 의미구조(Readers-Writers Semantics) 즉 동시 읽기 및 배타적 쓰기 의미구조(Concurrent-Read-&-Exclusive-Write Semantics)를 가지고 있다.

(3) HU-DF 구조(Scheme)내의 내용 코드 채널(Content-Code (CC-) channel) : 프로세스간의 그룹 통신(Inter-Process-Group Communication)을 위한 HU-DF 구조는 Mori 에 의해 개발된 데이터 필드 구조(Data Field Scheme)의 확장이다. 데이터 필드 구조의 핵심은 다중 전송 논리 채널(Multicast Logical Channel) 의 생성 및 논리 채널에 대한 프로세스의 동적 연결(Dynamic Connection)을 물리적 통신 네트워크의 특이성으로 부터 프로세스 설계자에게 투명성을 제공해 주는 것이다. 만약 물리적 통신 장치가 방송(broadcast) 능력을 가진다면 논리적 다중 채널은 모든 프로세싱 노드가 채널을 사용하여 내용 코드라고 불리는 채널의 식별자(ID)를 포함하는 헤더(header)를 가진 메시지를 방송(물리적 통신 장치를 통해) 하게 한다. 논리 채널에 연결된 프로세싱 노드들은 물리적 방송 장치를 통해서 들어 오는 모든 메시지를 볼 수 있으나 관련된 내용 코드를 포함하는 메시

지만을 주의한다. HU-DF 구조는 기존의 데이타 필드와는 다른데 즉, 이것은 논리 채널에 프로세스를 동적, 탄력적으로 연결시켜주며 전통적인 사건 메시지(Event Message) 뿐만 아니라 분산 중복 메모리 의미구조에 비탕을 둔 상태 메시지(State Message)도 지원한다.

이러한 세가지의 기본 요소는 병행 분산 프로그램 구조에 대한 완전한 일반 장치를 표현한다. 그러므로 다양한 I/O 조작과 함께 이러한 세가지 요소를 지원하는 어떠한 운영 체제도 일반적인 목적의 커널로 볼 수 있다. 세가지 유형의 요소들로 구성되는 프로그램들은 PCD(Process-CREW-DF) 프로그램이라고 불린다.

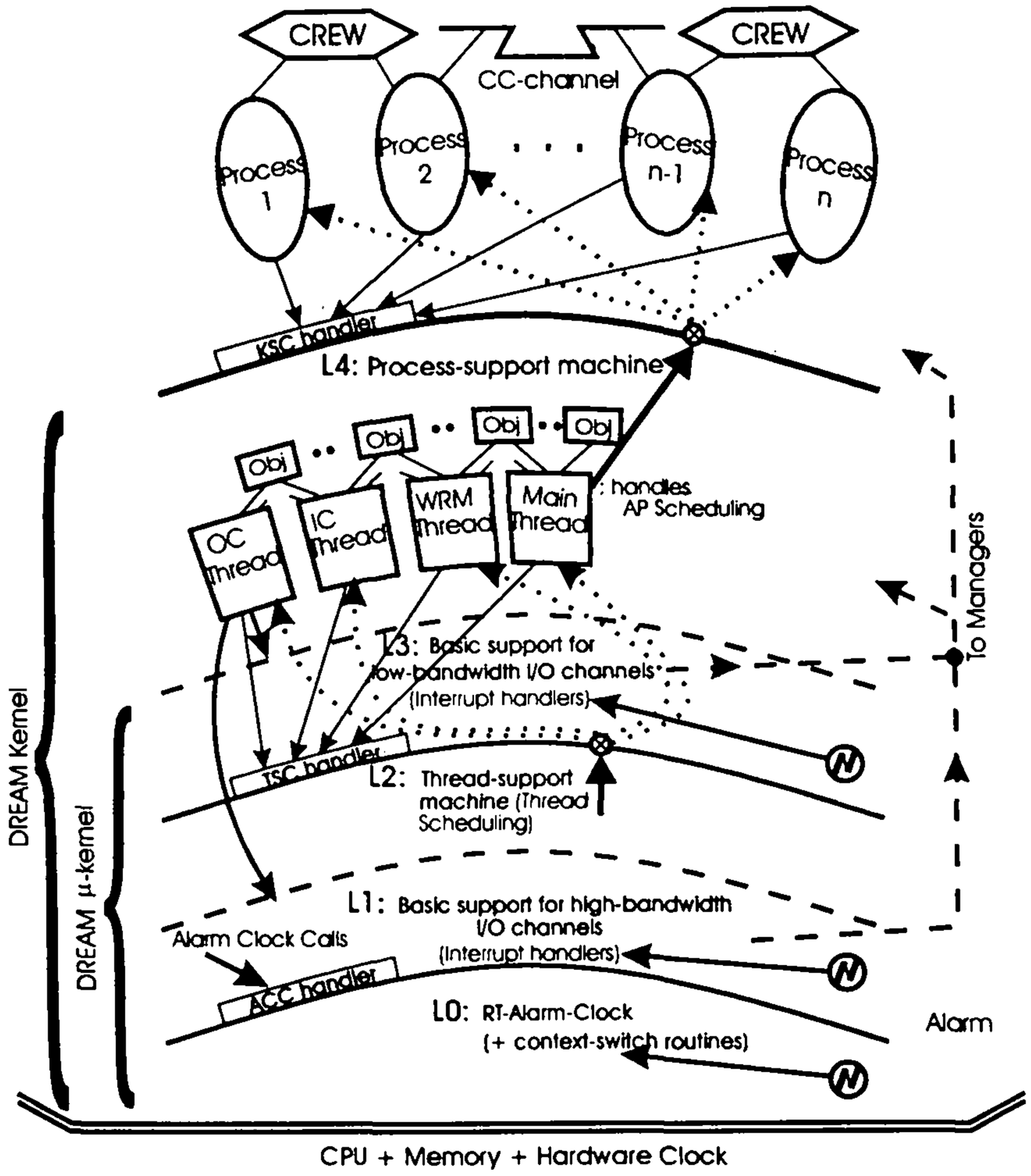
나. 5 계층 구조와 시간 차용 구조

DREAM 커널의 구조는 [그림 7-1]과 같다.

[그림 7-1]에서 볼 수 있는 바와 같이 DREAM 커널은 구성요소들을 계층화한 독특한 접근 방법을 채택하였다. 이러한 특별한 계층 방법은 보증된 적시 서비스 능력을 실현하는 중요한 열쇠가 된다. 커널은 전체적으로 5 개 계층으로 구성되어 있고 5 개 계층의 하단 4 개는 DREAM 마이크로 커널(Micro-Kernel)을 구성한다. DREAM 커널은 프로세스 실행 엔진(Process Execution Engine)으로 볼 수 있으며 DREAM 마이크로 커널은 커널 쓰레드 실행 엔진(Kernel-Thread Execution Engine)으로 볼 수 있다.

커널 쓰레드(줄여서 쓰레드)는 DREAM 커널내에서 실행되는 동적인 동시성 단위(Active Concurreny Unit)이다. 쓰레드의 집합은 운영체제 적재(load)시에 고정된다. 모든 쓰레드는 동일한 주소 공간을 공유한다. 메인 쓰레드(Main Thread, MT)는 프로세스의 선택때마다 수행되어야 할 다음 프로세스를 선택하고 실행시키며, 그외의 모든 쓰레드는 주기적 쓰레드인데 이것은 RTO.k 객체에서의 주기적인 시간 구동 장치와 같이 행동한다.

쓰레드가 수행을 위하여 선택되면 쓰레드 시간 단편(Thread Time-Slice)이라고 불리는 한 시간 단위로 수행될 수 있다. 실행을 위하여 선택된 주기적 쓰레드가 완전한 쓰레드 시간 단편을 필요로 하지 않는다면 이것은 시간 단편의 나머지 부분을 MT에게 돌려 줄(donate) 수 있으며 또한 나머지 부분의 크기에 따라 일하지 않고(idle) 없어질 수도 있다. 그러므로 쓰레드 스케쥴링은 프로세스 스케쥴링과는 달리 간단하고 낮은 오버헤드를 필요로 하는 운영이다. MT를 제외한 모든 커널 쓰레드를 이러한 방식 즉, 시간당 하나의 쓰레드 시간 단편을 사용한 주기적 쓰레드로 제한한 것은 하드웨어 사용을 줄이면서 각각의 쓰레드의 최악의 경우의 응답 시간의 해석을 가능하게 하는 잘 정의된 접근 방법이라고 믿는다.



[그림 7-1] DREAM 커널의 구조

[그림 7-1]에 나타난 계층 구조는 시간 차용 구조 계층(Time-Leasing Machine Layering) 이라고 불리는 조직적인 원리에 바탕을 두고 있다. 이 원리는 보증된 적시 서비스 능력을 가진 커널을 획득할 수 있는 중요한 것이다. 이러한 원리 밑에서 하위 계층 L0는 하드웨어의 모든 능력을 가진다. 그래서 L0는 필요에 따라 하드웨어를 사용한다. L0는 실시간 경보 클럭(Real-Time Alarm Clock)을 포함하는데 이것은 요구를 받아들이

는데 대해서 현재의 시각을 제공하고 또한 "기상 호출(Wake-Up Call)" 서비스를 제공한다. L0 가 하드웨어를 사용하고 난 후의 하드웨어 시간의 나머지는 상위 계층 L1 에 빌려준다. L1 은 L0 로 부터 받은 하드웨어 시간의 한 부분(즉, L0 가 사용하지 않은 기계 시간)을 사용한다. 마찬가지로 L2 는 L0 와 L1 이 사용하지 않은 하드웨어 시간의 한 부분을 사용한다. 이러한 재귀적인 시간 차용(Recursive Time Leasing) 관계는 [그림 7-2]에 표시된 것과 같이 L4 에 까지 적용된다.

DREAM 커널의 상위 4 계층은 다음과 같은 요소들을 포함한다.

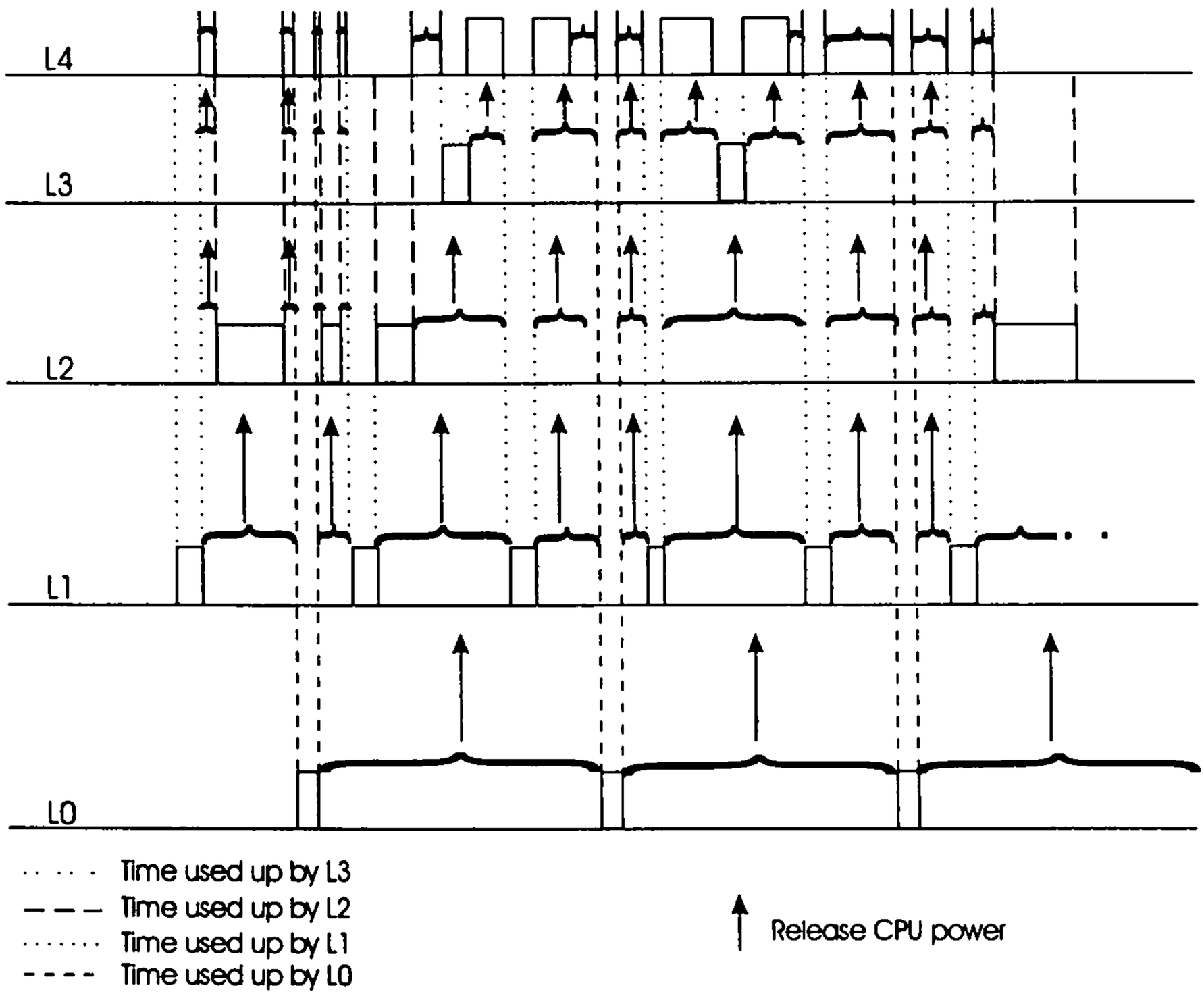
(1) L1 은 LAN 접속과 같은 높은 대역폭(bandwidth)의 I/O 에 대한 기본적인 지원 기능을 포함한다.

(2) L2 는 쓰레드 시간 단편 및 다른 시간들의 종료에 맞추어 구동되는 쓰레드 스케쥴러를 포함한다.

(3) L3 는 직렬 문자 I/O 와 같은 낮은 대역폭의 I/O 에 대한 기본 지원 기능을 포함한다

(4) L4 는 프로세스 스케쥴러와 프로세스에 대한 다른 지원 기능, CREW 감시기 및 CC 채널등을 포함한다. 프로세스 스케쥴러는 프로세스 시간 단편 및 다른 시간들의 종료시에 구동된다. 여기서의 프로세스 시간 단편의 크기는 쓰레드 시간 단편의 크기와는 다르다.

시간 차용 구조 계층 원리의 다른 중요한 부분은 기본적인 응답 기간동안 어떠한 계층에서 사용된 하드웨어 시간의 양이라도 일정한 한계(threshold)이내로 제한된다는 것이다.



[그림 7-2] 시간 차용 구조 계층

여기서의 기본적인 응답 기간은 커널 서비스에 대한 프로세스 호출 시각과 서비스가 완료되는 시각간의 최대 간격을 지칭한다. 이 원리를 채택하는 명백한 목적은 각각의 상위 계층에서 사용가능한 기계 시간의 일정양을 보증하기 위한 것이다.

L0 에 의해서 사용된 기계시간의 양은 일정하며 기본 응답 기간동안에 사용되는 L0 의 최대치는 쉽게 계산될 수 있다. 기본 응답 기간동안에 L2 및 L4 가 사용하는 최대치도 무난히 계산될 수 있다. 그러나 외부의 자원으로 부터의 인터럽트에 반응해야 하는 L1 과 L3 의 경우는 다르다. 원칙적으로는 L1 의 LAN 접속 관리자는 수신 메시지 버스트(burst)에 기인하는 LAN 접속에 의해 생성되는 인터럽트 버스트를 받는다. L1 의

이러한 조건하에서는 L4에 의해 프로세스에 전달하는 어떠한 서비스도 적시성을 보증하는 것은 불가능 하다는 것을 발견하였다. 그렇다면 L1에 의한 인터럽트 생성의 주기는 통제되어야 하고 일정한 한계를 초과하지 않도록 지정되어야 한다. 이것은 주기가 한계에 도달하면 인터럽트 주기가 한계를 벗어나게 하는 LAN 접속에 의한 더 이상의 인터럽트는 억제(disable)되어야 한다는 것을 의미한다. 이것은 다시말하면 어떤 메시지들은 손실된다는 것을 의미한다. 이것은 모든 서비스들을 프로세스들에 적시에 전달함을 보증하기 위한 댓가이다. 시간 차용 구조 계층 원리의 한 부분인 이러한 인터럽트 자원의 동적 제어(Dynamic Control of Interrupt Sources)는 어떠한 적시성 보증 운영체제에서도 피할 수 없는 요구조건이다.

다. 커널 쓰레드

위에서 언급한 것과 같이 MT를 제외한 커널 쓰레드는 한번에 한 쓰레드 시간 단편을 사용하는 주기적인 쓰레드이다. DREAM 커널의 기본적인 단일 프로세서 버젼에서는 다음과 같은 4개의 필수적인 커널 쓰레드가 있다.

(1) OCT (Outgoing Communication Thread) : 통신 네트워크를 통한 메시지의 송신을 관리한다.

(2) ICT (Incoming Communication Thread) : 통신 네트워크를 통해 수신된 메시지를 목적 프로세스에 배분하는 것을 관리한다.

(3) WRMT (Watchdog-&-RTO-Management Thread) : 객체 메소드의 구동을 관리하고 마감시간을 벗어나는지를 점검한다.

(4) MT (Main Thread)

만약 DREAM 커널이 단지 프로세스 실행 엔진으로서 동작하고 RTO.k 객체의 실행 엔진이 아니라면 WRMT의 단지 한 부분 즉, 감시 타이머 서비스(Watchdog Timer Service)만이 필요하다.

고대역폭 장비로 부터의 데이터 입력은 L1 의 인터럽트 관리자와 L4 의 ICT 등의 장치의 조합된 노력을 통해 실현된다. B1 으로 불리는 적당한 크기의 버퍼가 있는데 고대역폭 장치는 데이터를 여기에 완전히 찰때까지 넣는다. 버퍼 B1 이 차면 장치는 인터럽트를 발생한다. 인터럽트에 대응해서 L1 의 인터럽트 관리자는 B1 에 있는 데이터를 인터럽트 관리자 및 ICT 가 접근 가능한 더 큰 버퍼 B2 로 이동시킨다. 인터럽트 관리자는 ICT 가 B2 에 접근할 준비를 갖추기 전에 많은 데이터들을 B2 에 삽입할 수 있다. 이후 ICT 가 다음 쓰레드 시간 단편을 가지면 B2 의 데이터는 목적지 AP 로 이동 시킨다.

여기서 주목하여야 할 것은 만약 응용에 관련되어 빠른 응답 활동이 지원되어야 한다면 특별한 응용에 관련된 쓰레드를 도입할 수 있다는 것이다. 그러나 모든 커널 쓰레드의 집합은 어느정도의 성능 수준을 유지하면서 커널의 적시 서비스 능력을 보증함을 손상받지 않기 위해서는 운영체제 적재시에 고정되어야 한다.

라. 쓰레드와 쓰레드간의 원소 섹션

MT 를 제외한 커널 쓰레드를 한번에 한 쓰레드 시간 단편을 사용한 주기적 쓰레드로 제한한 것은 쓰레드 스케쥴링의 오베헤드를 줄이고 이러한 오버헤드를 쉽게 해석할 수 있게 한다. 커널 쓰레드의 최악의 경우의 응답시간을 단순화 하기 위하여 취하는 부가적인 조치는 DREAM 커널내의 커널 쓰레드에 대하여 데이터에 대한 잠금(lock)을 사용하지 않고 공유 데이터을 접근하는데 있어서 그들간의 충돌을 막기 위해 제약을 가하는 것이다. 대신 공유 데이터 구조를 접근하는데 필요한 쓰레드는 공유 데이터 구조를 방해하지 않기 위해서 쓰레드 지원 기계(L2 에 있는)를 호출한다. 즉, L2 를 호출하여 쓰레드 스위치를 억제하는데 이것은 쓰레드로 부터 기계의 자원을 빼앗지 않기 위함이며 쓰레드가 L2 에게 방해 금지 기간(Disturbance Prohibition Period)이 끝났음을 알

려줄 때까지 지속된다 (이것은 쓰레드가 공유 데이터 구조에 접근함을 끝낼때 명백히 발생하는 것이다). 그래서 방해 기간동안에 쓰레드 시간 단편이 종료될 지라도 (즉, 쓰레드가 공유 데이터를 접근하는 동안) 쓰레드 지원 기계는 수행 쓰레드를 변경시키지 않는다. 쓰레드가 공유 데이터 구조를 접근하는 수행 기간동안의 이러한 코드 세그먼트를 쓰레드 쓰레드간 원소 섹션(Thread-to-Thread Atomic Section, TT-AS)이라고 부른다. 방해하지 않는 요구를 쓰레드 지원 기계에 전송하는 것을 "TT-AS 에 들어간다"고 한다. 마찬가지로 방해하지 않는 요구를 취소하는 것을 "TT-AS 에서 나온다"고 한다.

일반적으로 TT-AS 접근방법은 쓰레드가 원소 섹션(AS)을 수행하는 기간이 쓰레드 시간 단편보다 훨씬 작을 때에만 작동한다. 그렇지 않으면 쓰레드 시간 단편이 잘못 선택되었거나 쓰레드와 그의 공유 데이터 구조가 적절히 설계되지 않았을 때 작동한다. 이러한 TT-AS 접근방법은 전통적인 데이터 잠금을 기반으로 하는 접근방법에 비해 더욱 효율적이다. 왜냐하면 MT를 제외한 커널 쓰레드는 주기적인 쓰레드이고 TT-AS 접근방법 보다 데이터 잠금 접근방법하에서 실행중인 쓰레드의 빈번한 변경이 발생할 수 있기 때문이다.

마. 2 수준 스케쥴링

DREAM 커널에서 처리하는 AP 와 커널 쓰레드의 두가지 유형의 병행 실행 단위가 있는데 병행 실행 단위의 스케쥴링은 두개의 다른 커널 계층에서 발생한다. 프로세스 시간 단편은 쓰레드 시간 단편보다 작을 수 없다. 일반적으로 프로세스 시간 단편은 쓰레드 시간 단편의 정수배가 된다.

L2(쓰레드 지원 기계)에서 쓰레드 스케쥴러는 하드웨어 머신을 사용하기 위하여 네 개의 쓰레드 ICT, OCT, WRMT 및 MT를 스케쥴한다. 그리고 L4 에서는 MT 가 이것의 핵심(AP 가 아닌 AP 스케쥴러)을 수행할 때 하드웨어 머신을 사용하기 위하여 다양한

AP 들을 스케쥴한다. AP 가 자발적으로 하드웨어 머신에 제어를 돌려주거나 프로세스 시간 단편의 종료를 명령하면 제어는 다른 쓰레드(MT 가 아닌) 또는 MT 의 핵심(AP 스케쥴러)으로 가게된다.

또한 TT-AS 의 사용과 유사한 방법으로 AP 들은 프로세스 프로세스간 원소 섹션(PP-AS)을 사용할 수 있다. 그러므로 AP 간의 동기화는 세가지 다른 장치로 실현될 수 있다. 즉, PP-AS, 세마포어(semaphore) 및 CREW 감시기등이다. 커널 쓰레드의 사용에서와는 달리 AP 들의 동적인 생성은 DREAM 커널에 의해 지원된다.

이러한 2 수준의 스케쥴링 접근방법의 장점은 다음과 같다. AP 스케쥴링에 의해 어떻한 방법으로도 접속되지 않는 형태의 커널 쓰레드의 스케쥴링은 주기적인 커널 쓰레드가 설계시각에 결정되는 시각에 정확하게 AP 에 필수적인 서비스를 제공함을 보증하기가 용이하여 진다. 또한 AP 간의 중개자로서 MT-Core 의 설계는 구현의 복잡성을 상당히 감소하며 직접적인 AP-to-AP 컨텍스트 스위치(Context Switch) 설계에 대한 컨텍스트 스위치 오버헤드를 어느정도 줄여준다.

바. RTO.k 객체의 PCD 프로그램에의 대응

DREAM 커널이 RTO.k 객체를 실행할 때 이것은 실제로 이것과 동등한 PCD 프로그램을 실행한다. 즉, 프로세스, CREW 감시기 및 데이터 필드(CC 채널)로 구성된 프로그램을 실행한다. [그림 7-3]은 RTO.k 객체의 요소들과 PCD 프로그램의 대응되는 요소들을 대응한 관계를 보여준다.

[그림 7-3]에서 보는 것과 같이 객체 메소드는

(1) SpM 과 SvM 은 모두 프로세스에 대응된다.

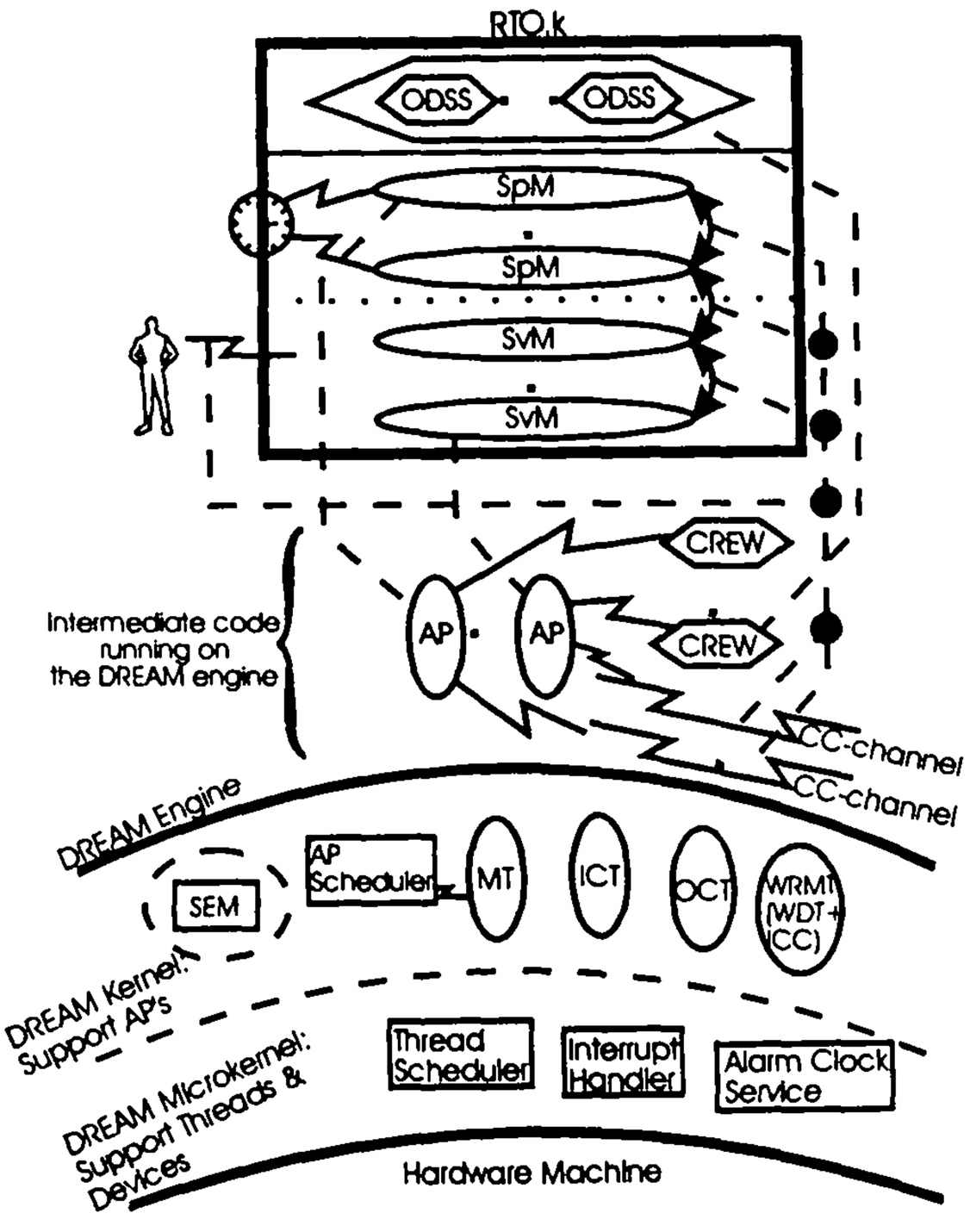
(2) ODS 세그먼트(ODSS)는 CREW 감시기에 대응된다.

(3) SvM 에 대한 접근 경로는 CC 채널을 통한다.

(4) 클라이언트에 결과를 보내는 경로는 CC 채널을 통한다.

CC 채널을 이용하는 이 방법은 객체 위치에 대한 투명성을 제공한다.

SpM 이 프로세스에 대응되면 프로세스는 DREAM 커널에 SpM 의 AAC, 관련된 마감시간 명세 및 ODSS 에 대한 SpM 의 접근 권한을 제시해야 한다. 마찬가지로 SvM 이 프로세스에 대응되면 프로세스는 DREAM 커널에 관련된 마감시간 명세, ODSS 에 대한 SvM 의 접근 권한 및 파이프라인(pipeline) 수행 가능성에 관련된 기타 정보를 제시해야 한다.



[그림 7-3] RTO.k 객체의 PCD 프로그램에의 대응

이전에 언급한 것과 같이 RTO.k 객체를 실행하는 이런 접근방법 즉, 상응한 PCD 프로그램에 객체를 대응하는 방법은 RTO.k 객체와 다양한 하드웨어 구조간의 탄력적인 연결을 하여 DREAM 커널로 하여금 다양한 유형의 병행 분산 응용을 지원하는 범용적인 커널이 되게 하는 장점을 가지고 있다.

8 절 결론

실시간 시뮬레이션은 최근 들어 급격히 보급되고 있고 관심도 급증되고 있는 분야이다. 선진국에서도 그동안 여러 응용 분야에서 제품을 개발하였지만 아직까지도 확립된 기반 기술이 없는 실정이다. 이 분야는 그래픽, 애니메이션 및 멀티미디어와 접속하여 급속히 신장하고 있고 응용 분야가 어디까지에 미칠 것이가를 판단하기 힘든 상황이다. 국내의 실정은 대부분의 시스템을 수입에 의존하고 있고 이에 대한 기술 수준은 아직까지 기초 수준에 머물고 있는 실정이다.

이러한 상황에서 선진국에 대한 경쟁력을 확보하는 길은 이 분야의 기반 기술을 확보하고 이를 바탕으로 제품을 개발하여야 하는 것이다. 이러한 상황을 감안 할 때 본 과제는 매우 적절한 시기에 적절한 시도를 한 것이라 볼 수 있다. 특히 국제적으로 인정 받고 있는 국제 공동 연구자와의 협동 연구라는 형태가 우리로서는 좋은 기회라고 할 수 있다. 우리의 최종 목표는 실시간 시뮬레이션 개발환경 구축이며 지능형 생산공장의 미시적 시뮬레이터 구축이다.

본 과제의 1 차년도는 최종 목표를 달성하기 위한 사전 조사와 개발 접근 방법인 실시간 객체 지향 모델링 기법의 정립에 중점을 두어 추진하였다. 국내의 실시간 시뮬레이터의 현황을 파악하였고, 실시간 모델링 기법을 정립한 것은 실시간 연구 개발의 방

향을 올바로 잡은 것으로 그 의의가 크다고 하겠다. 이 모델링 기법은 실시간 시뮬레이션 통합 개발 환경으로 발전을 하여야 한다.

통합 개발 환경 구축의 일환으로 실시간 객체 지향 언어를 정의하였으며 이 모델의 시간 제약성을 보증해 줄 수 있는 실시간 시뮬레이션 엔진의 설계 또한 큰 의의를 갖는다고 하겠다. 이 모델링 기법을 이용한 미시적 실시간 시뮬레이션의 적용은 이 모델의 성능을 확인 할 수 있으며 이는 계속적으로 개발하여야 할 과제이다.

2 차년도에는 1 차년도에서 정립한 모델링 기법을 기반으로 하여 실시간 객체 지향 언어 정의에 의한 전처리기를 개발하고 실시간 시뮬레이션 엔진을 개발할 것이며 실시간 데이터 베이스를 설계하고 1 차년도에 설계한 시뮬레이터 응용을 심화하여 미시적 시뮬레이션 모델을 구축할 것이다. 본 과제의 최종 목표는 실시간 시뮬레이션 개발환경 구축 및 지능형 생산공장 시뮬레이터이며 특히 실시간 개발환경 구축에 많은 노력을 기울일 것이다.

참고문헌

[1] 한국정보과학회, "특집: 이산 사건 시뮬레이션", 정보과학회지, 제 13 권, 제 4 호, 1995 년 4 월

[2] 포항공대 정부통신연구소, "광양제철소 P/C 분야 Data Processing 표준 기능 구축", 연구보고서, 1994 년 12 월

[3] 포스콘 기술 연구소, POSCON 기술보, 제 1 권 제 2 호, 1995 년 7 월

[4] 한국에너지 연구소, 교육훈련용 Nuclear Simulator 개발, 연말보고서, 1988

[5] 김성일, 원자력 발전소 운전원 훈련용 시뮬레이터 운영, 제 2 회 모의 훈련 체계 세미나 및 전시회, 1994

[6] Yong-Kwan Lee et al., KEPCO's 3-pack simulator development plan, '95 SCS conference, 1995

[7] K.H. (Kane) Kim, Luiz Bacellar, Yuseok Kim, "Distinguishing Features and Pontential Roles of the RTO.k Object Model", Proceedings of WORDS'94(The 1st IEEE Computer Society's Workshop on Object-Oriented Real-Time Dependable Systems), pp. 36-45, Dana Point, USA, Oct. 1994.

[8] K.H.(Kane) Kim, Hermann Kopetz, "A Real-Time Object Model RTO.k and an Experimental Investigation of Its Potentials", Proceedings of COMPSAC'94 (The 18th IEEE Computer Society's International Computer Software & Application Conference), pp. 392-402, Taipei, Taiwan, Nov. 1994.

[9] K.H.(Kane) Kim, Luiz Bacellar, Yuseok Kim, Chittur Subbaraman, Hankil Yoon, Jungguk Kim, "A Timeliness-Guaranteed Kernel Model - DREAM Kernel - and Implementation Techniques", To appeare in Proceedings of RTCSA '95 (The 2nd Int'l Workshop on Real-Time Computing Systems and Applications), Tokyo, Japan, Oct. 1995.

[10] Alan Burns, Andy Wellings, "Real-Time Systems and Their Progrmming Languages", Addison-Wesley, 1990.

[11] A. Alan B. Pritsker, "Introduction to Simulation and SLAMII", John Wiley & Sons, 1993.

[12] Alan Finn, Randall Decker, Chris McClurg, Dayle Harmon, "Simulation of Multiple Access Protocols for Real-Time Control", Simulation, pp123-130, Feb. 1992.

[13] Ningjian Huang, Ka C. Cheok, Thomas G. Horner, Timothy Settle, "Real-Time Simulation and Animation of Suspension Control System Using TI TMS320C30 Digital Signal Processor", Simulation, pp405-416, Dec. 1993.

[14] Bernard P. Zeigler, Jinwoo Kim, "Extending the DEVS-Scheme Knowlege-Based Simulation Environment for Real-Time Event-Based Control", IEEE Tras. on Robotics and Automations, pp351-359, June 1993.

[15] Daniel T. Wick, Nagy M. Shehad, Ankur R. Hajare, "A Real-Time Simulation Facility for Testing the Telerobotic Assembly of the Space Station", Proceedings of the 1993 Simulation Multiconference on the International Simulators Conference, pp429-433, Mar. 1993.

[16] Brian Kerridge, "System Simulation Enbraces Real-Time Control Prototyping", EDN, pp49-57, May 1994.

[17] Paul Rogers, Maureen T. Flanagan, "On-line Simulation for Real-Time Scheduling of Manufacturing Systems", Industrial Engineering, pp37-40, Dec. 91.

[18] Roderic C. Deyo, "An Example of Real-Time Simulation: Multi-body Vehicles", Real-Time Integration Methods for Mechanical System Simulation, NATO ASI Series Vol. F69, pp3-31, 1990.

[19] Carl K. Chang, Young-Fu Chang, Mikio Aoyama, "A Real-Time Distributed Simulation of PBX with Software Reuse", Simulation, pp71-79, Feb. 1990.

[20] Tony Lee, Sumit Ghosh, "A Distributed Approach to Real-Time Payments-Processing in a Partially-Connected Network of Banks: Modeling and Simulation", Simulation, pp180-201 Mar. 1994.

[21] Greg Lomow, Dirk Baezner, "A Tutorial Introduction to Object-Oriented Simulation and Sim++", Procedings of the 1991 Winter Simulation Conference, pp157-163, 1991.

[22]Averill M. Law, W.David Kelton, Simulation Modeling & Analysis, McGraw-Hill, 1991, pp234-266

[23] James J. Swain, Flexible Tools for Modeling, OR/MS Today, Dec. 1993

부 록

부록 1. 시뮬레이션 특성 분류표

Simulation Methods/Tools Classification

Extending the DEVS-scheme Knowledge-Based Simulation Environment for Real-Time Event-Based Control (1993)

(Bernard P. Zeigler and Jinwoo Kim)

Feature	Degree
o Real-time simulation capability * (Use of physical real-time clock)	Yes
o Distributed and parallel processing * (No distributed execution, Distributed over LAN only, Distributed among node in a HPM, Distributed over LAN and among nodes in a HPM)	No distribution
o Hardware requirements - General : machine type and capacity - Special : machine type and capacity	PC-386 or Host
o Use of real measured data or randomly synthesized data only *	Real Measured data
o Number of states maintained * > 1M (high fidelity) > 100K ∧ < 1M (medium fidelity) < 100K (low fidelity))	HF possible
o Base language for specifying simulation models *	C
o System software requirements - Language tools - OS	
o Object orientation or not * (Simulation model)	
o Developer	Zeigler

o Development history	Extension of DEVS-scheme
o Ease of stepwise refinement & modular expansion *	Appropriate
o Simulation output * - Graphic . Animation . 3D - Statistics - Other media output	
o DB interface	No
o Reusable library - Simulation models - Simulator modules	
o Special user interface * - Domain specific interface - Dynamic user input - Graphic input - Other media input	

(Method : * , Tool : o)

A Real-Time Distributed Simulation of PBX with Software Reuse (1990)

(Carol K. Chang, Young-Fu Chang and Mikio Aoyama)

Feature	Degree
o Real-time simulation capability * (Use of physical real-time clock)	Yes
o Distributed and parallel processing * (No distributed execution, Distributed over LAN only, Distributed among node in a HPM, Distributed over LAN and among nodes in a HPM)	Distributed over LAN
o Hardware requirements - General : machine type and capacity - Special : machine type and capacity	VAX 11/780 SUN 3
o Use of real measured data or randomly synthesized data only *	Manually & accumulated in various event queues according to the attributes of these events
o Number of states maintained * > 1M (high fidelity) > 100K ∧ < 1M (medium fidelity) < 100K (low fidelity))	
o Base language for specifying simulation models *	C
o System software requirements - Language tools - OS	UNIX 4.2 BSD, IPC
o Object orientation or not *	

(Simulation model)	Object Oriented
o Developer	Univ. of Illinois at Chicago
o Development history	
o Ease of stepwise refinement & modular expansion *	Appropriate
o Simulation output * - Graphic . Animation . 3D - Statistics - Other media output	2D Graphic
o DB interface	No
o Reusable library - Simulation models - Simulator modules	Yes
o Special user interface * - Domain specific interface - Dynamic user input - Graphic input - Other media input	Graphic input

(Method : * , Tool : o)

Real-Time Simulation and Animation of Suspension Control System Using TI TMS320C30 Digital Signal Processor (1993)

(Ningjian Huang, Ka C. Cheok, Thomas G. Horner and Timothy Settle)

Feature	Degree
o Real-time simulation capability * (Use of physical real-time clock)	Yes
o Distributed and parallel processing * (No distributed execution, Distributed over LAN only, Distributed among node in a HPM, Distributed over LAN and among nodes in a HPM)	No distribution
o Hardware requirements - General : machine type and capacity - Special : machine type and capacity	PC-386(33Mz) with SVGA supported by DSP board
o Use of real measured data or randomly synthesized * data only	
o Number of states maintained * > 1M (high fidelity) > 100K ∧ < 1M (medium fidelity) < 100K (low fidelity))	
o Base language for specifying simulation models *	C
o System software requirements - Language tools - OS	
o Object orientation or not * (Simulation model)	No
o Developer	Oakland Univ. and TI

o Development history	
o Ease of stepwise refinement & modular expansion *	Possible to modular expansion
o Simulation output * - Graphic . Animation . 3D - Statistics - Other media output	Animation
o DB interface	No
o Reusable library - Simulation models - Simulator modules	
o Special user interface * - Domain specific interface - Dynamic user input - Graphic input - Other media input	Dynamic user input

(Method : ♦ , Tool : o)

Simulation of Multiple Access Protocols for Real-Time Control (1992)

(Alan Finn, Randall Decker, Chris McClurg and Dayle Harmon)

Feature	Degree
o Real-time simulation capability * (Use of physical real-time clock)	Yes
o Distributed and parallel processing * (No distributed execution, Distributed over LAN only, Distributed among node in a HPM, Distributed over LAN and among nodes in a HPM)	Distributed over LAN only
o Hardware requirements - General : machine type and capacity - Special : machine type and capacity	Hard-wired and microprocessor based
o Use of real measured data or randomly synthesized * data only	
o Number of states maintained * > 1M (high fidelity) > 100K ∧ < 1M (medium fidelity) < 100K (low fidelity))	
o Base language for specifying simulation models *	Pascal and C
o System software requirements - Language tools - OS	BONES Simulation Tool
o Object orientation or not * (Simulation model)	No
o Developer	United Technologies Research Center

o Development history	
o Ease of stepwise refinement & modular expansion *	
o Simulation output * - Graphic . Animation . 3D - Statistics - Other media output	Graphic
o DB interface	No
o Reusable library - Simulation models - Simulator modules	No
o Special user interface * - Domain specific interface - Dynamic user input - Graphic input - Other media input	

(Method : * , Tool : o)

A Distributed Approach to Real-Time Payments-Processing in a Partially-Connected Network of Banks (1994)

(Tony Lee and Sumit Ghosh)

Feature	Degree
o Real-time simulation capability * (Use of physical real-time clock)	
o Distributed and parallel processing * (No distributed execution, Distributed over LAN only, Distributed among node in a HPM, Distributed over LAN and among nodes in a HPM)	Distributed over B-ISDN
o Hardware requirements - General : machine type and capacity - Special : machine type and capacity	SUN 4/60 (16MB MM, 12.5 MIPS CPU)
o Use of real measured data or randomly synthesized * data only	Generate stochastically
o Number of states maintained * > 1M (high fidelity) > 100K ∧ < 1M (medium fidelity) < 100K (low fidelity))	
o Base language for specifying simulation models *	C
o System software requirements - Language tools - OS	UNIX
o Object orientation or not * (Simulation model)	No
o Developer	Brown Univ.

o Development history	
o Ease of stepwise refinement & modular expansion *	
o Simulation output * - Graphic . Animation . 3D - Statistics - Other media output	Conventional graph
o DB interface	No
o Reusable library - Simulation models - Simulator modules	No
o Special user interface * - Domain specific interface - Dynamic user input - Graphic input - Other media input	No

(Method : * , Tool : o)

Object-Oriented Simulation and Sim++ (1991)

(Greg Lomow and Dirk Baezner)

Feature	Degree
o Real-time simulation capability * (Use of physical real-time clock)	
o Distributed and parallel processing * (No distributed execution, Distributed over LAN only, Distributed among node in a HPM, Distributed over LAN and among nodes in a HPM)	Distributed among node in a HPM
o Hardware requirements - General : machine type and capacity - Special : machine type and capacity	Not restricted
o Use of real measured data or randomly synthesized * data only	Both
o Number of states maintained * > 1M (high fidelity) > 100K ∧ < 1M (medium fidelity) < 100K (low fidelity))	
o Base language for specifying simulation models *	Sim++
o System software requirements - Language tools - OS	Not restricted
o Object orientation or not * (Simulation model)	Object Oriented
o Developer	Jade Simulations International Corporation

o Development history	
o Ease of stepwise refinement & modular expansion *	Appropriate
o Simulation output * - Graphic . Animation . 3D - Statistics - Other media output	
o DB interface	No
o Reusable library - Simulation models - Simulator modules	Yes
o Special user interface * - Domain specific interface - Dynamic user input - Graphic input - Other media input	

(Method : * , Tool : o)

C++T Definition in BNF

(1) RTO.k program definition

RTO.k_program * *Rto_class_definition_list*

main ( )

{

static_Rto_instantiation_list

static_instance_activation_list

}

SpM_body_definition

SvM_body_definition

exception_handler_definition

MVD_expiration_handler_definition

Rto_class_definition_list * *Rto_class_definition*

| *Rto_class_definition* *Rto_class_definition_list*

Rto_class_definition * **RTO_class** *Rto_class_name*

{ *Rto_class_body* } ;

Rto_class_name * *identifier*

Rto_class_body * *ODS_section*

SpM_section

SvM_section

static_Rto_instantiation_list * *static_Rto_instantiation*

| *static_Rto_instantiation* *static_Rto_instantiation_list*

static_Rto_instantiation * *Rto_class_name* *Rto_instance_list*

Rto_instance_list * *Rto_instance_name* ;

| *Rto_instance_name* , *Rto_instance_list*

static_instance_activation_list * *static_instance_activation*

| *static_instance_activation*

static_instance_activation_list

static_instance_activation * *Rto_instance_name* . **activate** ;

(2) ODS definition

ODS_section * **ODS** : *ODS_segment_definition_list*

ODS_segment_definition_list * *ODS_segment_definition*

| *ODS_segment_definition*

ODS_segment_definition_list

ODS_segment_definition * *ODS_segment_name* { *ODS_segment_body* } ;

ODS_segment_name * *identifier*

ODS_segment_body * *ODSS_data_declaration_list*

ODSS_access_function_declaration_list

ODSS_data_declaration_list * *var_declaration_list* *MVD_specifier* ;

| *var_declaration_list* *MVD_specifier* ;

ODSS_data_declaration_list

var_declaration_list * *type_specifier identifier_list*

identifier_list * *identifier* | *identifier* , *identifier_list*

MVD_specifier * ℳ | *MVD_declaration* *MVD_expiration_handler_declaration*

MVD_declaration * **during** *effective_period*

effective_period * *hmss_expression*

hmss_expression * *hour_exp* : *min_exp* : *sec_exp* *msec_exp*

hour_expression * *digits*

min_expression * *digits*

sec_expression * *digits*

msec_expression * ℳ | . *digits*

ODS_access_function_declaration_list * ℳ

| *ODS_access_function_declaraiton*

ODS_access_function_declaration_list

ODS_access_function_declaration * *CC_specifier type_specifier*

function_name (*arg_specifier*) ;

CC_specifier * ℳ | **shared** | **exclusive**

function_name * *identifier*

arg_specifier * ℳ | *type_specifier_list*

type_specifier _list * *type_specifier* | *type_specifier* , *type_specifier_list*

MVD_expiration_handler_declaration * ℳ | **on_expiration** *function_name* ( )

(3) SpM definition

SpM_definition * **SpM :** *SpM_list*

SpM_list * ℳ | *SpM_definition* *SpM_list*

SpM_definition * *SpM_name* {

using_services_specifier

using_SpM_specifier

using_ODSS_specifier

scheduling_mode_specifier

AAC_list

exception_handler_declaration

} ;

SpM_name * *identifier*

using_services_specifier * ℳ | **using_services** *server_Rto_SvM_list* ;

server_Rto_SvM_list * *Rto_class_name* :: *SvM_name*

| *Rto_class_name* :: *SvM_name* , *server_Rto_SvM_list*

using_SpM_specifier * ℳ | **using_SpM** *SpM_name_list* ;

SpM_name_list * *SpM_name* | *SpM_name* , *SpM_name_list*

using_ODSS_specifier * ℳ | **using_ODSS** *ODS_segment_access_list* ;

ODS_segment_access_list * *ODS_segment_name* : *access_mode*

| *ODS_segment_name* : *access_mode* ,

ODS_segment_access_list

access_mode * **R | RW | UNSPEC**

scheduling_mode_scpefier * ℳ | **always** | **on_demand**

AAC_list * ℳ | *AAC_definition* *AAC_list*

AAC_definition * *AAC_name* {

activation_time_specifier

deactivation_time_specifier

period_specifier

relative_start_time_specifier

worst_case_execution_time_specifier

};

AAC_name * *identifier*

activation_time_specifier * ℳ | **from** *activation_time* ;

deactivation_time_specifier * ℳ | **to** *deactivation_time* ;

period_specifier * ℳ | **every** *period* ;

relative_start_time_specifier * ℳ

| **start_during** *realtive_earliest_start_time* , *relative_latest_start_time* ;

*worst_case_execution_time_specifier** ℳ

| **finish_within** *worst_case_execution_time* ;

activation_time * *hmss_expression* *relative_time_indicator*

deactivation_time * *hmss_expression* *relative_time_indicator*

relative_time_indicator * ℳ | **relative**

period * *hmss_expression*

relative_earliest_start_time * *hmss_expression*

relative_latest_start_time * *hmss_expression*

worst_case_execution_time * *hmss_expression*

exception_handler_declararion * ℳ | **on_failure** *exception_handler_name* **( ) ;**

exception_handler_name * *identifier*

SpM_body_definition_list * ℳ

| *SpM_body_definition* *SpM_body_definition_list*

SpM_body_definition * *Rto_class_name* :: *SpM_name* **( )**

{ *function_body* }

dynamic_schedule_SpM_statement * **schedule** SpM_name **with** *AAC_name* **;**

(4) SvM definition

SvM_section * **SvM :** *SvM_list*

SvM_list * ℳ | *SvM_definition* *SvM_list*

SvM_definition * *SvM_name* (*SvM_argument_type_spec*)

{

using_services_specifier

using_SpM_specifier

using_ODSS_specifier

maximum_service_request_acceptance_rate

worst_case_execution_time_specifier

worst_case_message_pickup_delay_specifier

exception_handler_declaration

} ;

SvM_argument_type_spec * ℳ

| *input_msg_spec*

| *input_msg_spec* , *output_msg_spec*

maximum_service_request_accpetance_rate * ℳ

| **accept** *digits* **per** *hmss_expression* ;

worst_case_message_pickup_delay_specifier * ℳ

| **start _within_on_message_arrival** *hmss_expression*

SvM_body_definition_list * ℳ

| *SvM_body_definition* *SvM_body_definition_list*

SvM_body_definition * *Rto_class_name* :: *SvM_name* (*SvM_argument_spec*)

{ *function_body* }

SvM_argument_spec * ℳ

| *incoming_message_spec*

| *incoming_message_spec* , *reply_message_spec*

(5) Inter-RTO communication

service_request_statement * *Rto_instance_name* . *SvM_name*

(*request_message_spec*) ;

| *Rto_instance_addr* -> *SvM_name*

(*request_message_spec*) ;

request_message_spec * *sync_type_specifier* ,

service_deadline_specifier ,

send_msg_addr_specifier ,

recv_msg_addr_specifier

sync_type_specifier * **SYNC** | **ASYNC** | **ASYNC-REPLY**

service_deadline_specifier * ℳ | *hmss_expression*

send_msg_addr_specifier * ℳ | *addr_specifier*

async_reply_wait_statement * **wait_reply** (*recv_msg_addr_specifier*) ;

return_reply_statement * **return_reply** (*reply_msg_addr_specifier* , *client_id*) ;

(6) Dynamic RTO creation and destruction

dynamic_Rto_creation_statement * *Rto_dynamic_alloc_statement*

dynamic_instance_activation

Rto_dynamic_alloc_statement * *Rto_pointer_var* = **new** *Rto_class_name*

(*actual_arg_list*) ;

dynamic_instance_activation * *Rto_pointer_var* -> **activate** ;

Rto_pointer_var * *identifier*

Rto_destruction_statement * **RTO_exit** ;

Rto_destruction_and_reply_statement * **RTO_exit_reply** (*reply_message_spec*) ;

제 1 차년도
연차 보고서

RTS 기술개발

RTS Technology Development

Development of a Real-Time Object Model and Supporting Operating System Facilities for Real-Time Simulation

연구기관

University of California, Irvine

과 학 기 술 처

여 백

Final Report

Development of A Real-Time Object Model and Supporting Operating System Facilities for Real-Time Simulation - Year I

August 27, 1995

Contractor:

UCI-KRG
14 Urey Court
Irvine, California 92715, U.S.A.

Principal Investigator:

K. H. (Kane) Kim, Ph.D

(714) 824-5542 (day), 856-2664 (evening), 824-4076 (FAX)
Internet: Kane@Ece.Uci.Edu

Performance Period: November 1994 - July 1995

Sponsor:

POSTECH
San 31, Hyoja Dong
Pohang, Kyungbuk, 790-784
Republic of Korea

여 백

This report summarizes the main results of our research carried out during the period of November 1994 - July 1995 under the Research Contract entered with POSTECH in November, 1994.

A summary of the main results obtained is given and further details on some aspects are included in appendices. .

1. Research Objective

The objective of the research efforts made during the reporting period (Year I) was to establish a proper extension of the basic object model which is capable of uniformly and accurately representing both real-time embedded computer systems and application environments Not only description but also simulation of application environments is often performed as integral steps of validating control computer system designs. A desirable accurate mode of simulating application environments is the real-time simulation in which *the simulation objects show the same timing behavior that the simulation targets do.* A desirable model is thus one that supports realization of the uniformity and the high degree of accuracy in representation of application environments, environment simulators, and control computer system designs at different levels evolving during the system development cycle.

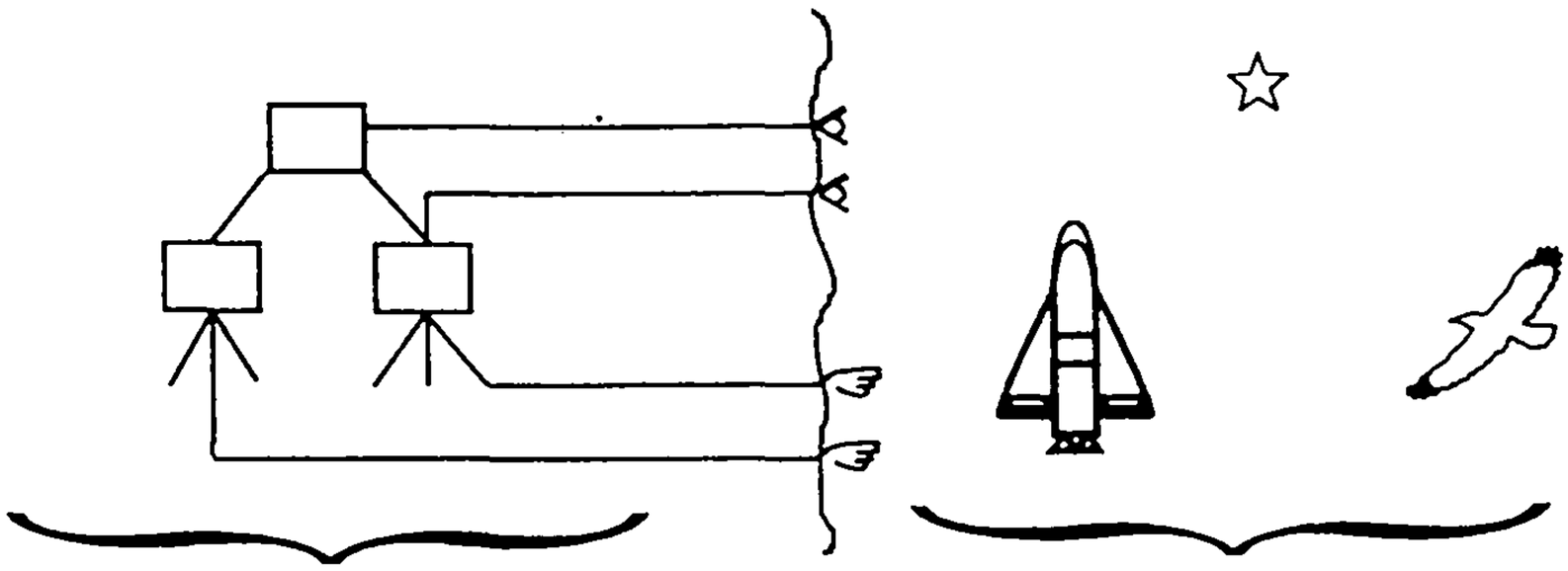
We have established such a desirable real-time object model called the RTO.k object model, also called the *time-triggered real-time object* (TT-RTO) model. An initial abstract framework of the model was formulated several years ago jointly by the first co-author and Hermann Kopetz at Technical University of Vienna. This initial framework has evolved into a concrete syntactic structure associated with unambiguous execution semantics in recent years [Kim93, Kim94a, Kim94b]. The RTO.k object model was partially validated through two specification, design, and real-time simulation experiments conducted recently:
(1) An experiment that involved an application of the RTO.k structuring scheme to both the development of a defense system and that of an environment simulator and
(2) An experiment that involved an application of the RTO.k structuring scheme to both the development of a freeway traffic control system and that of an environment simulator.
These experiments reinforced our belief that the RTO.k model had the necessary representational power and also offered an efficient and rigorous way to develop complex real-time systems.

2. Definition of Real-Time Simulation

We define the real-time simulation as a mode of simulation in which *the simulation objects show the same timing behavior that the simulation targets do.* Figure 1 shows some useful roles of real-time simulation during development of complex real-time control computer system. As shown, real-time simulation involves the use of a real-time clock. After a control computer system has been implemented, it is often highly useful to connect it to a simulator of the application environment for validation of the control computer system rather than directly proceeding to interface the computer system with the application environment. A highly desirable simulator here is one capable of accurately immitating the timing behavior of the environment, i.e., real-time simulation of the environment. Also, before the control computer system is fully implemented, it is often useful to do real-time simulation of the control computer system to be implemented.

Figure 1.

Roles of Real-Time Simulation



Control Computer System — Application Environment

Real - - - - - - - - - Real

Real - - - - - - - - - Simulation

- high fidelity
  : Highly useful,
  Important to handle real-time & parallel proc - challenge ★
- low fidelity
  : Little challenge, somewhat useful

} Real-time clock must be used !

(Simulation - - - - - Real) : Not meaningful

Simulation -Medium fidelity (Intermediate executable design) - - Simulation

- high fidelity
  : useful (simulation is a challenge)
- low fidelity
  : somewhat useful

Simulation -Low fidelity (Early executable design) - - -

- high fidelity : useful
- low fidelity : useful

} Virtual-clock may be used !

★ Every action for updating an ODS & every reading of an object state by an external client must be timely !!
Also, inter-object info transfer must be timely !!

UCI KRG
3.16.95
(Rev.3.22.95)

3. A Scheme for Classification of Simulation Methods and Tools

To facilitate a systematic survey and analysis of the available simulation methods and tools for the purpose of evaluating the potentials of existing approaches and tools for real-time simulation as well as identifying desirable real-time simulation methods and approaches to be developed, we have established the following classification scheme.

- RT simulation or not ◆
  - ⇒ Gap between the timing behavior of the simulation and the timing behavior of the simulation target
  - ⇒ Use of physical real-time clock
- Distributed and parallel processing ◆
  - No distributed execution,
  - Distributed over LAN only,
  - Distributed among node in a HPM,
  - Distributed over LAN and among nodes in a HPM
- Hardware requirements
  - General : machine type and capacity
  - Special : machine type and capacity
- Use of real measured data or randomly synthesized data only ◆
- Number of states maintained ◆
  - > 1 M (high fidelity)
  - > 100 K ∧ < 1 M (medium fidelity)
  - < 100 K (low fidelity)
- Base language for specifying simulation models ◆
- System software requirements
  - Language tools
  - OS
- Object orientation or not ◆
  - Simulation model
- Developer
- Development history
- Ease of stepwise refinement & modular expansion ◆

- Simulation output ♦
  - Graphic
    - . Animation
    - . 3D
  - Statistics
  - Other media output
- DB interface
- Reusable library
  - Simulation models
  - Simulator modules
- Special user interface ♦
  - Domain specific interface
  - Dynamic user input
  - Graphic input
  - Other media input

A brief survey conducted indicated that little systematic approaches for real-time simulation have been formulated although various ad hoc approaches have been used in different applications. It appeared that only one research group, headed by Bernard P. Zeigler, had adopted in a paper published in 1993 the definition of real-time simulation which is similar to ours given in Section 2. Also, we could not find any signs of serious past attempts to establish proper real-time object models capable of supporting real-time simulation.

4. The RTO.k Object Model

Only a brief overview is given in this section. More details can be found in appendices.

The basic structure of an RTO.k object is depicted in Figure 2. It is an extension of the conventional object model(s) and two most important and unique extensions are the following:

(a) *Two clearly separated groups of methods:*

For some methods of an RTO.k object, a real-time clock serves as the mechanism for triggering the method executions as the clock reaches some values specified at design time and such methods are called time-triggered (TT-) methods, also called the spontaneous methods (SpM's), and *clearly* separated from the conventional service methods (SvM's) triggered by

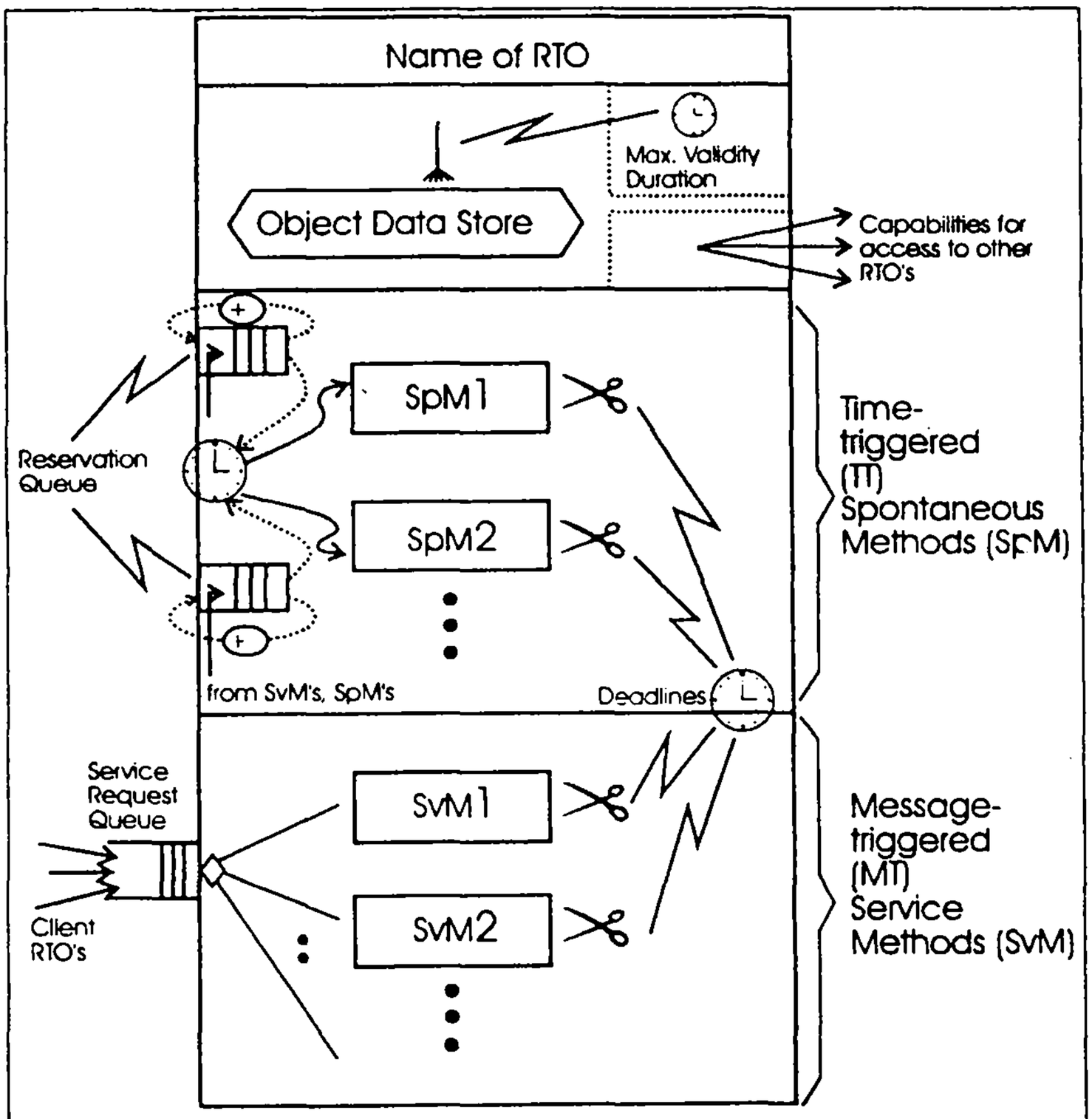


Figure 2. Structure of the RTO.k object model
(Adapted from [Kim94c])

messages from clients. The two types of methods in an RTO.k object are different not only in the way their executions are triggered but also in that

> "actions to be taken at real times *which can be determined at the design time* can appear only in SpM's"

Therefore, actions of the type "at constant-clock-value do S" or the type "sleep-until constant-clock-value" can appear only in SpM's. Incorporation of SpM's means introducing the potential for the following two new types of concurrent executions of object methods in addition to the potential for concurrent executions of SvM's that exist in conventional objects:

> (Type I) Concurrency among SpM executions: This concurrency is specified in an implicit but natural manner, e.g., two SpM's designed to be triggered at 10 am.
>
> (Type II) Concurrency between SpM executions and SvM executions.

(b) *Basic concurrency constraint (BCC):*

In order to dramatically reduce the designer's efforts in guaranteeing timely service capabilities of RTO.k objects, the execution rule which prevents conflicts between SpM's and SvM's is incorporated. Basically, *activation of an SvM triggered by a message from an external client is allowed only when potentially conflicting SpM executions are not in place.* To be exact, when a message-triggered SvM is not free of conflict with an SpM in accessing the same portion of the object data space (ODS), execution of the former method (SvM) must not be allowed in a time zone earmarked for a TT-execution of the latter method (SpM). This restriction is called the basic concurrency constraint (BCC). Therefore, SpM's are given higher priorities for execution over the SvM's. Note that this BCC does not impose any restriction on concurrent execution of SpM's or concurrent execution of SvM's. Therefore, executions of SpM's are not disturbed by SvM executions and triggering times of SpM's are fixed at the design time. If a statement of the type "at 10am do S" appears in an SpM, its timely execution can be easily assured.

The above two features make the RTO.k object model clearly distinguished from other proposed real-time object models [Att91, Ish92, Tak92]. In addition, the RTO.k object contains the following features not found in the conventional object model(s):

(c) For each execution of a method of an RTO.k object, a deadline is imposed;

(d) Real-time data contained in an RTO.k object become invalid after the interval called the maximum validity duration passes.

Triggering times for SpM's must be fully specified as constants during the design time. Those real-time constants appear in the first clause of an SpM specification called the autonomous activation condition (AAC) section. An example of an AAC is

"for t = from 10am to 10:50am every 30min

start-during (t, t+5min) finish-by t+10min"

which has the same effect as

{"start-during (10am, 10:05am) finish-by start_time+10min",

"start-during (10:30am, 10:35am) finish-by start_time+10min"}.

A provision is also made for making the AAC section of an SpM contain only candidate triggering times, not actual triggering times, so that a subset of the candidate triggering times indicated in the AAC section may be dynamically chosen for actual triggering. Such a dynamic selection occurs when an SvM within the same RTO.k object requests future executions of a specific SpM. The AAC specifying candidate triggering times rather than actual triggering times starts with a declaration "if-demanded". Therefore, there are two different modes of determining triggering times for SpM's:

(a) fully determined during the system design, in which case the SpM is said to be *statically scheduled*, and

(b) determined during the run time when an SvM requests executions of the SpM and designates a subset of the candidate triggering times prepared during the design time as actual triggering times, in which case the SpM is said to be *partially dynamically scheduled.*

An underlying design philosophy of the RTO.k object model is that an RTCS will always take the form of a network of RTO.k objects. RTO.k objects interact via calls by client objects for service methods in server objects. The caller may be an SpM or an SvM in the client object. In order to facilitate highly concurrent operations of client and server objects, non-blocking (sometimes called asynchronous) types of calls (i.e., service requests) can be made to SvM's.

The designer of each RTO.k object provides a guarantee of timely service capabilities of the object by indicating the *deadline for every output* produced by each SvM (and each SpM which may be executed on requests from SvM's) in the specification of the SvM (and some relevant SpM's) advertised to the designers of potential client objects. Before determining the deadline specification, the server object designer must convince himself/herself that with the object execution engine (hardware plus operating system) available, the server object can be implemented to always execute the SvM such that the output action is performed within the

deadline. Again, the BCC contributes to major reduction of these burdens imposed on the designer.

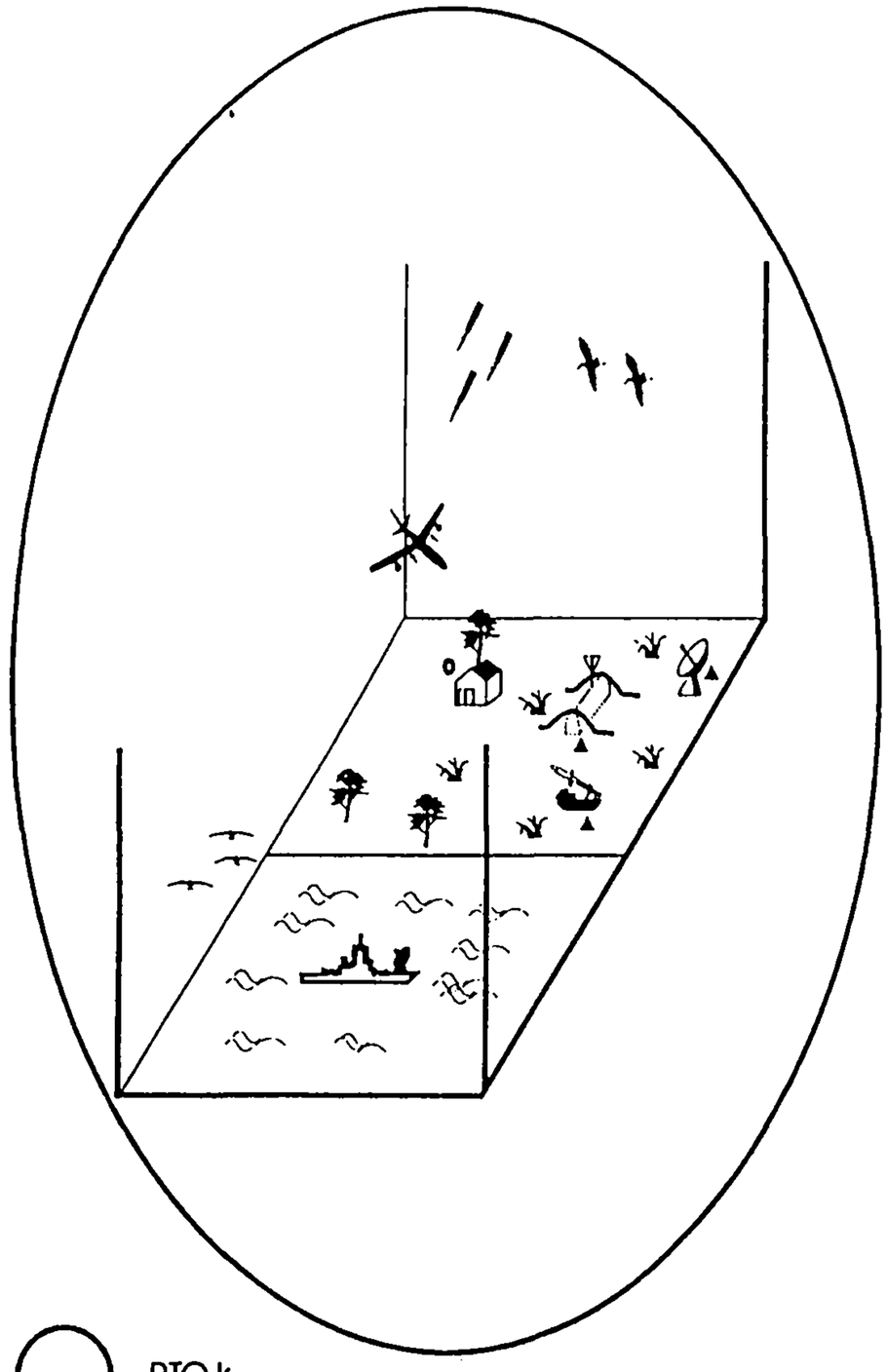
As will be discussed in the next section, the RTO.k object model is effective not only in the multiple-level abstraction of real-time (computer) control systems under design but also in the accurate representation and simulation of the application environments. It enables uniform structuring of control computer systems and application environment simulators [Kim94b] and this presents considerable potential benefits to the system engineers.

5. Real-Time Simulation with the RTO.k Object Model

The object model was initially formulated and used in simulation applications [Dah72]. Therefore, use of the object model in modeling the environment objects, i.e., modular entities in the environment which have time-varying internal states, has been practiced for a long time. However, as mentioned in the preceding section, the recently formulated version of the object model, the RTO.k object model, supports more accurate detailed modeling of environment objects. This aspect and how the RTO.k object model can be used during the requirements specification and high-level design steps are discussed in this section with examples derived from the defense C^3 area.

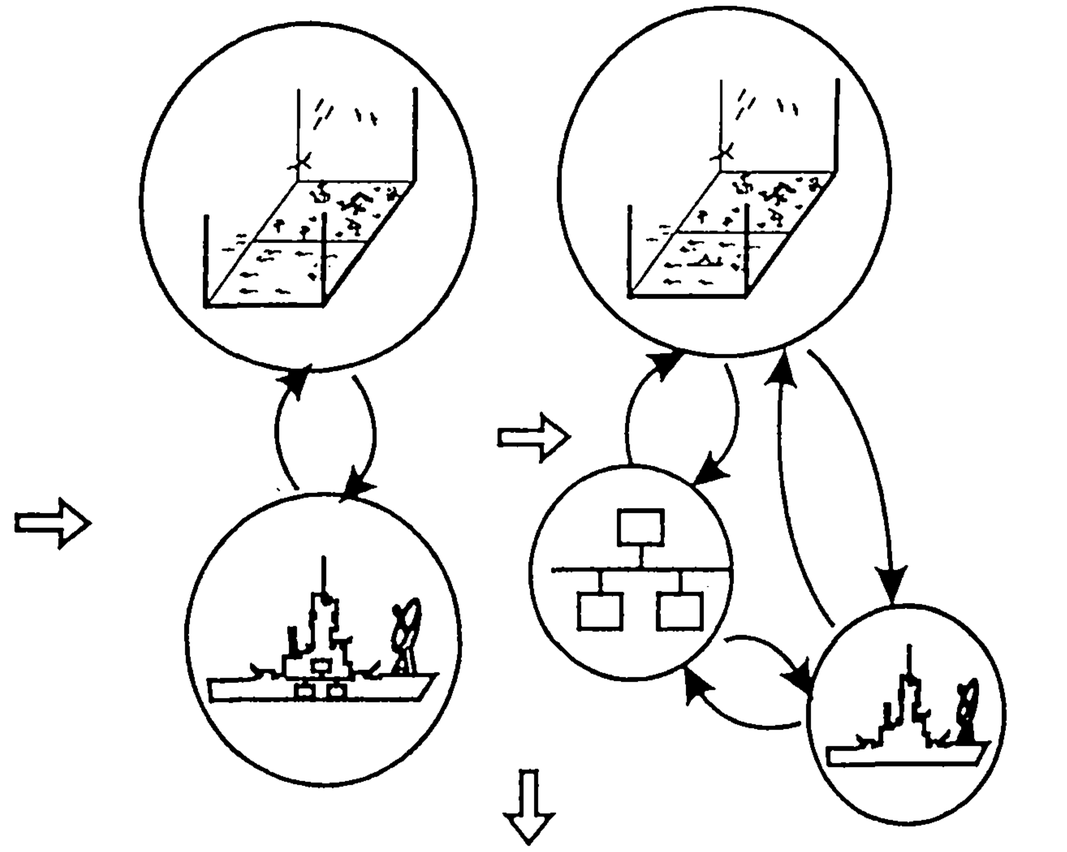
Consider an anti-missile defense scenario depicted in Figure 3. The environment in this context means a sky+land+sea segment of interest, called the "theater", and any moving objects in that theater including a valuable target to be defended (e.g., a ship) and flying objects (e.g., hostile reentry vehicles (RV's) and non-threatening slow-moving objects).

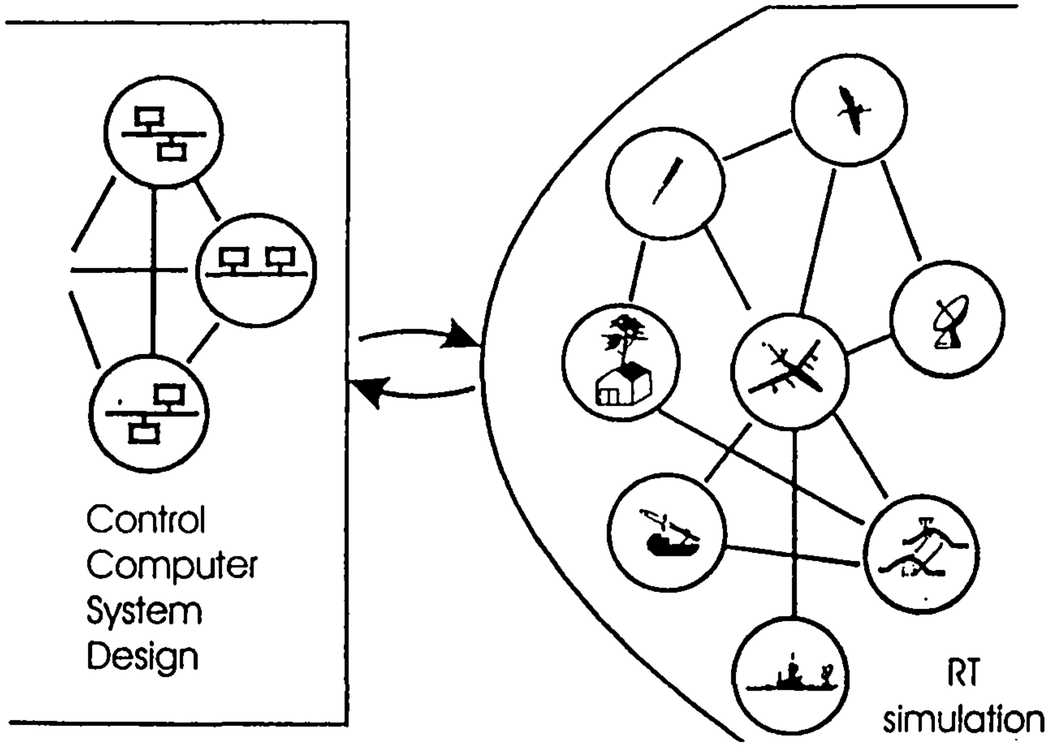
Un. form Structuring



RTO.k

Figure 3. An anti-missile





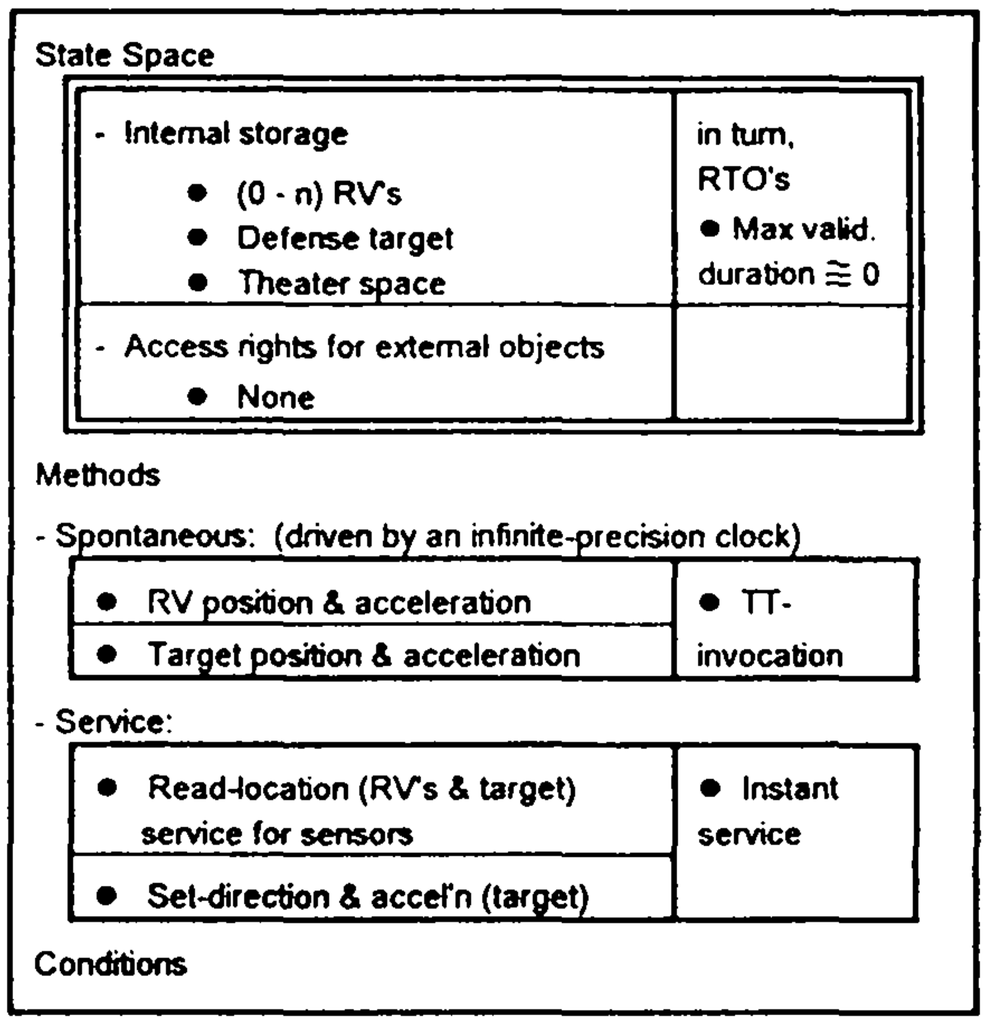


Figure 4. An RTO.k specification of a reduced theater

Initially the top-level requirements given by the customer who places an order for the defense system which is called the surface defense network (SDN) here are as follows.

(1) Each RV should be intercepted if it is dangerous.

(2) If there are more dangerous RV's than the interceptors, then the early arriving RV's should be intercepted and the defense target should be moved toward a safer location.

The system designer will first decide on the set of sensors and the set of actuators (e.g., interceptors) to be deployed. Thereafter, the functions of the computer based control system will be determined based on the control theory logic adopted.

Assume for the time being that sensors and actuators have not been chosen and there is nothing in the land area. Assume further that there is not any non-threatening flying object in the sky portion of the theater. An RTO.k representation of this reduced theater is depicted in Figure 4.

- The internal storage (object data store) of this high-level RTO.k object basically consists of the space in the theater, a defense target (ship), and a dynamically varying number of RV's. Therefore, the information kept in this RTO.k object is a composition of the information kept in the defense target, the RV's, and any other object in the theater space. A noteworthy property here

is that each of these components that are treated as components of the object data store, i.e., the defense target and the RV's themselves, can in turn be represented as an RTO.k object.

Conceptually the TT-methods in this top-level RTO.k object model for the theater are *activated continuously* and each of their executions is *completed instantly*. If we adopt the less precise version of the model in which the time domain is a discrete domain and the time gap between two instants adjacent in the domain is called a clock tick, then a less accurate representation of the environment results. That is, such view dictates the activation frequency of any spontaneous method to be no more than once per every clock tick while allowing each execution to be completed before or by the time of the following activation of the same method. Therefore, the spontaneous methods are the mechanisms for representing (or simulating) continuous state changes that occur naturally in the environment objects. The natural parallelism that exists among the environment objects is precisely represented by use of multiple TT-methods which may be activated simultaneously. In general, the accuracy of an RTO.k object representation of the environment is a direct function of the activation frequencies of TT-methods.

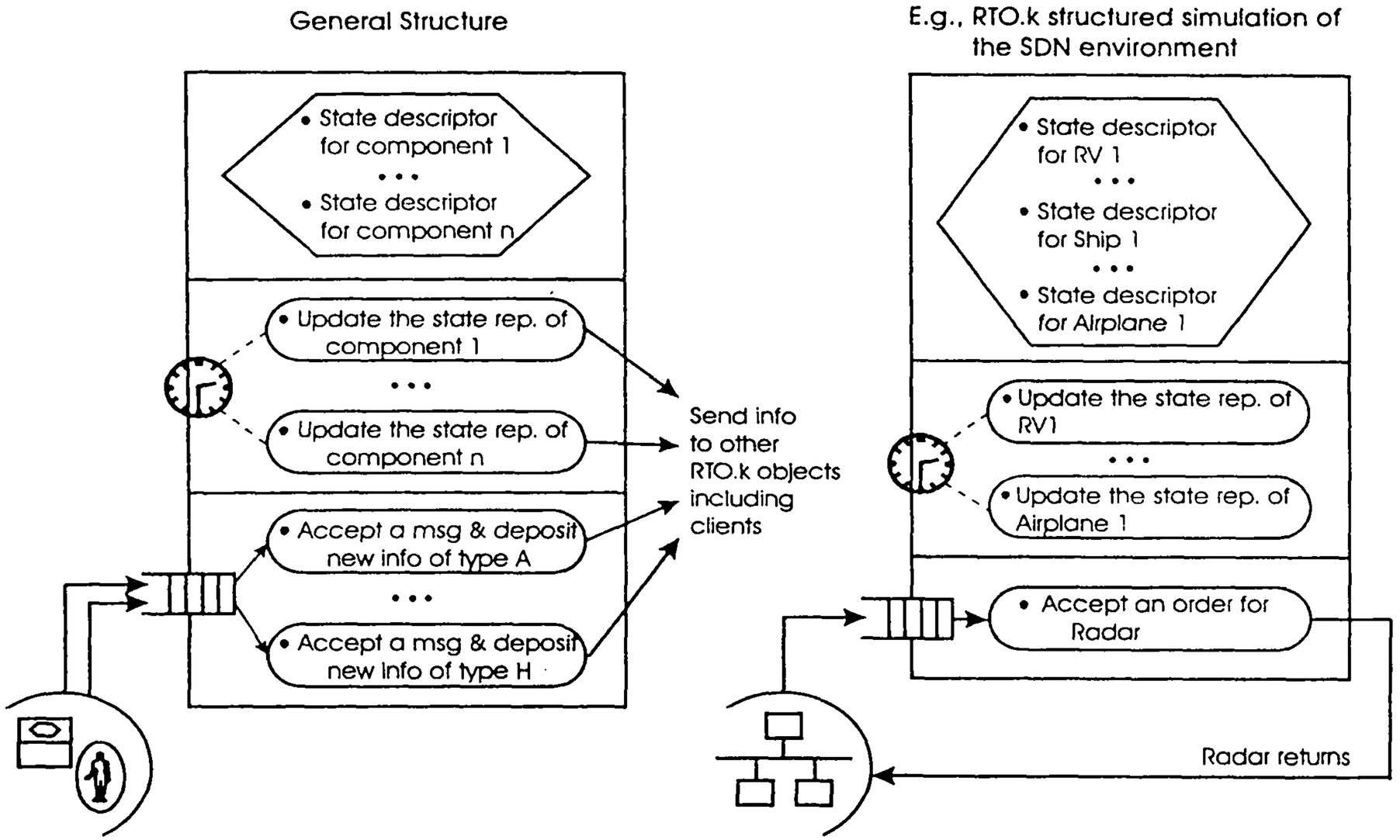
The service methods in Figure 4 are defined with the assumption that sensors and actuators will be located outside the theater. In this respect, the theater assumed in Figure 4 is different from the theater shown in Figure 3. The clients which will invoke the service methods in Figure 4 are unusual ones, i.e., sensors and actuators interfacing with environment objects. Sensors work to obtain information about the states of environment objects, primarily locations, movement directions, and some signatures of RV's and those of the defense target. This relationship between sensors and environment objects can be represented partially by the service methods such as "Read-location (environment object)" in Figure 4, with the understanding that such a method is executed at the instant at which a sensor makes an observation of the environment state.

Similarly, actuators work to make impact on the conditions and future courses of environment objects. In this example, the only possible impact that can be made on RV's by the system being designed is the collision of the interceptors against the hostile RV's. These interceptors are actuators produced at an early stage of the system design and once they are produced, they should be treated as environment objects in the theater as well. Although Figure 4 represents the theater before introduction of such actuators, there are some control points in the defense target which can be accessed from the computer system, typically structured as a control computer network (CCN), via a communication channel, e.g., a radio communication device. The "Set-direction & acceleration (target)" service method in Figure 4 represents such possibility.

Figure 5 depicts the RTO.k based real-time simulation approach in a generic form.

Figure 5

Real-Time Simulation via the RTO.k Object Modeling



The type or mode of simulation in which the simulation objects show the same timing behavior that the simulation targets do

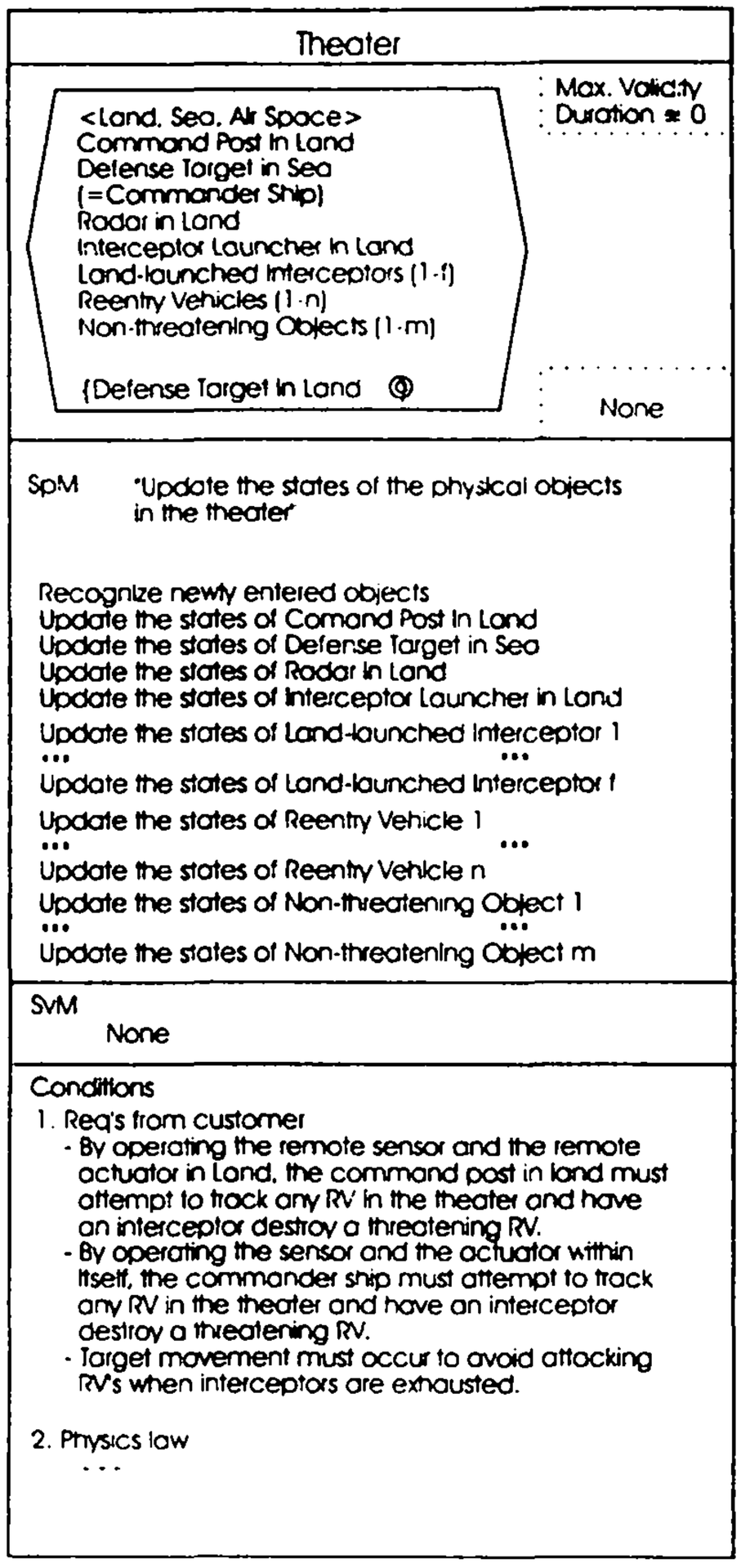


Figure 6. An RTO.k specification of the theater

The theater depicted in Figure 3 already includes some sensors, i.e., a radar in the land and a radar on the commander ship in sea, and some actuators, i.e., interceptors in the land and interceptors on the commandership. A CCN is also housed in the commander ship which is all inside the theater. A faithful RTO.k representation of this theater in Figure 3 is shown in Figure 6.

As mentioned earlier, a single RTO.k object specification of the theater can be refined into a network of RTO.k object specifications, each corresponding to a different environment object or a computing object. Figure 7 depicts such a refinement process. All the knowledge contained in the *Conditions* section of the single RTO.k object specification should be retained, most likely in a scattered form, in the network of lower-level object specifications. Additional knowledge may also be introduced during the refinement process. In our experiment, all RTO.k object specifications of environment objects were converted into RTO.k object structured simulators of the environment objects. Efficient and systematic production of simulators as well as flexible control in the degree of simulation accuracy are significant benefits of the aforementioned modeling and simulation approach which have been witnessed in our experiment.

A detailed design of the CCN located inside the defense target in sea in Figure 7 has been carried out at the UCI DREAM Laboratory. The entire SDN prototype was implemented on a local area network of 7 personal computers. The result was an easily modifiable implementation of a non-trivial C3 system with highly predictable temporal behavior. The development effort was considerably less than what would have been required if we used the structuring approach which had been used in our previous development of several C^3 protypes [Kim89].

A similar experiment was conducted with the freeway traffic control as the application. The results of this experiment were equally encouraging.

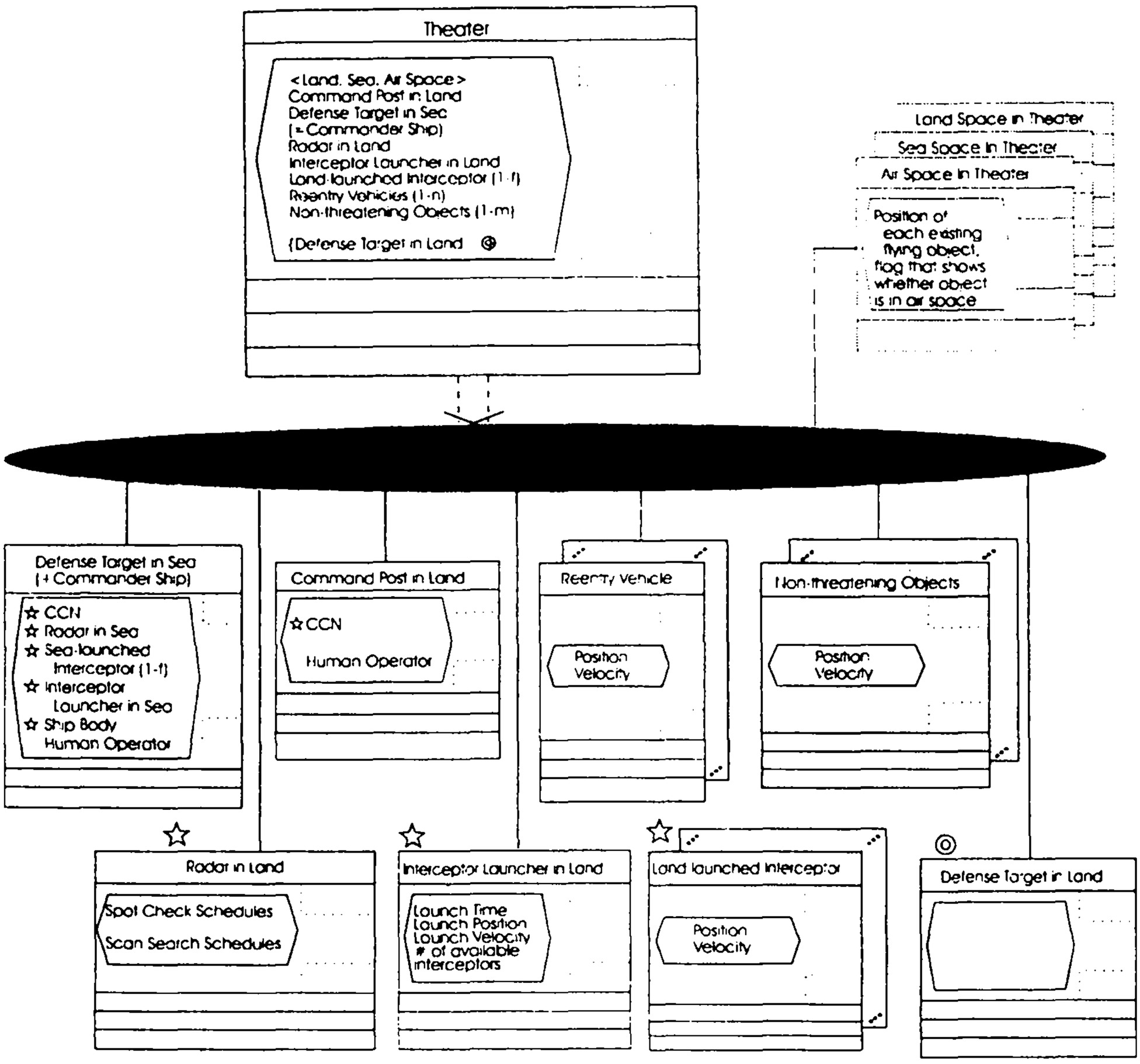


Figure 7. Decomposition of the theater RTO.k

6. An execution Engine for RTO.k Objects

An execution engine for RTO.k objects must possess guaranteed timely service capabilities. Otherwise, timely service capabilities of RTO.k objects cannot be guaranteed. Recently an execution engine model called the DREAM (Distributed Real-Time Ever Available Microcomputing) kernel was defined and then its first prototype, called the DREAM kernel v.D2, was implemented to run on a network of PC's connected by Ethernet. It actually supports conventional processes and shared data monitors as well and thus it contains full features of a general-purpose kernel. Therefore, the architectural principles adopted in the DREAM kernel to enable guaranteering timely services to its clients, i.e., processes and RTO.k objects, are broadly applicable to any situations where real-time operating systems are developed. The DREAM kernel rigidly manages execution times of concurrency units and interrupt handlers in order to provide guaranteed timely services to both RTO.k object structured applications and conventional process structured applications.

We feel that this DREAM kernel can be extended into a RTO.k simulation engine, capable of executing RTO.k models of simulation targets, with relative ease. Further details of this kernel are provided in [Kim95b, Kim95c] included in this report as Appendices B and C.

7. Conclusion

A promising object model for both design of control computer systems and real-time simulation of both computer systems and application environments, which is called the RTO.k object model, has been obtained. The model has been partially validated through experimental implementation of real-time simulators. Some progresses have been made in development of a simulation engine capable of executing RTO.k models of simulation targets. One of the major tasks to be carried out in the next year is to implement a language processor which converts RTO.k objects in a C++ extension into the code running on a simulation engine.

References

[Att91] Attoui, A. and Schneider, M., "An Object Oriented Model for Parallel and Reactive Systems", *Proc. IEEE CS 12th Real-Time Systems Symp.*, 1991, pp. 84-93.

[Bri77] Brinch Hansen, P., "The Architecture of Concurrent Programs," Prentice-Hall, 1977.

[Dah72] Dahl, O.J., "Hierarchical Program Structuring", in Dahl, Dijkstra, & Hoare eds., 'Structured Programming', Aca. Press, NY, 1972.

[Ish92] Ishikawa, Y., Tokuda, H., and Mercer, C. W., "An Object-Oriented Real-Time Programming Language", *IEEE Computer*, October 1992, pp. 66-73.

[Kim89] Kim, K.H., "An Approach to Experimental Evaluation of Fault-Tolerant Distributed Computing Schemes", IEEE Transactions on Software Engineering, June 1989, pp.715-725.

[Kim93] Kim, K.H., "Achieving Ultra-High Reliability of Distributed and Parallel Computer Systems in Safety-Critical Applications ", *Proc. InfoScience '93* (Int'l Conf. organized by Korea Information Science Society in commemoration of its 20th anniversary), Seoul, Oct. 22, 1993, pp.190-199 (Invited paper).

[Kim94a] Kim, K.H. et al., "Distinguishing Features and Potential Roles of the RTO.k Object Model", *Proc. 1994 IEEE CS Workshop on Object-oriented Real-time Dependable Systems (WORDS)*, Oct. '94, Dana Point, pp.36-45.

[Kim94b] Kim, K.H. and Kopetz, H., "A Real-Time Object Model RTO.k and an Experimental Investigation of Its Potentials", *Proc. 1994 IEEE CS Computer Software and Applications Conf. (COMPSAC)*, Nov. 1994, Taipei, pp.392-402.

[Kim94c] Kim, K.H., "A Utopian View of Future Object-Oriented Real-Time Dependable Computer Systems", (Invited paper) *Proc. 1st Int'l Workshop on Real-Time Computing Systems and Applications*, Seoul, Korea, Dec. 1994, pp. 59-69.

[Kim95a] Kim, K.H., Mori, K., and Nakanishi, H., "Realization of Autonomous Decentralized Computing with the RTO.k Object Structuring Scheme and the HU-DF Inter-Process-Group Communication Scheme", *Proc. 1995 IEEE CS Int'l Symp. on Autonomous Decentralized Systems (ISADS)*, April 1995, Phoenix, pp.305-312.

[Kim95b] Kim, K.H., "Toward New-Generation Real-Time Object-Oriented Computing", *Proc. IEEE CS 5th Workshop on Future Trends of Distributed Computing Systems (FTDCS)*, Cheju Island, Aug. '95, pp.520-529.

[Kim95c] Kim, K.H. et al., "A Timeliness-Guaranteed Kernel Model -- DREAM Kernel -- and Implementation Techniques ", To appear in Proc. RTCSA '95 (1995 Int'l Workshop on Real-Time Computing Systems & Applications), Tokyo, Oct. '95.

[Kop89] Kopetz, H., Damm, A., Koza, C., Mulazzani, M., Wolfgang, S., Senft, C., and Zainlinger, R., "Distributed Fault-Tolerant Real-Time Systems: The Mars Approach", IEEE Micro, Feb. 1989, pp. 25-39.

[Kop90] Kopetz, H. and Kim, K.H., "Temporal Uncertainties in Interactions among Real-Time Objects", *Proc. IEEE Computer Society's 9th Symp. on Reliable Distributed Systems*, Huntsville, AL, Oct. 1990, pp.165-174.

[Mor93] Mori, K., "Autonomous Decentralized Systems: Concept, Data Field Architecture, and Future Trends", *Proc. IEEE CS Int'l Symp. on Autonomous Decentralized Systems (ISADS 93)*, Mar. 1993, Kawasaki, Japan, pp. 28-34.

[Ras89] Richard Rashid, Robert Baron, Alessandro Forin, David Golub, Michael Jones, Daniel Julin, Douglas Orr, Richard Sanzi, "Mach: A Foundation for Open Systems", *Proc. the IEEE Second Workshop on Workstation Operating Systems (WWOS2)*, September 1989, pp.109-113.

[Tak92] Takashio, K., and Tokoro, M., "DROL: An Object-Oriented Programming Language for Distributed Real-Time Systems", *Proc. OOPSLA*, 1992, pp. 276-294.

Appendix A.

[Kim94a] Kim, K.H. et al.,
"Distinguishing Features and Potential Roles of the RTO.k Object Model",
Proc. 1994 IEEE CS Workshop on Object-oriented Real-time Dependable Systems (WORDS), Oct. '94, Dana Point, pp.36-45.

Proc. WORDS '94 (IEEE Computer Society Workshop on Object-oriented Real-time Dependable Systems), Dana Point, Oct. '94, pp.36-45.

Distinguishing Features and Potential Roles of the RTO.k Object Model

K. H. (Kane) Kim, Luiz Bacellar, Yuseok Kim,
University of California, Irvine

Dong K. Choi,
University of Pennsylvania

Steve Howell, and Michael Jenkins
USN Naval Surface Warfare Center

Abstract: In recent years, search for proper extension of the basic object model to meet the needs present in the hard-real-time system development environments has become a serious research issue. The first co-author and Hermann Kopetz at Technical University of Vienna, formulated an extension of the basic object model as one attempt to meet such needs and it has been called the RTO.k object model. In the past two years, we have been making efforts to develop practical easy-to-use tools which assist the system engineers in
(1) RTO.k structured description and simulation of application environments and
(2) RTO.k structured hierarchical design of control computer systems.
In this paper, unique features of the RTO.k model which distinguish it from other extensions of the basic object model as well as common features will be presented first. The roles which the RTO.k model can play during various steps of the real-time system engineering process will then be discussed.

1. Introduction

Almost from the beginning days of object-oriented structuring [Dah72, Boo91, Rum91], numerous engineers and researchers concerned with real-time computing applications thought about introducing deadlines into the basic object model. However, system engineers have not been able to produce convincing demonstrations of the significant improvements in development of hard-real-time distributed computer systems (DCS's) with such a simply extended structuring approach. We feel that the most significant and desirable advance in the art of hard-real-time DCS development will have occurred when design-time guarantee of timely service capabilities of a system becomes an integral and easily practiced step in system engineering. Therefore, search for proper extensions of the basic object model to meet such and other needs present in the hard-real-time system development environments has become a serious research issue.

A seemingly distinct but closely related motivation to find a proper extension of the basic object model arises from the need to establish a coherently integrated methodology for engineering real-time DCS's [Kim93].

We perceive that the following conditions exist in the current practice of real-time system engineering:
(1) Lack of rigor in requirements specification, in particular, specification of temporal behavior requirements and dependability requirements;
(2) Weak traceability among various specifications and system models used during high-level design, implementation, validation, and evaluation; and
(3) Lack of integration in design techniques with poor consequences in the dependability and the guaranteed response delay of the systems produced.
In our view, one of the missing cornerstones for developing a coherently integrated engineering methodology is a system (and component) model which is effective not only for abstraction and stepwise refinement of real-time computer system designs but also for representing and providing a basis for analysis of the application environments. Not only descriptions but also simulations of application environments are often performed as integral steps of validating control computer system designs. A desirable model is thus one that supports realization of the *uniformity* and the high degree of *accuracy* in representation of application environments, environment simulators, and control computer system designs at different levels evolving during the system development cycle.

The RTO.k object model, also called the *time-triggered real-time object* (TT-RTO) model, is a result of the attempt by the first co-author and Hermann Kopetz at Technical University of Vienna to find a proper extension of the basic object model which is capable of uniformly and accurately representing both embedded computer systems and application environments. Based on the initial framework formulated by Kopetz and Kim in late 1980's [Kop90], a concrete syntactic structure associated with unambiguous execution semantics has been produced in recent years [Kim93, Kim94b]. The ease of facilitating design-time guarantee of timely service capabilities of objects has also been used as the fundamental guiding principle in devising this first concrete version of the RTO.k object model. Two most distinguishing characteristics of this model relative to other proposed extensions of the basic object model are (1) the clear-cut separation of the *time-triggered object methods*, also called the *spontaneous methods*, from the conventional,

message-triggered object methods, also called the *service methods*, and (2) the execution rule called the *basic concurrency constraint*.

Under the RTO.k object based uniform structuring approach, the combination of a control system design and an environment simulator takes the form of a network of RTO.k objects. A specification and implementation experiment that involved an application of the RTO.k structuring scheme to both the development of a defense system and that of an environment simulator was conducted recently [Kim94b]. This experiment reinforced our belief that the RTO.k model had the necessary representational power and also offered an efficient and rigorous way to develop complex real-time systems.

An overview of the major features of the RTO.k model is given in the next section and then the distinguishing features of the RTO.k model not found in other proposed extensions of the basic object model are discussed in Section 3 together with the costs and benefits of those features. Potential useful roles of the RTO.k object model in engineering of complex real-time DCS's are discussed in Section 4. The paper concludes in Section 5.

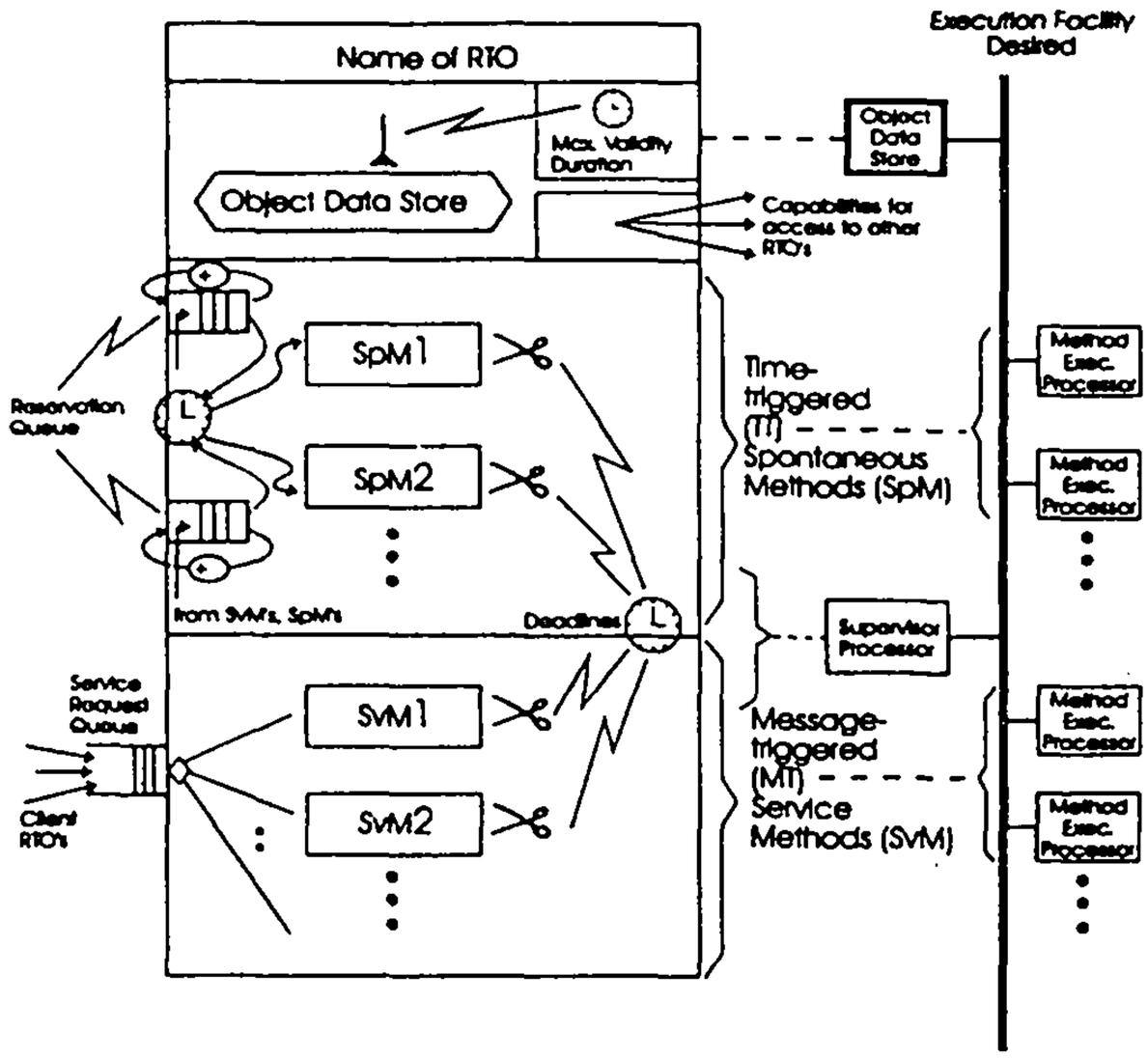


Figure 1. Structure of the RTO.k object model

2. Major features of the RTO.k model

The following two goals have been adopted as the fundamental guiding principles in formulation of the current RTO.k object model.

(a) Uniform structuring of both real-time DCS's and their application environment simulators;
(b) Facilitating design-time guarantee of timely service capabilities of objects.

2.1 Major extensions of the basic object model

As an extension of the conventional object model(s) the RTO.k object model is :
a) independent of the language (textual or graphic) used to program or specify object designs, and
b) independent of the way inheritance is facilitated.
Since one can envision the emergence of multiple concrete language constructs based on the current RTO.k object model, the model discussed here can be viewed as a framework for an evolving model family.

The basic structure of the RTO.k object model is depicted in Figure 1. The RTO.k object model is an extension of the conventional object model(s) in four essential ways:
(a) For some methods of an RTO.k object, a real-time clock serves as the mechanism for triggering the method executions as the clock reaches some values specified at the design time and such methods are called time-triggered (TT-) methods, also called the spontaneous methods (SpM's), and *clearly* separated from the conventional service methods (SvM's) triggered by messages from clients. Actions to be taken when the real-time clock reaches some values *which can be determined at the design time* can appear only in SpM's;
(b) A concurrency constraint which prevents conflicts between TT-methods and message-triggered methods is incorporated. Basically, *activation of a service method triggered by a message from an external client is allowed only when potentially conflicting TT-method executions are not in place.* To be exact, when a message-triggered service method is not free of conflict in accessing the same portion of the object data space (ODS) with a TT-method, execution of the former (message-triggered) method must not be allowed in a time zone earmarked for a TT-execution of the latter (spontaneous) method. This restriction is called the basic concurrency constraint (BCC). Therefore, TT-methods are given higher priorities for execution over the message-triggered methods. Note that this BCC does not impose any restriction on concurrent execution of TT-methods or concurrent execution of message-triggered methods;
(c) For each execution of a method of an RTO.k object, a deadline is imposed;
(d) Real-time data contained in an RTO.k object become invalid after the interval called the maximum validity duration passes.

In Figure 1, the two distinct types of methods for operating on its *object data space* (also called the *object data store*), the SpM's and the message-triggered SvM's, are shown. The clear-cut separation of SpM's from SvM's is an important feature of the RTO.k object model.

The two types of methods are different not only in the way their executions are triggered but also in that actions of the type "at constant-clock-value do S" or the type "sleep-until constant-clock-value" can appear only in SpM's. For example, if a value computed by an SvM needs to be output precisely at a certain time, then the RTO.k object must be designed such that a request is made by the producer SvM to an SpM to carry out the output action. The exact mechanism by which such a request is conveyed will become clearer in the next subsection.

2.2 Autonomous activation condition (AAC)

Triggering times for SpM's must be *fully specified as constants* during the design time. Those real-time constants appear in the first clause of an SpM specification called the autonomous activation condition (AAC) section. The AAC may be specified in the following form.

```
ab      "AAC-begin"
{ [AAC name:]
    for <time-var> = from <activation-time>
              to <deactivation-time>
              [every <period>]
    start-during
         (<earliest-start-time>,
           <latest-start-time>)
    finish-by<deadline>
}*
ae      "AAC-end"
```

where the "star" expression x* or {x}* is a regular expression for the set {NULL, x, xx, xxx, ...}.

For example, consider the following case.

```
ab
AAC-1:
    for T =  from 10:00:00.000am
         to 11:00:00.000am
         every 0.005sec
    start-during  (T, T+0.001sec)
    finish-by     T+0.003sec
ae
```

The above AAC-1 specifies : "This SpM must be executed every 5 msec starting at 10 am until 11 am and each execution must start at any time within the one millisecond interval (T, T + 0.001sec) and must be completed by T + 0.003sec."

Note that "for t = from 10am to 10am start-during (t, t+5min)" has the same effect as "start-during (10am, 10:05am)". Note also that the AAC section may contain multiple unnamed or uniquely named AAC clauses.

The execution engine uses the AAC of an SpM to generate future execution triggering schedules for the SpM at appropriate times and inserts them into the reservation queue (depicted in Figure 1) of the SpM. Each reservation queue is a sorted queue in which the nearest future triggering schedule is at the head of the queue.

A provision is also made for making the AAC section of an SpM contain only candidate triggering times, not actual triggering times, so that a subset of the candidate triggering times indicated in the AAC section may be dynamically chosen for actual triggering. Such a dynamic selection occurs when an SvM within the same RTO.k object requests future executions of a specific SpM. Again, candidate triggering times must be fully specified as constants during the design time. An SvM requests future executions of an SpM by placing a reservation into the reservation queue associated with the SpM. The reservation queue holds future execution triggering schedules for the SpM determined up to the present. To simplify the job of the compiler, a reservation by an SvM for execution of an SpM must include the name of an AAC clause appeared in the SpM definition. The part of the AAC section containing AAC clauses that specify candidate triggering times rather than actual triggering times starts with a declaration "if-demanded". For example,

```
ab    if-demanded
a1:  start-during  (10am, 10:05am)
        finish-by  10:15am;
a2:  start-during  (11am, 11:05am)
        finish-by  11:15am  ae
```

defines two candidate triggering times. An SvM may then call this SpM, say SpM-1, as follows.

```
if current-time < 9:30am
    then  request SpM-1(AAC=a1)
    else  request SpM-1(AAC=a2).
```

Therefore, there are two different modes of determining triggering times for SpM's:
(a) fully determined during the system design, in which case the SpM is said to be statically scheduled, and
(b) determined during the run time when an SvM requests executions of the SpM and designates a subset of the candidate triggering times prepared during the design time as actual triggering times, in which case the SpM is said to be partially dynamically scheduled.

On the other hand, actions to be taken when the real-time clock reaches values which cannot be determined at the design time may appear in SvM's, even if the actions may have to be executed periodically. Therefore, an SvM may include a statement of the type "for X seconds every Y seconds do S". However, the start time of this statement execution is not known until the SvM is called by a client.

2.3 Interactions among RTO.k objects

An underlying design philosophy of the RTO.k object model is that a real-time DCS will always take the form of a network of RTO.k objects. RTO.k objects interact via calls by client objects for SvM's in server

objects. The caller may be an SpM or an SvM in the client object. In order to facilitate highly concurrent operations of client and server objects, non-blocking (sometimes called asynchronous) types of calls (i.e., service requests) in addition to the conventional blocking type of calls can be made to SvM's.

Therefore, the following two basic types of calls can be made to SvM's in the server RTO.k object.

(a) Blocking call: After calling an SvM, the client waits until a result message is returned from the SvM. The syntactic structure may be in the form of

Obj-name.SvM-name(parameter-1, parameter-2, ...).

Since the client and the server object may be resident in two different processing nodes, this call is in general implemented in the form of a remote procedure call. Even if there is no result parameter in the SvM, the execution completion signal is returned to the client;

(2) Non-blocking call: After calling an SvM, the client can proceed to follow-on steps (i.e., statements or instructions) and then waits for a result message from the SvM. The syntactic structure may be in the form of

Obj-name.SvM-name(parameter-1, parameter-2, ..,
mode NWFR, timestamp TS);
----- statements -----
get-result Obj-name.SvM-name(TS) by deadline;

The mode specification "NWFR" which is an abbreviation of "No-Wait-For-Return" indicates that this is a non-blocking call. When the client calls the SvM, the client records a time-stamp into a variable, say TS. The time-stamp uniquely identifies this particular call for the SvM as distinct from other (past or future) calls for the same SvM from this client. Therefore, later when the client needs to ensure by execution of the "get-result" statement the arrival of the results returned from the earlier non-blocking call for the SvM, not only the SvM-name but also the variable TS containing the time-stamp associated with the subject call must be indicated. When a client makes multiple non-blocking calls for SvM's before executing a "get-result" statement, the time-stamp unambiguously indicates to the execution engine which non-blocking call is referred to.
If the results have not been returned at the time of executing the "get-result" statement, the client waits until the execution engine recognizes the arrival of the results. A non-blocking call thus creates concurrency between a client (SpM or SvM) and a server SvM which lasts until the execution of the corresponding "get-result" statement. In some situations, a client does not need any result from a non-blocking call for an SvM. Such a client does not use a "get-result" statement.

2.4 Concurrency and working ODS specification

The following types of concurrency can be exploited in execution of object methods in an RTO.k object:
(a) Concurrency among SpM executions;
(b) Concurrency among SvM executions;
(c) Concurrency between SpM executions and SvM executions.

Concurrency among the SpM's is specified in an implicit but natural manner, e.g., two SpM's designed to be triggered at 10 am. In general, if the specified execution periods of two SpM's are in overlap, then there is clearly concurrency implied in the two SpM specifications.

The approach adopted in the RTO.k object model for exploiting concurrency of type-b and type-c is to explicitly declare the portion of the object data space (ODS) used by each method and allow concurrent execution of methods whenever there is no data conflict. Therefore, the ODS-section in the RTO.k class consists of declarations of ODS-segments (ODSS's). Each ODS-segment is explicitly named. The specification of each object method then includes its working ODS specification which indicates the set of ODS-segments that can be operated on during the execution of the method. This specification also indicates if each ODS-segment can be accessed for "read-only" or "read-&-write" purpose. This facilitates detection of potential data conflicts among the object methods that need to be executed. Each ODS-segment is thus an atomic storage unit specified explicitly as such by the designer. The sizes of such ODS-segments determine the degree of parallelism that can be exploited in execution of object methods.

During the design time the working ODS specifications are used for detection of errors in SpM specifications. If two SpM's are specified for concurrent execution while both SpM's have "read-&-write" privileges for the same ODS-segment(s), then the SpM specifications are incorrect.

During the execution time the working ODS specifications are used by the execution engine to determine whether SvM's requested by clients can be initiated immediately or should be delayed until some potentially conflicting methods are completed. For example, suppose the working ODS specification of an SvM, SvMx, consists of {(ODSS-1, RW)} where RW and RO indicate the "read-&-write" right and the "read-only" right, respectively. Once a client calls for SvMx and the execution engine starts evaluating the possibility of initiating the SvMx execution, the engine checks if any SpM or SvM currently in execution has an access right (RO or RW) for ODSS-1. If so, then SvMx cannot be initiated yet. Otherwise, the engine checks if any SpM which will have an access right for ODSS-1 is going to be triggered before t + MET(SvMx) where t represents the current time and MET(SvMx) represents the maximum execution time estimate for SvMx. Again, if so, SvMx cannot be initiated yet. Otherwise, SvMx is initiated.

To facilitate exploitation of additional concurrency among SvM executions at the cost of increased burdens on the object designer for determining the worst-case service time, access modes for the working ODS-segments of SvM's may be "unspecified" and instead, each ODS-segment may be encapsulated within a CREW (concurrent-read-&-exclusive-write) monitor which is an

extension of the monitor in [Bri73] and possesses the readers-writers semantics [Bri73]. In this case, an SvM called by a client can be initiated as long as there is no potential data conflict with an SpM. Potential data conflicts with another SvM can be ignored since shared ODS-segments are safely protected within CREW-monitors.

Suppose three object methods in the same RTO.k object have the following working ODS specifications.

Working ODS of SpM1: {(ODSS1, RO), (ODSS2, RW)}
Working ODS of SvM1: {(ODSS1, unspecified)},
Working ODS of SvM2: {(ODSS1, unspecified)}.

Suppose also that SvM1 has been called by a client, the current time is 10am, SpM1 is to be triggered at 10:05am, and MET(SvM1) is 10 minutes. SvM1 cannot be initiated at this time because the access mode of SvM1 for ODSS1 is unspecified and thus can be RW and also because if SvM1 is initiated now, there is possibility of the SvM1 execution being unfinished at 10:05am at which time SpM1 will be triggered. Suppose SpM1 is completed at 10:10am. At this time, SvM1 is initiated. If SvM2 is called by a client at 10:12am and SvM1 is still in execution at this time, SvM2 can be initiated as long as there is no potential data conflict between SvM2 and an SpM. This is because ODSS1 is encapsulated within a CREW-monitor and thus can only be shared harmoniously by SvM1 and SvM2.

2.5 Design-time guarantee of timely service capabilities of RTO.k objects and specification of maximum service times

An important underlying design philosophy of the RTO.k object model is that a real-time DCS will always take the form of a network of RTO.k objects and

each RTO.k object should be maximally autonomous in timing its actions while providing dependable services to client RTO.k objects.

Therefore, the designer of an RTO.k object can view SpM's as internal capabilities and SvM's as services advertised to all potential clients.

The designer of each RTO.k object provides a guarantee of timely service capabilities of the object by indicating the *deadline for every output* produced by each SvM (and each SpM which may be executed on requests from SvM's) in the specification of the SvM (and some relevant SpM's) advertised to the designers of potential client objects. An output action here may be
(a) an updating of a portion of the ODS,
(b) sending a message to either another RTO.k object (which may or may not be the client) or a device shared by multiple objects, or
(c) placing a reservation into the reservation queue for a certain SpM.

The specification of each SvM which is provided to the designers of potential client RTO.k objects must contain at least the following:
(a) an *input specification* that consists of
(a1) the types of input parameters that the server object can accept and
(a2) the maximum request acceptance rate, i.e., the maximum rate at which the server object can receive service requests from client objects;
(b) an *output specification* that indicates the *maximum delay* (not the exact output time) and the *nature of the output value* for every output produced by the SvM. In a sense, the maximum among all the maximum delays associated with output actions expected from the SvM is the maximum service time of the SvM.

If service requests from client objects arrive at a server object at a rate exceeding the maximum acceptance rate indicated in the input specification for the server object, then the server may return exception signals to the client objects. The system designer can prevent such "overflow" occurrences by careful design or provide exception handlers if he/she is certain that exception handlers will never fail to achieve the application goals satisfactorily (in absence of component failures).

The specification of the maximum delay for an output from an SvM is a serious commitment on the part of the server object designer to the designers of potential client objects. It is a *guarantee of timely service*. Before determining the maximum delay specification, the server object designer must convince himself/herself that with the object execution engine (hardware plus operating system) available, the server object can be implemented to always execute the SvM such that the output action is performed within the maximum delay (in absence of component failures). This means that the server object designer must consider
(1) the worst-case delay from the issuance by a client object of a service request to the initiation of the corresponding SvM by the server object, and
(2) the worst-case execution time for the SvM from its initiation to each of its output actions.

On the other hand, a client RTO.k object imposes a deadline on the server RTO.k object for creation of all the intended computational effects (i.e., all intended output actions). The deadline imposed by a client must not be earlier than the deadline adopted and advertised by the designer of the server object for completion of the corresponding SvM.

The specifications of the SpM's which may be executed on requests from the SvM must also be provided to the designers of the client objects which may call the SvM. The specification of such an SpM must contain at least the following:
(1) an *autonomous activation condition (AAC)*, and
(2) an *output specification*.

There is no input specification in an SpM specification but the output specification for an SpM indicates, for every output expected from the execution of an SpM, the exact time *at* or *by* which it will be produced and the nature of every value carried in the output action. As an example, one SpM can be designed to "update a specific data item

in the object data store" (nature of output) "at 10:30am (time for output) if the SpM started its execution between 10:10am and 10:15am.

The basic concurrency constraint (BCC) was incorporated into the RTO.k object model to ease the design-time guarantee of timely service capabilities. At least it makes it very easy to analyze the execution time behavior of SpM's. Executions of SpM's are not disturbed by SvM executions and triggering times of SpM's are fixed at the design time. For example, if a statement of the type "at 10am do S" appears in an SpM, its reliable execution can be easily assured. Therefore, if most of the computations are done by SpM's and only simple client interface functions are handled by SvM's, then it should be easier to guarantee timely service capabilities than in the cases of objects in which most computations are done by SvM's and frequent competition among SvM's occurs for access to the same portion of the ODS.

2.6 Maximum validity duration

The ODS-section consists of declarations of ODS-segments and each ODS-segment declaration consists of variable declarations. Here maximum validity durations (MVD's) can be associated with individual variables as follows.

<type-specifier> <identifier> [during <effective-period> | by <expiration-time>]

For example,

"int K during 5 msec"

indicates that a new value stored into the integer-type variable K will be valid for use only for 5 milliseconds. The MVD specification can be used by error detectors planted in the execution engine.

2.7 Overall structure of the object method specification

A model of a language construct for structuring RTO.k objects, the *RTO.k class*, was defined in [Kim94b]. An RTO.k class definition consists of four major sections.

RTO_class = class
begin

"connectivity_section :"
list of RTO_name;
"that can be called upon"

"object_data_space_section :"
list of object_data_space_segment;

"spontaneous_method_section :"
list of spontaneous_method;

"service_method_section :"
list of service_method

end;

The overall structure of the SpM specification in an RTO.k class may be as follows.

SpM-name <*name*>
using-SpM <*SpM-name*>*
using-data (<*ODS-segment-name*>, <*access-mode*>)*
< AAC section >
finish-by <*deadline*>
begin
<*method-body*>
end;

Here the "using-SpM" section lists the names of SpM's within the same RTO.k object to which activation requests can be sent by this SpM.

The overall structure of the SvM specification in an RTO.k class may be as follows.

SvM-name <*name*> (<*parameter specification*>)
using-services <*RTO-name.SvM-name*>*
using-SpM <*SpM-name*>*
using-data (<*ODS-segment-name*>, <*access-mode*>)*
[finish-within <*duration-limit*> |
finish-by <*deadline*>]
begin
<*method-body*>
end;

Here timely completion requirements can be specified in two different ways but the specification of the method completion deadline is an expression which cannot be fully evaluated until the execution-start time of the SvM is determined.

2.8 Extended mode interactions among RTO.k objects and among object methods within the same object

In addition to the two basic types of interactions among RTO.k objects, the blocking-call for an SvM and the non-blocking call for an SvM, a major variation of each of the two types is also adopted into the RTO.k object model for the sake of reduced communiction overhead in some situations. The essence of this extension is to allow an arrangement in which a client calls an SvM and then receives results from another SvM. The main motivation for facilitating this stems from the basic concurrency constraint which requires an execution of any SvM to be made on a stand-by basis, i.e., only when a sufficiently large time window between SpM executions opens up.

Therefore, if the maximum execution time estimate of an SvM, say SvMx, is MET(x) seconds and the SpM's are so frequently executed that a time window larger than MET(x) never opens up, then the SvM may never be executed. One way to get around this problem is to divide such an SvM, SvMx, into multiple smaller SvM's, SvMx1, SvMx2, ---, SvMxk. A client must then call each

smaller SvM at a time. Calling each smaller SvM incurs communication overhead for transmitting a call message from the client to the SvM and a result message from the SvM to the client. In general, these inter-object messages can involve much larger delays than intra-object messages. Naturally, one can conceive of an arrangement in which a client calls the first SvM (SvMx1), the latter in turn "passes its client" to the second SvM (SvMx2) as it completes its execution, and this continues until the last SvM (SvMxk) is executed and then returns the results to the inherited client which is the external client that called the first SvM. Passing the client from an SvM to the next SvM involves intra-object messages although such a client-transfer call message goes through the same service request queue which service requests from external clients go through.

A client-transfer call may involve passing parameters in an explicit manner as done in the case of a call by an external client or passing information through the shared data structures in the ODS. The syntactic structure for such a client-transfer call for an SvM may be in the form of

SvM-name(parameter1, parameter2, ---, mode CT,
external parameter-x1, parameter-x2, ---).

The mode specification "CT" which is an abbreviation of "Client Transfer" indicates that this is a client-transfer call. The parameters listed in the "external" clause are those which were used in the interface between the caller and the caller of the caller. Normally those parameters are inherited from the interface between the external client (i.e., located outside the subject server object) and the first SvM called by the client.

As a part of executing this client-transfer call for an SvM, the execution engine terminates the caller SvM, places a request for execution of the called SvM into the service request queue, and establishes the return connection (i.e., the connection to the client) from the called SvM to the client of the caller SvM that has just been terminated. When the "return" statement in the called SvM is executed, the results are returned through the return connection established. Since the external client which called the first SvM cannot predict from which SvM it will receive returned results, it must be implemented to accept results returned from any SvM.

There is no reason why this client-transfer call cannot be extended to the case of calling an SvM in another RTO.k object. The syntactic structure for such a client-transfer call for an external SvM may be in the form of

Obj-name.SvM-name(parameter1, parameter2, ---,
mode CT, external parameter-x1, ---).

Besides the above client-transfer call for an external SvM, no other language constructs need be introduced due to the adoption of the extended-mode interaction among RTO.k objects. This is because accepting results returned from the called SvM is a special case of accepting results returned from any SvM in the system. Moreover, under our philosophy of making the details of a server object transparent to the designer of a client, the client calling an external SvM need not be able to distinguish between the case where results are returned from the called SvM and the case where results are returned from another SvM.

3. Distinguishing characteristics of the RTO.k object model

In this section, major differences between the RTO.k model and other proposed extensions of the basic object model are discussed together with the costs and benefits which each distinct feature brings to the RTO.k model.

3.1 Clear-cut distinction between SpM and SvM

All real-time extensions of the basic object model proposed so far are active objects with some time constraints added. An active object is an object with its own thread of execution control. Therefore, multiple active objects can exhibit concurrency among their activities. Of all the models proposed as real-time extensions of the basic object model, about half of them do not provide anything resembling SpM's. Such models will not be discussed any further in this paper [Bih89, Cha90, Wol91]. In the object model adopted in the RTC++ project [Ish90, Ish92] and other models [Att91, Her92, Shi91], the clear-cut distinction made in the RTO.k model between SpM's and SvM's is not done. That is, the rule, "actions to be taken when the real-time clock reaches values which can be determined at the design time can appear only in SpM's", is not adopted in any of those models. This rule was adopted in the RTO.k to simplify the task of guaranteeing the timely service offered by the RTO.k object. Also, the number of SpM executions that can proceed in parallel has no fixed limit in the RTO.k model unlike in models such as the MO2 model [Att91] and others [Shi91].

In some models providing something similar to SpM's, interactions between those corresponding to SpM's and SvM's are not facilitated [Her92, Shi91, Tak92].

3.2 Basic concurrency constraint

The MO2 model proposed in [Att91] is the only other model which contains something resembling the basic concurrency constraint. However, the MO2 model allows only one SpM in an object. Also, an SvM in an RTO.k object is not initiated if it has the potential of running into data conflicts with any SpM in execution or with any SpM scheduled. On the other hand, SvM's and the SpM in an MO2 object may be in concurrent execution with intermittent competition for accessing data in the ODS. Therefore, the RTO.k model sacrifices some fine degree of parallelism for the sake of ease in guaranteeing timely services of objects.

3.3 Design-time guarantee of timely service of each object

The RTO.k object model was formulated with this specific objective of facilitating design-time guarantee of timely services of each object in mind. For example, the AAC section in the SpM declaration is restricted to be an expression which can be fully evaluated at design time. The execution engine which contains the operating system and both the inter-object communication facility and the intra-object inter-method communication facility, is required to yield easy analysis of the worst-case execution time for any local or remote method execution. It appears that this "conservative" policy was not adopted in any other model, or at least not pursued to the extent done in the RTO.k model. However, the conservative policy of the RTO.k model can result in some sacrifice of hardware utilization in comparison to the case of using "liberal" policies adopted in other models.

Having recognized the desirable characteristics of the execution engine, an execution engine model called the DREAM (Distributed Real-Time Ever Available Microcomputing) engine was formulated and its first prototype, DREAM kernel v1.0, was implemented on a PC LAN equipped with Intel 80486 processors, DOS-BIOS device drivers, the Packet Ethernet driver, and an interprocess multicast communication manager called the HU data field subsystem [Mor93, Kim95]. Services of the DREAM kernel including process management services can be obtained from within a C++ program (representing an implementation of an RTO.k object) via calls for DREAM library routines.

3.4 Interactions among objects

In some models, only the blocking type of service call is facilitated although the developers probably considered the issue of allowing non-blocking service calls a minor issue. In the MO2 model [Att91], the non-blocking service call is facilitated. The client-transfer call mechanism in the RTO.k object model is not available in any other model. The need for allowing client-transfer calls in the RTO.k model arose mainly due to the adoption of the basic concurrency constraint and the approach of design-time guarantee of timely services.

3.5 Maximum validity duration (MVD)

The specification and run-time checking of maximum validity durations were not adopted in any model but the RTO.k model. The usefulness of this facility is felt normally during the program/system validation. It should also be noticeable during the development of fault-tolerant hard-real-time systems.

3.6 Main differences between the RTO model in RTC++ and the RTO.k object model

Overall the object model in RTC++ [Ish92] and the MO2 model [Att91] are closer in nature to the RTO.k model than other models are. Since the differences between the RTO.k model and the MO2 model have been pointed out clearly in the above discussion, the RTC++ approach is reviewed in more detail here and compared against the RTO.k object model.

In RTC++, an RTO is declared as an active object and contains one or more locally possessed threads called *master threads* and incorporates specifications of timing constraints imposed on object methods and individual statements. It also defines a finite set of *slave threads* responsible for executing object methods called by clients. Each object is assigned a fixed priority and the priority of a client object is *inherited* by the server object during the execution of the method called by the client.

Main differences between the RTO model in RTC++ and the RTO.k model are the following:
(a) In RTC++, master threads, which are counterparts for the SpM's (spontaneous methods) in the RTO.k object, are not clearly separated in their roles from SvM's to the extent that SpM's in the RTO.k object are separated from SvM's. For example, SvM's in RTC++ cannot directly request executions of master threads but instead can perform by themselves all computing actions which would be done by SpM's in an RTO.k object.
(b) The basic concurrency constraint (BCC) can not be adopted in RTC++ since the approach of assigning fixed priorities to RTO's and the priority inheritance approach were adopted. Therefore, master threads cannot have higher priorities than SvM's in RTC++, which is the opposite of the BCC approach in the RTO.k model.
(c) No priorities are assigned to RTO.k objects. How to order competing accesses by object methods in execution for the same portion of the ODS is left to the execution engine which should utilize in its ordering decision the current information on time constraints associated with the competing methods.

4. Potential roles of the RTO.k structuring in engineering of real-time systems

Potential useful roles of the RTO.k model in engineering of complex real-time DCS's are now briefly discussed.

4.1 Uniform structuring of control computer systems and application environment specifications / simulators

The RTO.k object supports an interesting style of simulation. Suppose the application environment chosen is a sky+land+sea segment of interest, called the "theater", and any moving objects in that theater including ships and airplanes, etc. This environment can be represented and simulated by the RTO.k object in Figure 2.

The ODS in this RTO.k object contains state representations of the airplanes, the ships, and the theater space. This single RTO.k representation can be expanded into a representation in the form of a network of RTO.k objects, each representing an airplane or ship.

Each TT-method, when executed, updates a variable-set in the ODS representing the state of some physical object (i.e., airplane, ship) to reflect the current state of the physical object. Ideally the TT-methods should be *activated continuously* and each of their executions be *completed instantly*. This is fine when the object is used merely as a description rather than an executable program. However, such an execution engine for the RTO.k object cannot exist and thus we must adopt the less precise version of the model in which the time domain is a discrete domain, as an executable simulation model. That is, the limited power of the simulation engine dictates the activation frequency of any TT-method to be no more than once per every simulator clock tick while allowing each execution to be completed before or by the time of the following activation of the same method. Therefore, TT-methods are the mechanisms for simulating continuous state changes that occur naturally in the environment objects. The natural parallelism that exists among the environment objects is precisely represented by use of multiple TT-methods which may be activated simultaneously. In general, the accuracy of an RTO.k object structured simulation of the environment is a direct function of the activation frequencies of TT-methods.

Therefore, the RTO.k object model is an effective mechanism not only for variable-degree abstraction of real-time DCS's under design but also for variable-accuracy simulation of the application environments. Structures and notations used will be of the same kind in both control system design and environment simulation. The combination of a control system design and an environment simulator takes the form of a network of RTO.k objects. A specification and implementation experiment that involved an application of this RTO.k based uniform structuring scheme to both the development of a defense system and that of an environment simulator was conducted recently [Kim94b]. This involved the use of the DREAM kernel (v1.0) and concurrent programming in C++. This experiment confirmed the effectiveness of the uniform structuring approach and reinforced our belief that the RTO.k model offered an efficient and rigorous way to develop complex real-time systems.

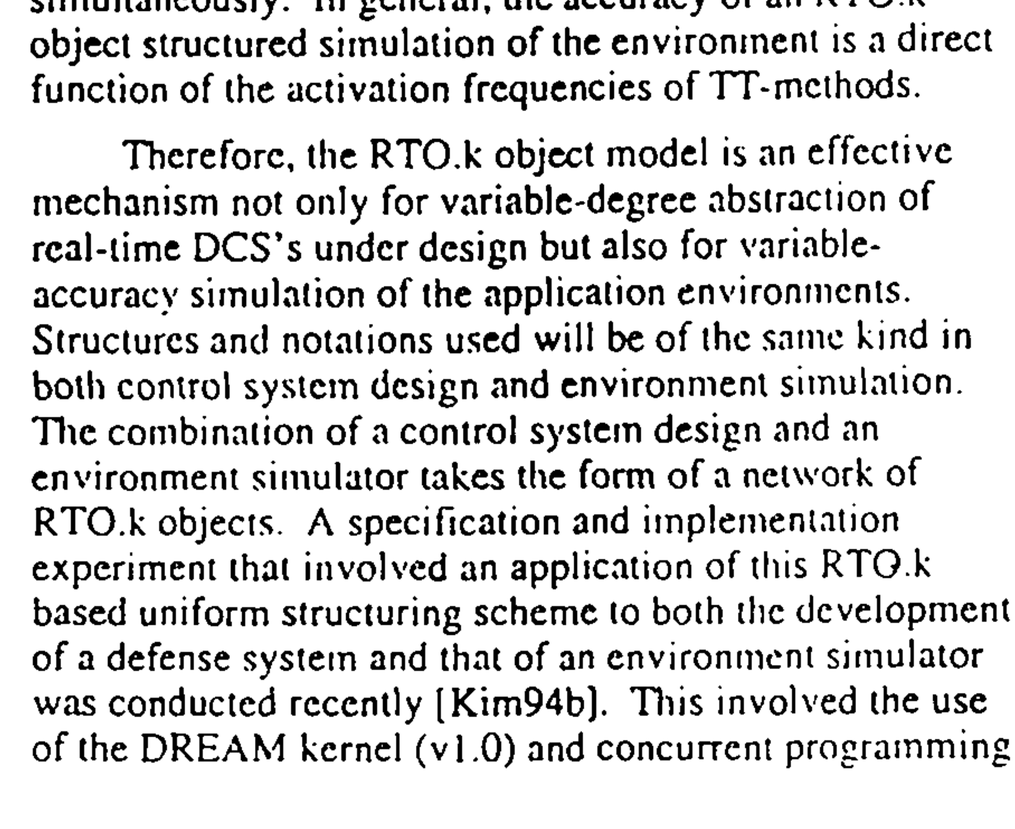


Figure 2. An RTO.k structured simulation model

We believe that the uniform structuring and accurate representation capabilities of the RTO.k object model can have positive consequences in all major phases of the real-time system engineering cycle such as the following :
(1) Requirement specification,
(2) High-level design,
(3) Stepwise refinement of a design to an implementation,
(4) Validation, and
(5) Maintenance.
Attempts to validate this hypothesis are considered highly meaningful subjects for future research.

4.2 Globally optimal resource allocation

We believe that the RTO.k structured specification of the requirements imposed on control computer systems provides much help in taking accurate global views of the temporal aspects of distributed cooperative computations. With such global views, systematic approaches to global resource allocation may become feasible. Some discussions on the issues to be resolved in realizing globally optimal resource allocation are given in [Kim94a].

5. Conclusion

We feel that the RTO.k object model offers some promises in achieving the *uniformity* and flexible and unrestricted degrees of *accuracy* in the representation of both application environments and control computer system designs evolving during the system development cycle. Realization of the full potential will require a great deal of future research of both analytical and experimental nature.

Acknowledgment: The research work reported here was supported in part by US Navy, NSWC Dahlgren Division under Contract No. N60921-92-C-0204, in part by the University of California MICRO Program under Grant No. 93-080, and in part by Hitachi, Ltd.

References

[Att91] Attoui, A. and Schneider, M., "An Object Oriented Model for Parallel and Reactive Systems", Proc. IEEE CS 12th Real-Time Systems Symposium, 1991, pp. 84-93.

[Bih89] Bihari, T., Gopinath, P., and Schwan, K., "Object-Oriented Design of Real-Time Software", Proc. IEEE CS 10th Real-Time Systems Symp., 1989, pp.194-201.

[Boo91] Booch, G., 'Object-Oriented Design', Benjamin Cummings, CA, 1991.

[Cha90] Champlain, M., "Synapse: A Small and Expressive Object-based Real-time Programming Language", SIGPLAN Notices, Vol. 25, No. 5, 1990, pp. 124-134.

[Dah72] Dahl, O.J., "Hierarchical Program Structuring", in Dahl, Dijkstra, & Hoare eds., 'Structured Programming', Aca. Press, NY, 1972.

[Her92] Hernandez, J. and Sanchez, J. A., "RT - MODULA2: An embedded in MODULA2 Language for writing Concurrent and Real Time programs", ACM SIGPLAN Notices, Vol. 27, No. 2, February 1992, pp. 26-36.

[Ish90] Ishikawa, Y., Tokuda, H., and Mercer, C. W., "Object-Oriented Real-Time Language Design: Constructs for Timing Constraints", Proc. ECOOP/OOPSLA '90, October 1990, pp. 289-298.

[Kim93] Kim, K.H. and Bacellar, L.F., "A Real-Time Object Model: A Step toward an Integrated Methodology for Engineering Complex Dependable Systems", Proc. 1993 Complex System Engineering Synthesis and Assessment Technology Workshop, US Navy NSWC, July 1993, pp.56-64.

[Kim94a] Kim, K.H. et al., "A Methodology Framework for Optimal Design of Real-Time Dependable Computer Systems", Proc. CSESAW '94 (1994 Complex Systems Engineering Synthesis and Assessment Technology Workshop), US Navy NSWC, Dahlgren Div., July 1994, pp.251-259.

[Kim94b] Kim, K.H. and Kopetz, H., "A Real-Time Object Model RTO.k and an Experimental Investigation of Its Potentials", Proc. 1994 IEEE Computer Society's Computer Software and Applications Conf. (COMPSAC), Nov. 1994, Taipei, pp.392-402.

[Kim95] Kim, K.H., Mori, K., and Nakanishi, H., "Realization of Autonomous Decentralized Computing with the RTO.k Object Structuring Scheme and the HU-DF Inter-Process-Group Communication Scheme", to appear in Proc. 1995 IEEE CS Int'l Symp. on Autonomous Decentralized Systems (ISADS), April 1995, Phoenix.

[Kop88] Kopetz, H. and Kim, K.H., "Consistency Constraints in Distributed Real Time Systems", in M.G. Rodd and T.L. d'Epinay eds., 'Distributed Computer Control Systems 1988', Pergamon Press, 1989, pp.29-34.

[Kop90] Kopetz, H. and Kim, K.H., "Temporal Uncertainties in Interactions among Real-Time Objects", Proc. IEEE CS 9th Symp. on Reliable Distributed Systems, Oct. 1990, pp.165-174.

[Mer90] Mercer, C.W. and Tokuda, H., "The ARTS Real-Time Qbject Model", Proc. IEEE CS 11th Real-Time Systems Symposium, 1990, pp. 2-10.

[Mor93] Mori, K., "Autonomous Decentralized Systems: Concept, Data Field Architecture, and Future Trends", Proc. Int'l Symp. on Autonomous Decentralized Systems (ISADS 93), Mar. 1993, Kawasaki, Japan, pp. 28-34.

[Rum91] Rumbaugh, J. et al., 'Object-Oriented Modeling and Design', Prentice Hall, New Jersey, 1991.

[Shi91] Shrivastava, S.K. and Waterworth, A., "Using Objects and Actions to provide Fault Tolerance in Distributed, Real-Time Applications", Proc. IEEE CS 12th Real-Time Systems Symposium, 1991, pp.276-285.

[Tak92] Takashio, K., and Tokoro, M., "DROL: An Object-Oriented Programming Language for Distributed Real-Time Systems", Proc. OOPSLA, 1992, pp. 276-294.

[Wol91] Wolfe, V., Davidson, S., and Lee, I., "RTC: Language Support For Real-Time Concurrency", Proc. IEEE CS 12th Real-Time Systems Symposium, 1991, pp. 43-52.

Proceedings of Words'94

The First Workshop on Object-Oriented Real-Time Dependable Systems

October 24–25, 1994 Dana Point, California

Sponsored by

The IEEE Computer Society
Technical Committee on Distributed Processing

In cooperation with

The IEEE Computer Society — Orange County Section

IEEE Computer Society Press
Los Alamitos, California

Washington • Brussels • Tokyo

Appendix B.

[Kim95b] Kim, K.H., "Toward New-Generation Real-Time Object-Oriented Computing", *Proc. IEEE CS 5th Workshop on Future Trends of Distributed Computing Systems (FTDCS)*, Cheju Island, Aug. '95, pp.520-529.

Proc. FTDCS '95 (IEEE Computer Society's 5th Workshop on Future Trends of Distributed Computing Systems), Cheju Island, Aug. '94, pp.520-529.

Toward New-Generation Real-Time Object-Oriented Computing

K. H. (Kane) Kim

Department of Electrical & Computer Engineering
University of California
Irvine, CA 92717
U.S.A.
Kane@Ece.Uci.Edu

Abstract

In recent years, the market conditions for the real-time computer system (RTCS) technology and the hardware economy have been showing significant changes and thus this author feels that time is ripe for vigorously pursuing new paradigms in designing and structuring RTCS's. Pursuing the new paradigms is called the "new-generation (NG) or second-generation (2G) real-time computing" approach in this paper and this approach has the flavor of an idealistic perfectionist approach. The essence of the NG real-time computing is to realize real-time computing in a general manner not alienating the main-stream computing industry and yet allowing system engineers to confidently produce certifiable RTCS's for safety-critical applications. After a discussion on the new real-time computing paradigms, two examples of technical approaches formulated to realize the NG real-time computing, the RTO.k object structuring scheme and the DREAM kernel model, are reviewed in this paper. Issues for future research in this area are also discussed.

Index Terms: real-time, new generation, real-time object, RTO.k, guaranteed timely service, DREAM kernel, micro-kernel, thread, process.

1. Introduction

Significant improvements achieved in recent years in hardware economy and component reliability have sprurred the growth in both the volume and the variety of the market for computer applications. One of the computer application fields which started showing noticeable new growth trends is the real-time computing application field. A newly raised important technological issue in this field as well as in other application fields is: "Is it possible to achieve *more efficient production* of *more reliable* computer(-based application) systems by wise use of more hardware ?".

In order to achieve noticeable progresses in the design efficiency and the system reliability attained, this author believes that the following real-time computing paradigms, which may be called the new-generation (NG) real-time computing paradigms must be vigorously pursued [Kim94c]:

(1) *General-form design style*: Future real-time computing must be realized in the form of a generalization of the non-real-time computing, rather than in a form looking like an esoteric specialization.

(2) *Design-time guarantee of timely service capabilities of subsystems*: To meet the demands of the general public on the assured reliability of future RTCS's in safety-critical applications, there does not appear to be any adequate way but to require the system engineer to produce design-time guarantees for timely service capabilities of various subsystems (which will take the form of objects in object-oriented system designs).

To realize this idealistic form of real-time computing, a powerful structuring scheme capable of dealing with all practically useful real-time and non-real-time computing requirements must be established. Preferably, such structuring methods should be of object-oriented (OO-) type, considering the modularity, generality, and natural abstraction benefits that object-oriented approaches bring in. Indeed, OO-structuring has had minimal impacts in real-time computer system (RTCS) engineering in contrast to its pervasive use in non-RTCS engineering. The main reason in this author's opinion is because an effective approach for obtaining timeliness-guaranteed OO-designs has not been available in industry. However, this appears to be merely a research issue of short-term nature, not a fundamental obstacle.

Indeed in the last several years there has been a growing trend of research activities aimed for extending the conventional OO-structuring approaches to support RTCS design [Att91, Bih89, Ish92, Kim94b, Kop90, Shi91, Tak92]. However, most of those works have not been aimed for supporting the design-time guarantee of timely service capabilities of objects, which is one of the

fundamental requirements of the NG real-time computing.

In recent years, the author and his collaborating researchers together established a real-time OO-structuring approach with the purpose of facilitating the NG real-time computing. The approach is called the RTO.k object structuring scheme [Kim94b, Kim94c, Kim94d]. The goal of the scheme cannot be fully realized without establishing supporting execution engines, programming language tools, and other design aids. The most urgently needed among them are the execution engines. An execution engine must possess guaranteed timely service capabilities. Otherwise, timely service capabilities of RTO.k objects cannot be guaranteed. In this paper, a recently formulated model of an execution engine, named the DREAM (Distributed Real-time Ever Available Micro-computing) kernel, is presented in an abridged form. The architecture of the DREAM kernel as well as the principle of rigid time management inside the kernel adopted to enable guaranteering timely services, is discussed.

The DREAM kernel is actually a general purpose operating system kernel which supports not only RTO.k objects but also conventional concurrent programs consisting of processes and shared data structure monitors.

We expect that active development of real-time OO system structuring techniques, such as the RTO.k object structuring scheme, and operating systems with guaranteed timely service capabilities, such as the DREAM kernel, will become conspicuous industry trends before year 2000. However, maturity of these ingredients for realization of the NG real-time computing, will be achieved much later than year 2000.

The paper starts in Section 2 with a review of the essence of the RTO.k object structuring scheme. Section 3 then presents in an abridged form the DREAM kernel which is an execution engine model supporting RTO.k objects as well as conventional processes and shared data structure monitors. Section 4 provides an overview of various issues to be resolved via future research in order to firmly establish the RTO.k object based NG real-time computing as a technology usable by common practioners. The paper conclues in Section 5.

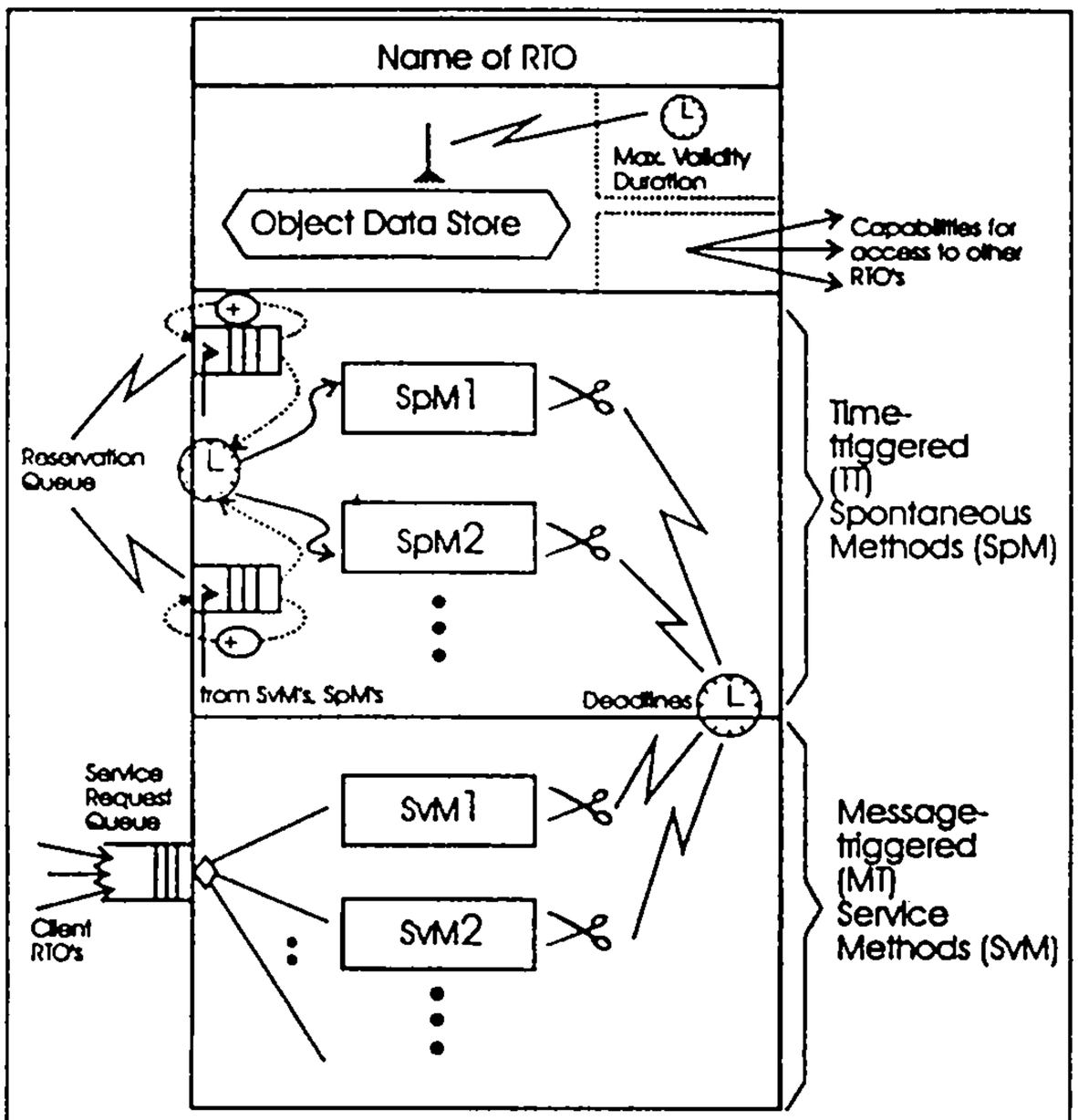


Figure 1. Structure of the RTO.k object model (Adapted from [Kim94b])

2. An overview of the RTO.k object structuring scheme

An initial abstract framework of the *RTO.k object model*, also called the time-triggered real-time object (TT_RTO) model, came out of the attempt by this author and Hermann Kopetz at the Technical University of Vienna to find a proper extension of the basic object model which is highly cost-effective in development of *hard-real-time* application systems. Based on the initial abstract framework formulated in late 1980's [Kop90], a concrete syntactic structure and execution semantics was developed in recent years [Kim94b, Kim94c, Kim94d].

The basic structure of an RTO.k object is depicted in Figure 1.
It is an extension of the conventional object model(s) and two most important and unique extensions are the following:

(a) *Two clearly separated groups of methods:*

For some methods of an RTO.k object, a real-time clock serves as the mechanism for triggering the method executions as the clock reaches some values specified at design time and such methods are called time-triggered (TT-) methods, also called the spontaneous methods (SpM's), and *clearly* separated from the conventional service methods (SvM's) triggered by messages from clients. The two types of methods in an RTO.k object are different not only in the way their executions are triggered but also in that

> "actions to be taken at real times *which can be determined at the design time* can appear only in SpM's".

Therefore, actions of the type "at constant-clock-value do S" or the type "sleep-until constant-clock-value" can appear only in SpM's. Incorporation of SpM's means introducing the potential for the following two new types of concurrent executions of object methods in addition to the potential for concurrent executions of SvM's that exist in conventional objects:

(Type I) Concurrency among SpM executions: This concurrency is specified in an implicit but natural manner, e.g., two SpM's designed to be triggered at 10 am.

(Type II) Concurrency between SpM executions and SvM executions.

(b) *Basic concurrency constraint (BCC):*

In order to dramatically reduce the designer's efforts in guaranteeing timely service capabilities of RTO.k objects, the execution rule which prevents conflicts between SpM's and SvM's is incorporated. Basically, *activation of an SvM triggered by a message from an external client is allowed only when potentially conflicting SpM executions are not in place.* To be exact, when a message-triggered SvM is not free of conflict with a SpM in accessing the same portion of the object data space (ODS), execution of the former method (SvM) must not be allowed in a time zone carmarked for a TT-execution of the latter method (SpM). This restriction is called the basic concurrency constraint (BCC). Therefore, SpM's are given higher priorities for execution over the SvM's. Note that this BCC does not impose any restriction on concurrent execution of SpM's or concurrent execution of SvM's. Therefore, executions of SpM's are not disturbed by SvM executions and triggering times of SpM's are fixed at the design time. At least this makes it very easy to analyze the execution time behavior of SpM's. For example, if a statement of the type "at 10am do S" appears in an SpM, its reliable execution can be easily assured.

The above two features make the RTO.k object model clearly distinguished from other proposed real-time object models [Att91, Bih89, Ish92, Shi91, Tak92]. In addition, the RTO.k object contains the following features not found in the conventional object model(s):

(c) For each execution of a method of an RTO.k object, a deadline is imposed;

(d) Real-time data contained in an RTO.k object become invalid after the interval called the maximum validity duration passes.

Triggering times for SpM's must be fully specified as constants during the design time. Those real-time constants appear in the first clause of an SpM specification called the autonomous activation condition (AAC) section. An example of an AAC is

"for t = from 10am to 10:50am every 30min
start-during (t, t+5min) finish-by t+10min"

which has the same effect as

{"start-during (10am, 10:05am) finish-by start_time+10min",
"start-during (10:30am, 10:35am) finish-by start_time+10min"}.

A provision is also made for making the AAC section of an SpM contain only candidate triggering times, not actual triggering times, so that a subset of the candidate triggering times indicated in the AAC section may be dynamically chosen for actual triggering. Such a dynamic selection occurs when an SvM within the same RTO.k object requests future executions of a specific SpM. The AAC specifying candidate triggering times rather than actual triggering times starts with a declaration "if-demanded". Therefore, there are two different modes of determining triggering times for SpM's:

(a) fully determined during the system design, in which case the SpM is said to be statically scheduled, and

(b) determined during the run time when an SvM requests executions of the SpM and designates a subset of the candidate triggering times prepared during the design time as actual triggering times, in which case the SpM is said to be partially dynamically scheduled.

An underlying design philosophy of the RTO.k object model is that an RTCS will always take the form of a network of RTO.k objects. RTO.k objects interact via calls by client objects for service methods in server objects. The caller may be an SpM or an SvM in the client object. In order to facilitate highly concurrent operations of client and server objects, non-blocking (sometimes called asynchronous) types of calls (i.e., service requests) can be made to SvM's.

The designer of each RTO.k object provides a guarantee of timely service capabilities of the object by indicating the *deadline for every output* produced by each SvM (and each SpM which may be executed on requests from SvM's) in the specification of the SvM (and some relevant SpM's) advertised to the designers of potential client objects. Before determining the deadline specification, the server object designer must convince himself/herself that with the object execution engine (hardware plus operating system) available, the server object can be implemented to always execute the SvM such that the output action is performed within the deadline. Again, the BCC contributes to major reduction of these burdens imposed on the designer.

The RTO.k object model is effective not only in the multiple-level abstraction of real-time (computer) control systems under design but also in the accurate representation and simulation of the application environments. In fact, it enables uniform structuring of control computer systems and application environment simulators [Kim94b] and this presents considerable potential benefits to the system engineers.

3. DREAM kernel: An execution engine model supporting RTO.k objects and conventional processes

An execution engine for RTO.k objects must possess guaranteed timely service capabilities. Otherwise, timely service capabilities of RTO.k objects cannot be guaranteed. Recently an execution engine model called the DREAM (Distributed Real-Time Ever Available Microcomputing) kernel was defined and then its first prototype, called the DREAM kernel v.D2, was implemented to run on a network of PC's connected by Ethernet. It actually supports conventional processes and shared data monitors as well and thus it contains full features of a general-purpose kernel. Therefore, the architectural principles adopted in the DREAM kernel to enable guaranteering timely services to its clients, i.e., processes and RTO.k objects, are broadly applicable to any situations where real-time operating systems are developed. The DREAM kernel rigidly manages execution times of concurrency units and interrupt handlers in order to provide guaranteed timely services to both RTO.k object structured applications and conventional process structured applications.

3.1 The core of the DREAM kernel as a process execution engine

3.1.1 Basic components of process-structured concurrent and distributed programs.

The DREAM kernel as a *process execution engine* supports the following three types of concurrent and distributed program components:

(1) Processes: In this paper, these are also called application processes although in reality, some of these processes may play the roles of system management processes, e.g., processing managing I/O.

(2) CREW (concurrent-read-&-exclusive-write) monitors: The CREW monitor is a shared data structure monitor and an extension of the *monitor* defined in [Bri77] in that the former possesses the *readers-writers* semantics (i.e., concurrent-read-&-exclusive-write semantics) instead of the exclusive-read-&-exclusive-write semantics associated with the latter.

(3) Content-code (CC-) channels in the HU-DF scheme [Kim95]: The *HU-DF (HU data field) scheme* for inter-process-group communication is an extension of the original *data field* scheme developed by Mori and other researchers in Hitachi, Ltd. [Mor86, Mor93]. The essence of the data field scheme is to facilitate dynamic creation of logical multicast channels and dynamic connection of processes to the logical channels in such a way that the idiosyncrasies of the physical communication networks are transparent to the process designer. If the physical communication facility has the broadcast capability, then a logical multicast channel is facilitated by making all processing nodes using the channel to broadcast (through the physical communication facility) messages with the headers containing the ID of the channel called the content code. The processing nodes connected to the logical channel can see all the messages coming through the physical broadcast facility but will pay attention only to those messages containing relevant content codes. The HU-DF scheme differs from the original data field scheme in that the former allows dynamic flexible connection of processes to the logical channels and supports not only conventional *event messages* but also *state messages* which are based on the distributed replicated memory semantics.

These three basic components represent fully general facilities for concurrent and distributed program structuring. Therefore, any operating system kernel that supports these three components along with various I/O operations can be viewed as a general purpose kernel.

3.1.2 Time-leasing machine layering.

The architecture of the DREAM kernel is depicted in Figure 2. As shown, the DREAM kernel adopts a unique approach for layering of its components. This special layering approach is the key to its realization of guaranteed timely service capabilities. The kernel consists of five layers in total and the bottom four of the five layers constitute the DREAM micro-kernel. Whereas the DREAM kernel can be viewed as a process execution engine, the DREAM micro-kernel can be viewed as a *kernel-thread execution engine*.

The kernel-thread, or thread for short, is an active concurrency unit operating inside the DREAM kernel. The set of threads is fixed at the operating system loading time. All the thread share the same address space. Except one called the Main Thread (MT) and responsible for selecting the next process to run at each time for process selection and causing the process to run , all other threads are periodic threads which behave like periodic SpM's in RTO.k objects. Once a thread is chosen to run, it can run for one time unit called thread time-slice. If a periodic thread chosen to run does not need the full thread time-slice, then it can "donate" the remaining portion of the time-slice to MT or just burn it by idling, depending upon the size of the remaining portion. Therefore, the thread scheduling is a simple low-overhead operation unlike the process scheduling.

The special layering depicted in Figure 2 is based on the organizational principle called the time-leasing machine layering, which is an important principle with respect to obtaining kernels with guaranteed timely service capabilities. Under this principle, the bottom layer, L0, owns the full power of the hardware machine. So, L0 uses the hardware machine at its own will. In the

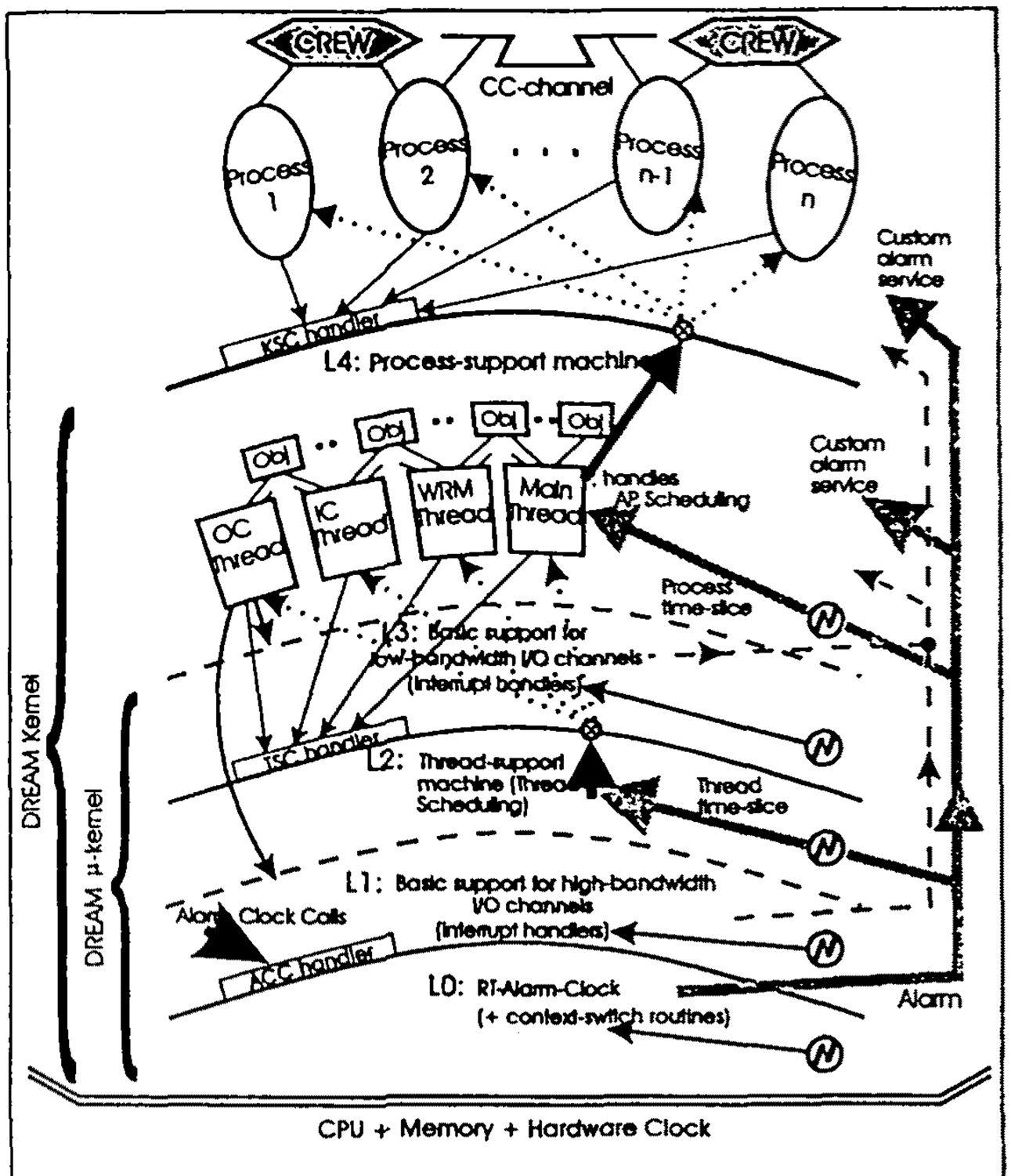


Figure 2. The architecture of the DREAM kernel

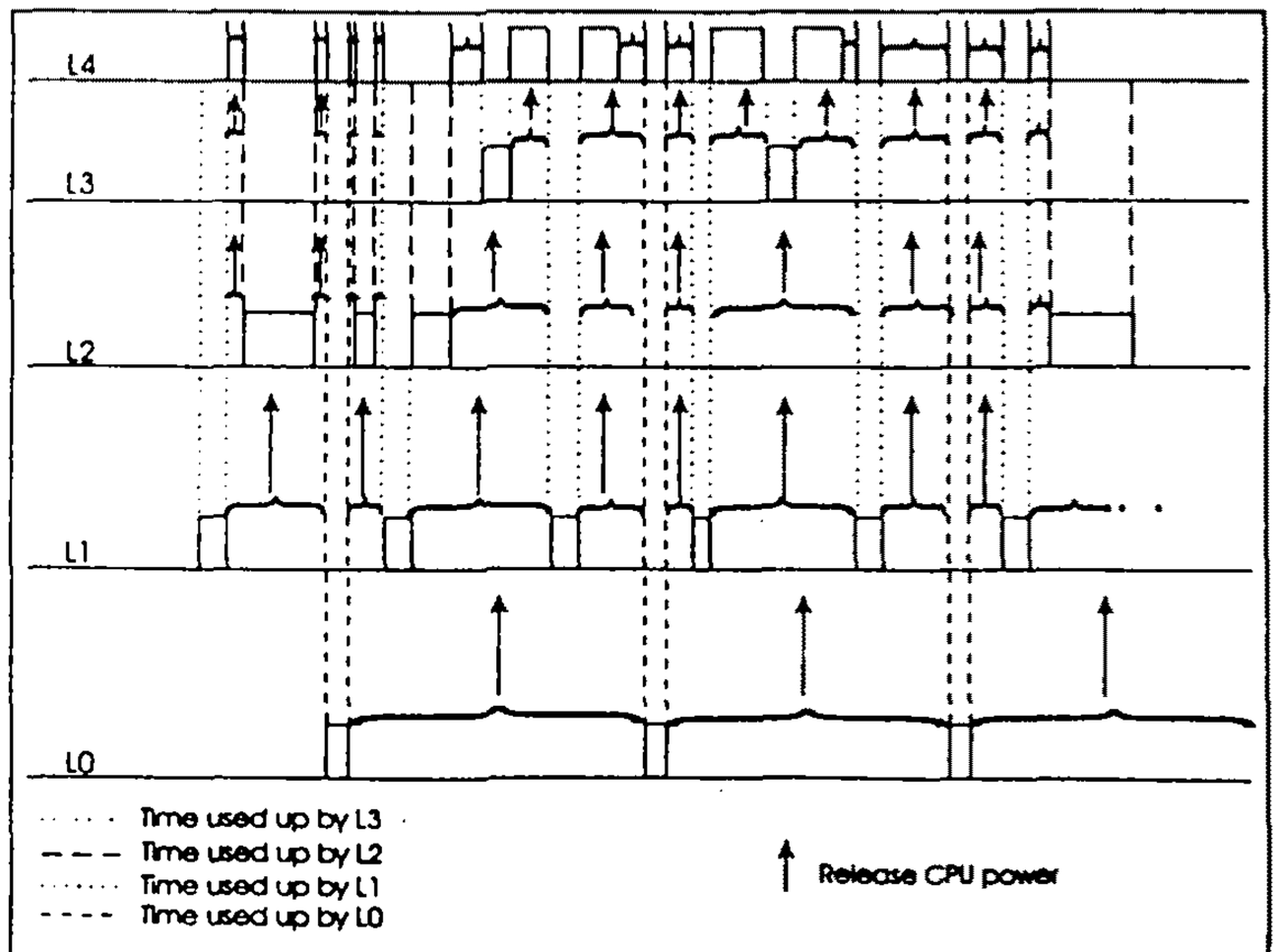


Figure 3. Time-leasing machine layering

current DREAM kernel, L0 contains a manager of a real-time alarm clock which supplies the current time upon receiving a request and also provides the "wakeup call" service. The remainder of the hardware machine time after L0's use of the hardware machine, is "leased" to the next upper layer, L1. L1 then uses a portion of the hardware machine time it receives from L0 (i.e., the machine time which is not used by L0). Similarly, L2 uses a portion of the hardware machine time which is not used by either L0 nor L1. This *recursive time leasing* relationship is applied to up to L4 as depicted in Figure 3.

In the current DREAM kernel, the upper four layers contain the following components:
(1) L1 contains basic support functions for high-band-width I/O such as LAN interface;
(2) L2 contains the thread scheduler which is activated upon expiration of a thread time-slice as well as at some other times;
(3) L3 contains basic support functions for low-band-width I/O such as serial character I/O;
(4) L4 contains the process scheduler and other support functions for processes, CREW monitors, and CC-channels. The process scheduler is activated upon expiration of a process time-slice as well as at some other times.

Another important part of the time-leasing machine layering principle is

> to enforce that the amount of hardware machine time which is used by any layer during a basic response period be limited within a specific threshold.

The *basic response period* here refers to the maximum interval between the instant at which a process calls for a kernel service and the instant at which the service is completed. The amount of machine time used by L0 is quite stable and the upper bound on L0's use during a basic response period can be relatively easily calculated. The upper bound on L2's use or L4's use during a basic response period can also be calculated without much difficulty. However, the cases of L1 and L3 which must respond to interrupts from external sources are different. In principle, the LAN interface manager in L1 can experience bursts of interrupts generated by the LAN interface due to an incoming message burst. We may discover that it becomes impossible to guarantee timeliness of any non-trivial services by L4 to processes under such conditions of L1. If so, the frequency of interrupt generations by L1 must

be controlled and set not to exceed a certain threshold. This means that once the frequency reaches the threshold, any further interrupts by the LAN interface which causes the interrupt frequency bound to be violated are disabled. This in turn means that some messages will simply be lost. That is a price paid for guaranteeing timely delivery of all non-trivial services to processes. This *dynamic control of interrupt sources* is a part of the time-leasing machine layering principle.

3.1.3 Kernel threads.

As mentioned in Section 3.1.2, kernel-threads, except MT, are periodic threads and it leads to low-overhead scheduling of threads. In addition, kernel-threads in the DREAM kernel avoid conflicts among themselves in accessing shared data without using locks for the data. Instead, a thread needing to access a shared data structure orders the thread support machine (in L2) not to disturb the former, i.e., order to disable the thread switch so as not to let the machine power be taken away from the former, until the former notifies the latter that the disturbance prohibition period is over (and this obviously occurs when the thread finishes its access to the shared data structure). So, even if a thread time-slice expiration occurs while the thread is accessing the shared data, the thread support machine does not change the running thread. Such a code-segment by which a thread accesses shared data structures is called a thread-to-thread atomic section (TT-AS). The sending of a no-disturbance request to the thread support machine is called an "Enter-TT-AS" operation. Similarly, the cancellation of a no-disturbance request is called an "Exit-TT-AS" operation.

The TT-AS approach works only if the duration in which the thread executes the AS is much shorter than a thread time-slice. Generally this should be the case. Otherwise, either the thread time-slice was poorly chosen or threads and their shared data structures were not designed properly. This TT-AS approach is more efficient than the conventional data lock based approaches since frequent changes of the running thread can occur under the latter approaches but not under the former approach.

In the basic uniprocessor version of the DREAM kernel, there are four threads: OCT (outgoing communication thread) which manages the sending of messages through the communication network, ICT (incoming communication thread) which manages the distribution of messages coming through the communication network to the destination processes, WRMT (watchdog-&-RTO-management thread) which manages the activation of object methods and checks if there are deadline violations, and MT (main thread). If the DREAM kernel were to operate as a process execution engine only, not as an execution engine for RTO.k objects, then only a part of WRMT, i.e, only the watchdog timer service, is needed. One thing noteworthy here is that special application-specific threads can be introduced if fast response activities specific to the application must be supported.

3.2 Facilities of the DREAM kernel for supporting RTO.k objects

When the DREAM kernel executes RTO.k objects, it actually executes equivalent programs composed of processes, CREW monitors, and data fields (i.e., CC-channels). Such programs composed of the three types of components are called the PCD-programs. Figure 4 depicts the mapping relationship between components of an RTO.k object and the cooresponding components of an equivalent PCD-program.

As shown, object methods, both SpM's and SvM's, are mapped to processes, ODS segments (ODSS's) to CREW monitors, access paths to SvM's to CC-channels, and result return paths to clients to CC-channels. This way of utilizing CC-channels facilitates the transparency of object locations.

Of the kernel-threads shown earlier in Figure 2,

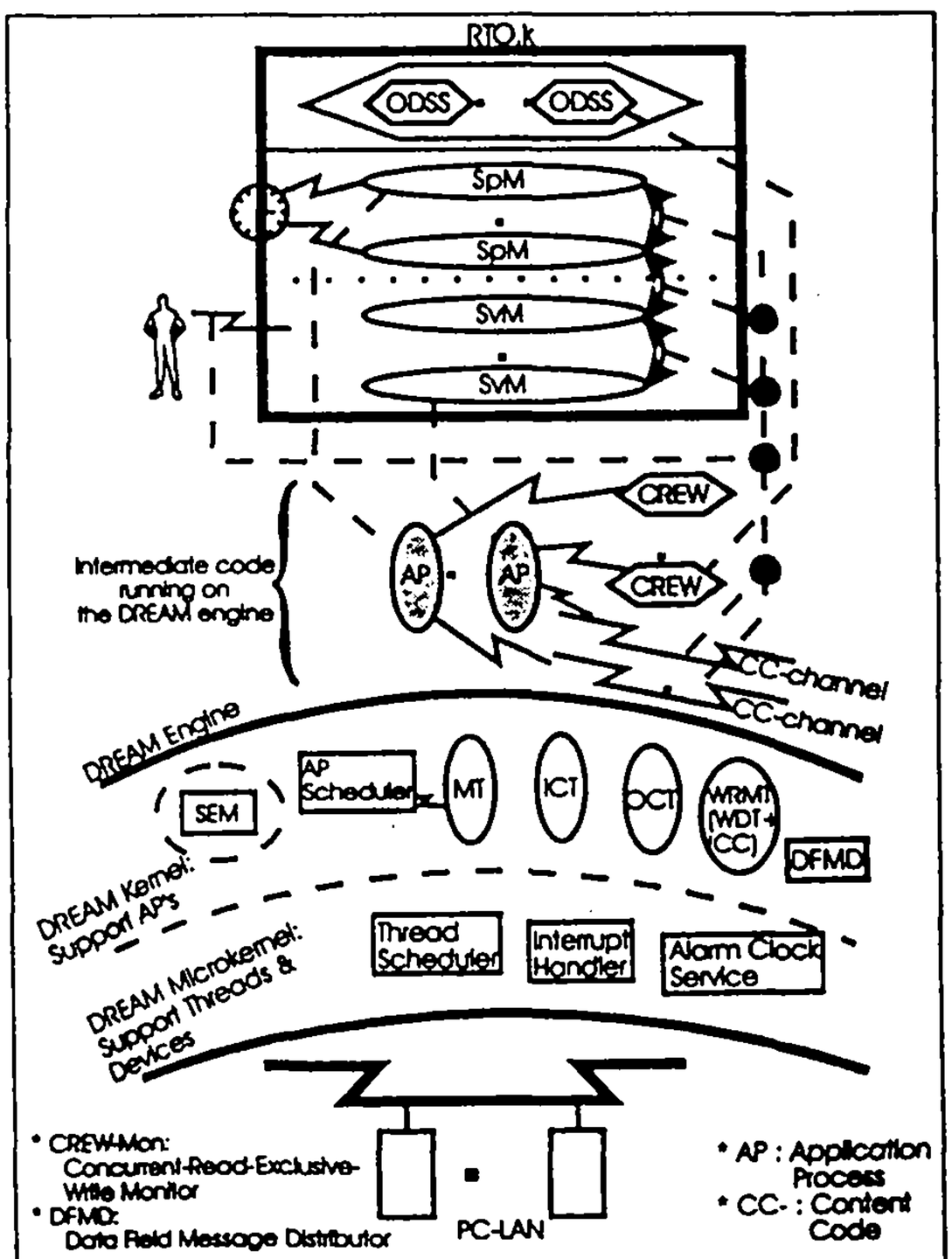


Figure 4. Execution support for RTO.k objects

WRMT provides services needed only to support RTO.k objects. It is responsible for scheduling near-term executions of processes corresponding to SpM's and also activating processes corresponding to SvM's in such manners that activated processes do not run into conflicts with SpM's. Therefore, WRMT utilizes such information as the AAC specification part of every SpM in an RTO.k object, the ODSS access rights of every object method, etc.

3.3 DREAM kernel v.D2: A prototype implementation

The first prototype DREAM kernel, v.D2, was implemented on the PC LAN equipped with Intel 80486 processors, DOS-BIOS device drivers, and the Packet Ethernet driver. Services of the DREAM kernel including process management services can be obtained from within a C++ program (representing an implementation of an RTO.k object) via calls for DREAM library routines. An early version of the prototype DREAM kernel was used in an experimental development of a defense system with its application environment simulator [Kim94b] and has since been replaced by the current version. In this experimental effort, translation of the RTO.k object structured program into an equivalent PCD-program was done manually. We plan to implement in the near future another version of the DREAM kernel which uses the IBM micro-kernel as its component.

4. Remaining issues in establishing the RTO.k object based approach to NG real-time computing

4.1 Operating system support

At present, the DREAM kernel is little more than a minimal-functionality model of a timeliness-guaranteed operating system kernel supporting RTO.k objects. It can be extended in several major directions. The most obvious among them is to extend it to utilize multi-processor architectures and highly parallel machine architectures. Adapting the DREAM kernel to commercial micro-kernel environments is another meaningful direction to pursue. Without investment by industry of sizable efforts to revise and extend their existing operating system products to support RTO.k objects, execution engines usable by common practioners will not be available anytime soon.

4.2 Programming language tools

An extension of C++ to support RTO.k object programming is the most natural path to follow in this area. The author and his research associates are currently developing one such extension named C++T. Our approach will be to convert C++T programs into PCD-programs first and then to convert PCD-programs into machine programs by use of commercial C++ compilers. Many other meaningful research issues, e.g., incorporating the RTO.k object structuring scheme into languages such as Ada, remain in this area.

4.3 Software engineering tools

The most urgently needed among various software engineering tools desired are those for assisting the RTO.k object designer in the process of determining the response time to be guaranteed. Such tools must be capable of making good estimates of worst-case execution times for various segments of object methods.

The capability of the RTO.k object structuring scheme for supporting uniform structuring of control computer systems and application environment simulators suggests various potential improvements in different parts of the real-time software engineering, ranging from the requirements specification part to the testing and maintenance parts. Much future research is need to check such potentials.

4.4 Fault tolerance

A promising and widely applicable approach to realizing real-time fault tolerance in RTO.k object structured DCS's is to replicate each important RTO.k object into a primary-shadow pair of self-checking RTO.k objects based on the operational principles of the *distributed recovery block* (DRB) scheme [Kim89, Hec91, Kim94a]. Plausible outlines of RTO.k object-pairs are provided in this section but their validation as well as development of efficient implementation techniques remain as important subjects for future research. For the sake of step-by-step exposition of technical issues and promising approaches, three different cases of objects are discussed in the order of increasing complexity. Operational principles are briefly reviewed first.

4.4.1 Operational principles of the distributed recovery block (DRB) scheme

The DRB scheme is based on a combination of both the concurrent processing and the recovery block structuring approach. Recovery block [Ran75, Ran94] is a language construct supporting the incorpora-tion of multiple versions of an application task procedure, called *try blocks* and designed to produce the same or similar

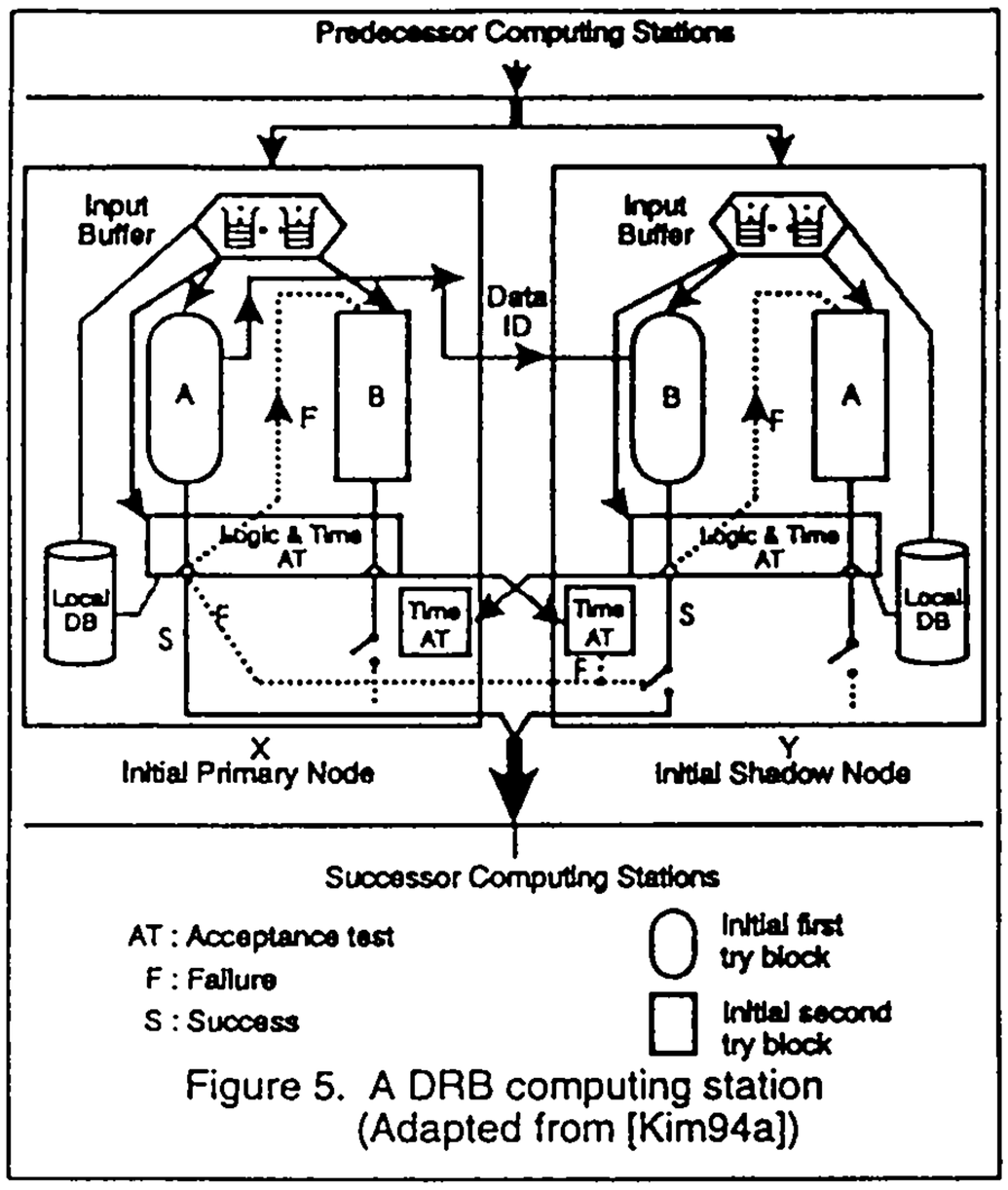


Figure 5. A DRB computing station (Adapted from [Kim94a])

computational results, together with an *acceptance test* routine. The acceptance test is a logical expression representing the criterion for determining the acceptability of the execution results of the try blocks. A *try* (i.e., execution of a try block) is thus always followed by an acceptance test.

The DRB scheme exploits concurrent execution of try blocks to facilitate fast forward recovery. Although more than two try blocks can be utilized under the DRB scheme [Kim94a], only the cases of using one or two try blocks in each recovery block are considered in most applications. In fact, if only one try block is used, then the DRB scheme is reduced to its core component, the pair of self-checking processing nodes (PSP) scheme, which is aimed primarily for tolerance of hardware faults. The specification of the maximum execution time allowed for any of the try blocks in each recovery block is an integral part of the DRB scheme. A try not completed within the time due to hardware faults or excessive looping is treated as a failure. Therefore, the acceptance test can be viewed as a combination of both logic and time acceptance tests.

The DRB computing station realized with two nodes is depicted in Figure 5. The recovery block is duplicated into both nodes and thus both primary and shadow nodes contain the same acceptance test and the same set of try blocks, A and B. However, the roles of the two try blocks are assigned differently in the two nodes. Primary node X uses try block A as the first try block initially, whereas shadow node Y uses try block B as the initial first try block. Therefore, until a fault is detected, both nodes receive the same input data, process the data by use of two different try blocks (i.e., block A on node X and block B on node Y), and check the results by use of the acceptance test. Both nodes perform all these tasks concurrently. The time acceptance test is used to ensure timely behavior of both nodes.

In a fault-free situation, both nodes will pass the acceptance test with the results computed with their first try blocks. In such a case, the primary node notifies the shadow of its success in the acceptance test. Thereafter, only the primary node sends its output to the successor computing station. However, if the primary node fails and the shadow node passes its test, the shadow node assumes the role of the primary, i.e., the nodes exchange their roles. To be more specific, the primary node attempts to inform the shadow node upon its failure in passing the acceptance test. The shadow node will take over the role of the primary as soon as it receives the notice. If the primary node crashes completely, the shadow node will recognize the failure of the primary upon expiration of the preset time limit. It will then become the new primary. On the other hand, if the shadow node fails first, the primary node need not be disturbed. In both cases, the failed node attempts to become an operational shadow node; it attempts to roll back and retry with its alternate try block to bring its application computation state or local database up-to-date. This attempt does not disturb the primary node.

4.4.2 Case I: Self-contained server object

Figure 6 depicts a primary-shadow pair of self-contained server RTO.k objects. A self-contained server object is an object capable of serving clients without depending on other objects. The *primary object* and the *shadow object* in Figure 6 are designed to provide functionally equivalent, *not necessarily identical*, services to the clients. Every computational output of each object is filtered through an acceptance test, following the principle of the DRB scheme. When the primary object passes an acceptance test, it sends an "I'm OK" signal to the shadow object and then executes an output action. Upon receiving this notice, the shadow object skips the corres-ponding output step.

Each SvM in the primary object, the corresponding SvM in the shadow object, and the acceptance test, can be structured as a recovery block. The designer must ensure that the initiation order of object methods is always the same in both partner objects. Further research is needed to discover how difficult or complicated this can be.

4.4.3 Case II: Self-contained object group with single interface object

Figure 7 depicts a primary-shadow pair of self-contained RTO.k object groups, each containing single

interface object. An interface object in an object group is an object interfacing with the outside of the object group.

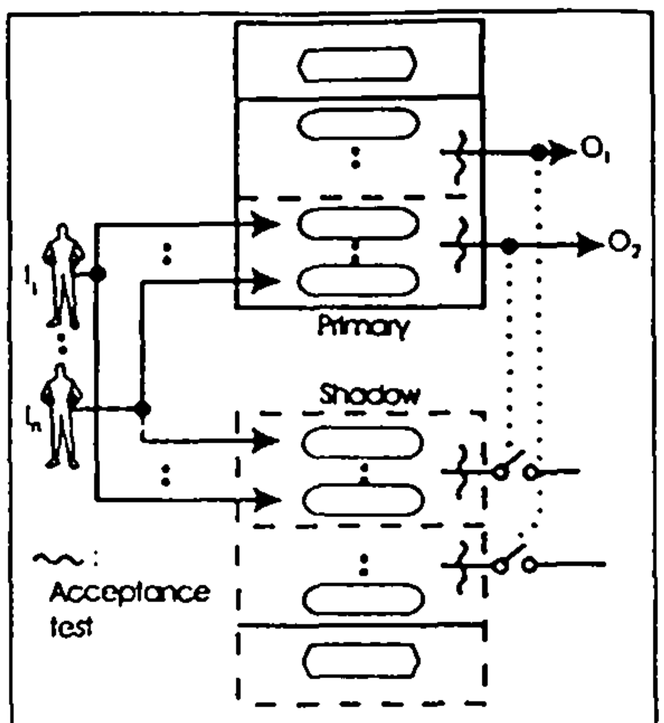


Figure 6. Primary-shadow pair of self-contained server RTO.k objects.

So, the interface object in an object group in Figure 7 is the only object in the group which directly receives service requests from the clients which are external to the object group, and takes output actions impacting the outside of the object group. Other objects in an object group receive service requests from other objects, including the interface object, in the group and their computational results directly impact only other objects in the group. The roles of the acceptance tests in Figure 7 are similar to those in Figure 6.

The recovery block structure can be used in an interface object in the same way as in the case depicted in Figure 6 if service calls to other objects in the object group are treated as internal, non-IO, operations. On the other hand, the number of non-interface objects needs not be the same in both primary and shadow object groups. So, there is room for using different newly defined language constructs which are somewhat closer to the conversation structure [Ran75, Kim82, Ran94] than to the recovery block. In addition to the research issues encountered in the case depicted in Figure 6, other new issues must be dealt with in future research. For example, if a processing node hosting a non-interface object crashes due to an error without a hardware fault, then how can that object or the containing object group recover to the point where the object group can start acting as a new shadow object group ?

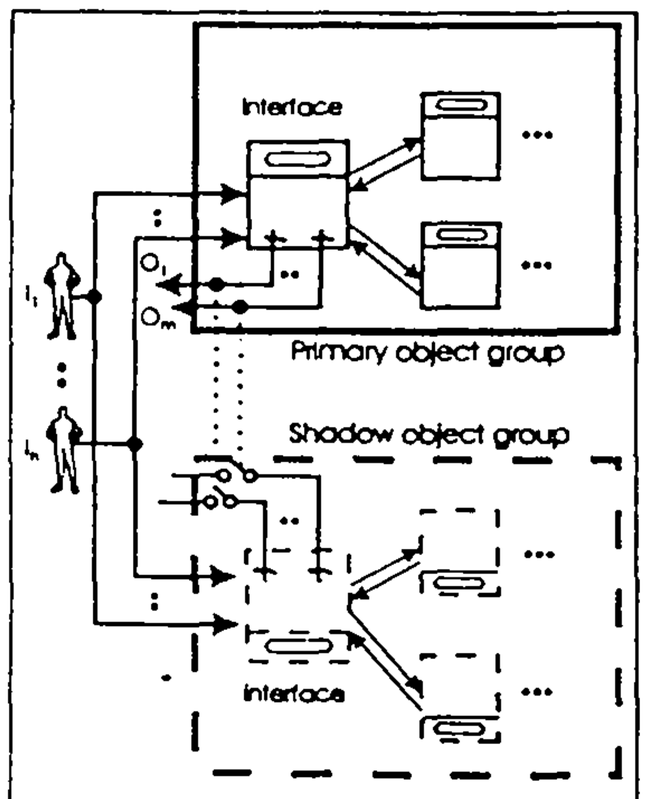


Figure 7. Primary-shadow pair of RTO.k object groups with single interface object in each group.

4.4.4 Case III: Self-contained object group with multiple interface objects

The pair of object groups depicted in Figure 8 differs from the pair in Figure 7 in that each object group contains multiple interface objects.

So, this is the most general case. This primary-shadow pair of object groups must operate on the basis of the principle of distributed parallel execution of a conversation rather than on the basis of the operating principle of the DRB scheme. In addition to the research issues raised in connection with the case depicted in Figure 7, questions such as how a *recovery line* and a *test line* [Kim82] can be designed must be dealt with in future research.

5. Conclusion

The core of the the *new-generation (NG) real-time computing* approach advocated in this paper and also in earlier publications by the author, is to realize real-time computing in a general manner not alienating the main-stream computing industry and yet enabling the system engineer to confidently produce certifiable RTCS's for safety-critical applications. This author believes that time is ripe for vigorously pursuing this idealistic approach. The specific research issues addressed in this paper as remaining issues are certainly just a small part, though probably the most urgent part, of the issues that should be adequately resolved to make the NG real-time computing

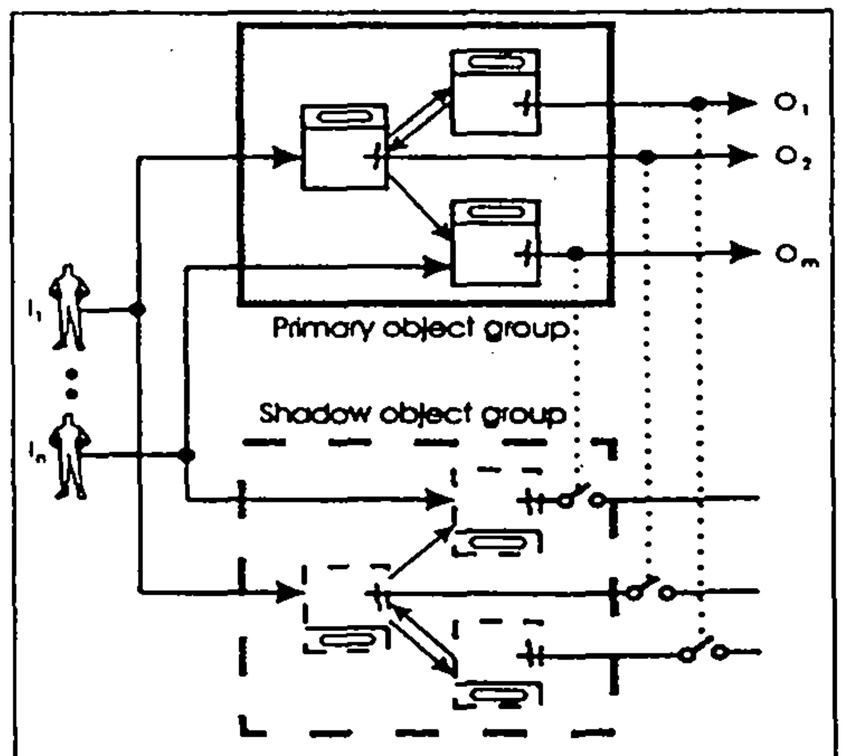


Figure 8 Primary-shadow pair of RTO.k object groups with multiple interface objects in each group.

a common practice.

Acknowledgment: The research work reported here was supported in part by US Navy, NSWC Dahlgren Division under Contract No. N60921-92-C-0204, in part by the California Transportation Department via the UCI Institute for Transportation Studies, and in part by Hitachi, Ltd.

References

[Att91] Attoui, A. and Schneider, M., "An Object Oriented Model for Parallel and Reactive Systems", *Proc. IEEE CS 12th Real-Time Systems Symp.*, 1991, pp. 84-93.

[Bih89] Bihari, T., Gopinath, P., and Schwan, K., "Object-Oriented Design of Real-Time Software", *Proc. IEEE CS 10th Real-Time Systems Symp.*, 1989, pp.194-201.

[Bri77] Brinch Hansen, P., "The Architecture of Concurrent Programs," Prentice-Hall, 1977.

[Hec91] Hecht, M., et al., "A Distributed Fault Tolerant Architecture for Nuclear Reactor and Other Critical Process Control Applications.", *Proc. IEEE Computer Society's 21st Int'l Symp. on Fault-Tolerant Computing*, June 1991, Montreal, pp.462-469.

[Ish92] Ishikawa, Y., Tokuda, H., and Mercer, C. W., "An Object-Oriented Real-Time Programming Language", *IEEE Computer*, October 1992, pp. 66-73.

[Kim82] Kim, K.H., "Approaches to Mechanization of the Conversation Scheme Based on Monitor.", *IEEE Trans. on Software Eng.*, Vol. SE-8, No. 3, May 1982, pp.189-197.

[Kim89] Kim, K.H. and Welch, H.O., "Distributed Execution of Recovery Blocks: An Approach to Uniform Treatment of Hardware and Software Faults in Real-Time Applications.", *IEEE Trans. Computers*, May 1989, pp.626-636.

[Kim94a] Kim, K.H., "The Distributed Recovery Block Scheme", Ch. 8 in Michael Lyu ed., '*Software Fault Tolerance*', John Wiley & Sons, 1994, pp.189-209.

[Kim94b] Kim, K.H. and Kopetz, H., "A Real-Time Object Model RTO.k and an Experimental Investigation of Its Potentials", *Proc. 1994 IEEE CS Computer Software and Applications Conf. (COMPSAC)*, Nov. 1994, Taipei, pp.392-402.

[Kim94c] Kim, K.H., "A Utopian View of Future Object-Oriented Real-Time Dependable Computer Systems", (Invited paper) *Proc. 1st Int'l Workshop on Real-Time Computing Systems and Applications*, Seoul, Korea, Dec. 1994, pp. 59-69.

[Kim94d] Kim, K.H. et al., "Distinguishing Features and Potential Roles of the RTO.k Object Model", to appear in *Proc. 1994 IEEE CS Workshop on Object-oriented Real-time Dependable Systems (WORDS)*, Oct. 1994, Dana Point, (Proceedings to be published in June, 1995, A draft version in the preliminary workshop proceedings).

[Kim95] Kim, K.H., Mori, K., and Nakanishi, H., "Realization of Autonomous Decentralized Computing with the RTO.k Object Structuring Scheme and the HU-DF Inter-Process-Group Communication Scheme", to appear in *Proc. 1995 IEEE CS Int'l Symp. on Autonomous Decentralized Systems (ISADS)*, April 1995, Phoenix.

[Kop90] Kopetz, H. and Kim, K.H., "Temporal Uncertainties in Interactions among Real-Time Objects", *Proc. IEEE Computer Society's 9th Symp. on Reliable Distributed Systems*, Huntsville, AL, Oct. 1990, pp.165-174.

[Mor93] Mori, K., "Autonomous Decentralized Systems: Concept, Data Field Architecture, and Future Trends", *Proc. IEEE CS Int'l Symp. on Autonomous Decentralized Systems (ISADS 93)*, Mar. 1993, Kawasaki, Japan, pp. 28-34.

[Mor86] Mori, K., et. al., "Autonomous Decentralized Software Structure and Its Application.", Proc. Fall Joint Computer Conference, Dallas, Texas, Nonember 1986, pp. 1056-1063.

[Ran75] Randell, B., "System Structure for Software Fault Tolerance.", *IEEE Transactions on Software Engineering*, June 1975, pp.220-232.

[Ran94] Randell, B. and Xu, Jie, "Recovery Blocks", Ch. 1 in Michael Lyu ed., '*Software Fault Tolerance*', John Wiley & Sons, 1994, pp.1-21.

[Shi91] Shrivastava, S.K. and Waterworth, A., "Using Objects and Actions to provide Fault Tolerance in Distributed, Real-Time Applications", *Proc. IEEE CS 12th Real-Time Systems Symp.*, 1991, pp.276-285.

[Tak92] Takashio, K., and Tokoro, M., "DROL: An Object-Oriented Programming Language for Distributed Real-Time Systems", *Proc. OOPSLA*, 1992, pp. 276-294.

Proceedings of the

Fifth
IEEE Computer Society Workshop on
Future Trends of Distributed Computing Systems

August 28-30, 1995 *Cheju Island, Korea*

Sponsored by

The IEEE Computer Society Technical Committee on Distributed Processing

In cooperation with

IFIP WG 10.4 on Dependable Computing
Korea Information Science Society (KISS)
Electronics and Telecommunications Research Institute (ETRI), Korea
Korea Research Foundation
Samsung Data Systems

IEEE Computer Society Press
Los Alamitos, California

Washington • Brussels • Tokyo

Appendix C.

[Kim95c] Kim, K.H. et al.,
"A Timeliness-Guaranteed Kernel Model -- DREAM Kernel -- and Implementation Techniques ",
To appear in Proc. RTCSA '95 (1995 Int'l Workshop on Real-Time Computing Systems & Applications), Tokyo, Oct. '95.

To appear in Proc. RTCSA '95 (1995 Int'l Workshop on Real-Time Computing Systems & Applications), Tokyo, Oct. '95.

A Timeliness-Guaranteed Kernel Model - DREAM Kernel - and Implementation Techniques

K. H. (Kane) Kim [*]
University of California, Irvine, U.S.A.

Luiz Bacellar [*] Yuseok Kim [*] Chittur Subbaraman [*] Hankil Yoon [*]
Jungguk Kim
HUFS, Korea
and
Kee-Wook Rim
ETRI, Korea

ABSTRACT An essential building-block for construction of future real-time computer systems (RTCS's) is a *timeliness-guaranteed operating system*. The first co-author recently formulated a model of an operating system kernel which can support both real-time processes and new-style real-time objects with guaranteed timely services. The model has been named the DREAM kernel. The key emphasis in formulating the DREAM kernel was in realization of guaranteed timely service capabilities with minimal loss of hardware utilization. This paper presents a summary of the main structuring principles that were exploited to realize guaranteed timely service capabilities together with modularity and expandability in the DREAM kernel. A prototype implementation of the DREAM kernel, v.D2, has been produced by the authors to run on a network of PC's connected by an Ethernet. Several implementation techniques that were adopted during the course of this prototype implementation and may be applicable to other real-time kernel development environments, are briefly discussed in this paper. The prototype kernel (v.D2) has been used to run a real-time object structured non-trivial defense C3 application together with a real-time simulator of the application environment.

1. Introduction

A few years ago the first co-author started subscribing to the view that it was time to start vigorously pursuing the following real-time computing paradigms [Kim94c, Kim95b]

(1) *General-form design style*. Future real-time computing must be realized in the form of a generalization of the non-real-time computing, rather than in a form looking like an esoteric specialization.

(2) *Design-time guarantee of timely service capabilities of subsystems*. To meet the demands of the general public on the assured reliability of future RTCS's in safety-critical applications, there does not appear to be any adequate way but to require the system engineer to produce design-time guarantees for timely service capabilities of various subsystems (which will take the form of objects in object-oriented system designs).

The motivating factors behind these paradigms which may be called the new-generation (NG) real-time computing paradigms are the newly improved hardware economy and component reliability which provide impetus in expanding the real-time computing application field.

An essential building-block for construction of NG real-time computer systems (RTCS's) is a *timeliness-guaranteed operating system* which together with the hardware platform forms an *execution engine* that provides guaranteed timely services to concurrent and distributed real-time application software. If the operating system lacks such guaranteed timely service capabilities, then timely service capabilities of real-time applications cannot be ensured. Existing commercial operating systems lack the capability for either supporting a general-form design style or providing guaranteed timely services to application software.

The first co-author recently formulated a model of an operating system kernel which can support real-time processes with guaranteed timely services. The model, named the DREAM (Distributed Real-time Ever Available Micro-computing) kernel, actually came out of an attempt to develop an execution engine that supports a new real-time object-oriented (OO-) structuring approach. The new object structuring approach is called the RTO.k object structuring scheme [Kim94a, Kim94b, Kim94c] and intended to facilitate the NG real-time computing. To realize the idealistic NG real-time computing, a powerful structuring scheme capable of dealing with all practically useful real-time and non-real-time computing requirements must be established. We believe that such structuring methods should preferably be of object-oriented (OO-) type, considering the modularity, generality, and natural abstraction benefits that object-oriented approaches bring in. In the last several years, there has been a growing trend of research activities aimed for extending the conventional OO-structuring approaches to support RTCS design [Att91, Ish92, Kim94b, Kop90, Tak92]. However, most of those works have not been aimed for supporting the design-time guarantee of timely service capabilities of objects, which is one of the fundamental requirements of the NG real-time computing. Therefore, the RTO.k object structuring scheme, though an extension of the

---

[*] These co-authors are staff members of the DREAM Laboratory, ECE Dept., Univ. of California, Irvine (UCI), USA, directed by the first co-author.

conventional OO-structuring approaches, has several unique features of fundamental nature which will be reviewed later in Section 3.1.

As an execution for RTO.k objects, the DREAM kernel was designed to enable flexible linking between RTO.k objects and various hardware structures. The kernel enables this by supporting
(1) real-time processes with various activation and synchronization requirements,
(2) shared data structure monitors, and
(3) real-time multicast logical (RML-) channels,
which in turn execute the components of RTO.k objects. Therefore, the DREAM kernel can support both process-structured real-time application software and RTO.k object structured application software. The key emphasis in formulating the DREAM kernel was in realization of guaranteed timely service capabilities with minimal loss of hardware utilization. The DREAM kernel can thus be viewed as a model of *a general-purpose timeliness-guaranteed OS kernel*.

One of the two main purposes of this paper is to provide an overview of the main structuring principles that were exploited to realize guaranteed timely service capabilities together with modularity and expandability in the DREAM kernel. Among the main principles exploited is that of structuring multiple layers of "time-leasing" machines, each of which guarantees a certain amount of machine time being available to upper-layer machines. The DREAM kernel has a five-layer structure and the lower four of the five layers form the DREAM micro-kernel.

A prototype implementation of the DREAM kernel, v.D2, has been produced by the authors to run on a network of PC's connected by an Ethernet. Presenting some major implementation techniques adopted is the other main purpose of this paper. The prototype kernel (v.D2) has been used to run an RTO.k structured non-trivial defense C3 application, together with a real-time simulator of the application environment.

The paper starts in Section 2 with a discussion on the core of the DREAM kernel that is essentially a timeliness-guaranteed process execution engine. The entire DREAM kernel is then discussed in Section 3 and this section begins with a review of the essence of the RTO.k object structuring scheme. The remainder of the section focuses mainly on the part the DREAM kernel that is directly related to supporting RTO.k objects. Section 4 discusses the prototype implementation, v.D2, and some implementation techniques adopted are presented. The paper concludes in Section 5 with discussions on the issues to be resolved via future research.

2. DREAM kernel as a process execution engine

The DREAM supports both process-structured real-time application software and RTO.k object structured application software. This section focuses on the core of the DREAM kernel that is a timeliness-guaranteed process execution engine. The main structuring principles that were exploited to realize guaranteed timely service capabilities together with modularity and expandability are reviewed.

2.1 Basic components of process-structured real-time concurrent and distributed programs

The DREAM kernel as a *process execution engine* supports the following three types of concurrent and distributed program components.

(1) Processes. In this paper, these are also called *application processes* (AP's) although in reality, some of these processes may play the roles of system management processes, e.g., processing managing I/O. The processes may frequently impose deadlines on the kernel for execution of their next computation-segments.

(2) CREW (concurrent-read-&-exclusive-write) monitors. The CREW monitor is a shared data structure monitor and an extension of the *monitor* defined in [Bri77] in that the former possesses the *readers-writers* semantics (i.e., *concurrent-read-&-exclusive-write* semantics) instead of the exclusive-read-&-exclusive-write semantics associated with the latter.

(3) Content-code (CC-) channels in the HU-DF scheme [Kim95a]. The *HU-DF (HU data field) scheme* for inter-process-group communication is an extension of the original *data field* scheme developed by Mori and other researchers in Hitachi, Ltd. [Mor93]. The essence of the data field scheme is to facilitate dynamic creation of multicast logical channels and dynamic connection of processes to the logical channels in such a way that the idiosyncrasies of the physical communication networks are transparent to the process designer. If the physical communication facility has the broadcast capability, then a logical multicast channel is facilitated by making all processing nodes using the channel to broadcast (through the physical communication facility) messages with the headers containing the ID of the channel called the content code (CC). The processing nodes connected to the logical channel can see all the messages coming through the physical broadcast facility but will pay attention only to those messages containing relevant content codes. The HU-DF scheme differs from the original data field scheme in that the former allows dynamic flexible connection of processes to the logical channels and supports not only conventional *event messages* but also *state messages* which are based on the distributed replicated memory semantics (a variation of the state message semantics in [Kop89])

These three basic components represent fully general facilities for concurrent and distributed program structuring. Therefore, any operating system kernel that supports these three components along with various I/O operations can be viewed as a general purpose kernel. Programs composed of the three types of components are called the PCD (process-CREW-DF) programs.

2.2 Five-layer structure and the principle of time-leasing machine layering

The architecture of the DREAM kernel is depicted in Figure 1.

As shown, the DREAM kernel adopts a unique approach for layering of its components. This special layering approach is the key to its realization of guaranteed timely service capabilities. The kernel consists of five layers in total and the bottom four of the five layers constitute the DREAM micro-kernel. Whereas the DREAM kernel can be viewed as a *process execution engine*, the DREAM micro-kernel can be viewed as a *kernel-thread execution engine*.

The kernel-thread, or thread for short, is an active concurrency unit operating inside the DREAM kernel. The set of threads is fixed at the operating system loading time. All the threads share the same address space. Except one called the *Main Thread (MT)* and responsible for selecting the next process to run and causing the process to run within the time periods allocated to itself (MT), all other threads are periodic threads (which behave like periodic time-triggered methods in RTOK objects reviewed in Section 3.1). Once a thread is chosen to run, it can run for one time unit called *thread time-slice*. If a periodic thread chosen to run does not need the full thread time-slice, then it can "donate" the remaining portion of the time-slice to MT or just burn it by idling, depending upon the size of the remaining portion. Therefore, the thread scheduling is a simple low-overhead operation unlike the process scheduling. We believe that restricting all kernel threads except MT to be of this type only, i.e., periodic threads using one thread time-slice at a time, is a well justified approach for making the analysis of the worst-case response time of each thread to be simple without much sacrificing the hardware utilization.

The special layering depicted in Figure 1 is based on the organizational principle called the time-leasing machine layering, which is an important principle with respect to obtaining kernels with guaranteed timely service capabilities. Under this principle, the bottom layer, L0, owns the full power of the hardware machine. So, L0 uses the hardware machine at its own will. L0 contains a manager of a real-time alarm clock which supplies the current time upon receiving a request and also provides the "wake-up call" service. The remainder of the hardware machine time after L0's use of the hardware machine, is "leased" to the next upper layer, L1. L1 then uses a portion of the hardware machine time it receives from L0 (i.e., the machine time which is not used by L0). Similarly, L2 uses a portion of the hardware machine time which is not used by either L0 nor L1. This *recursive time leasing* relationship is applied to up to L4.

The upper four layers in the DREAM kernel contain the following components.
(1) L1 contains basic support functions for high-bandwidth I/O such as LAN interface.

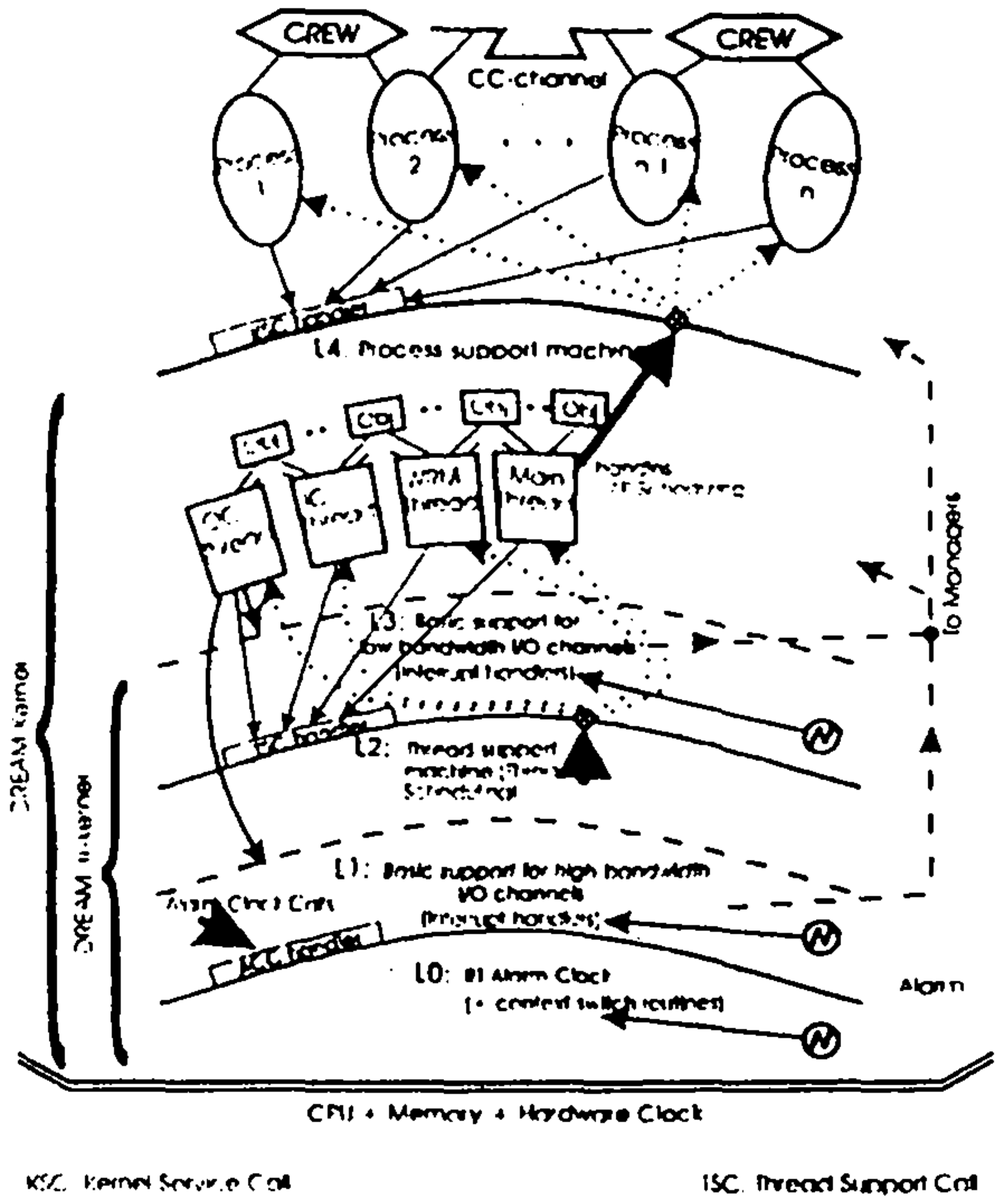


Figure 1. The architecture of the DREAM kernel

(2) L2 contains the thread scheduler which is activated upon expiration of a thread time-slice as well as at some other times.
(3) L3 contains basic support functions for low-bandwidth I/O such as serial character I/O.
(4) L4 contains the process scheduler and other support functions for processes, CREW monitors, and CC-channels. The process scheduler is activated upon expiration of a *process time-slice* as well as at some other times. Here the size of the process time-slice is usually different from the size of the thread time-slice.

Another important part of the time-leasing machine layering principle is

> *to enforce that the amount of hardware machine time which is used by any layer during a basic response period be limited within a specific threshold.*

The *basic response period* here refers to the maximum interval between the instant at which a process calls for a kernel service and the instant at which the service is completed. The obvious purpose of adopting this principle is to guarantee a certain amount of machine time being available to each of the upper layers.

The amount of machine time used by L0 is quite stable and the upper bound on L0's use during a basic response period can be relatively easily calculated. The upper bound on L2's use or L4's use during a basic response period can also be calculated without much difficulty. However, the cases of L1 and L3 which must respond to interrupts from external sources are different.

In principle, the LAN interface manager in L1 can experience bursts of interrupts generated by the LAN interface due to an incoming message burst. We may discover that it becomes impossible to guarantee timely delivery of any non-trivial services by L4 to processes under such conditions of L1. If so, *the frequency of interrupt generations by L1 must be controlled and set not to exceed a certain threshold*. This means that once the frequency reaches the threshold, any further interrupts by the LAN interface which causes the interrupt frequency bound to be violated are disabled. This in turn means that some messages will simply be lost. That is a price paid for guaranteeing timely delivery of all non-trivial services to processes. We believe that this *dynamic control of interrupt sources*, which is a part of the time-leasing machine layering principle, is a requirement that is inevitable in any timeliness-guaranteed operating system.

2.3 Kernel-threads

As mentioned in the preceding subsection (2.2), kernel-threads, except MT, are periodic threads using one thread time-slice at a time. In the basic uniprocessor version of the DREAM kernel, there are four essential kernel-threads:
(1) OCT (outgoing communication thread) which manages the sending of messages through the communication network,
(2) ICT (incoming communication thread) which manages the distribution of messages coming through the communication network to the destination processes,
(3) WRMT (watchdog-&-RTO-management thread) which manages the activation of object methods and checks if there are deadline violations, and
(4) MT (main thread).
If the DREAM kernel were to operate as a process execution engine only, not as an execution engine for RTO.k objects, then only a part of WRMT, i.e., only the watchdog timer service, is needed.

Data input from a high-bandwidth device is realized through a combined effort of the device, the interrupt handler in L1, and the ICT in L4. There is usually a modest-size buffer, say B1, into which a high-bandwidth device can pour data until it is filled. The device generates an interrupt when the buffer B1 is filled. In response to the interrupt, the interrupt handler in L1 moves the data in B1 into another larger buffer B2 accessible to both the interrupt handler and the ICT. The interrupt handler may insert many data sets into B2 before the ICT becomes ready to access B2. Later when the ICT gets the next thread time-slice, it moves the data in B2 to the destination, an AP.

One thing noteworthy here is that special application-specific threads can be introduced if fast response activities specific to the application must be supported. However, the set of all kernel threads must become fixed at the operating system loading time in order not to damage the ability to guarantee timely service capabilities of the kernel at a reasonable performance level.

2.4 Thread-to-thread atomic sections (TT-AS's)

Restricting kernel-threads, except MT, to be periodic threads using one thread time-slice at a time enables low-overhead scheduling of threads and easy analysis of such overhead. An additional measure taken to simplify the worst-case response time of a kernel thread is to restrict kernel-threads in the DREAM kernel to avoid conflicts among themselves in accessing shared data without using locks for the data. Instead, a thread needing to access a shared data structure orders the thread support machine (in L2) not to disturb the former, i.e., orders L2 to disable the thread switch so as not to let the machine power be taken away from the thread, until the thread notifies L2 that the disturbance prohibition period is over (and this obviously occurs when the thread finishes its access to the shared data structure). So, *even if a thread time-slice expiration occurs during the disturbance period (i.e., while the thread is accessing the shared data), the thread support machine does not change the running thread.* Such a code-segment during the execution of which a thread accesses shared data structures is called a thread-to-thread atomic section (TT-AS). The sending of a no-disturbance request to the thread support machine is called an "Enter-TT-AS" operation. Similarly, the cancellation of a no-disturbance request is called an "Exit-TT-AS" operation.

The TT-AS approach works only if the duration in which the thread executes the atomic section (AS) is much shorter than a thread time-slice. Generally this should be the case. Otherwise, either the thread time-slice was poorly chosen or threads and their shared data structures were not designed properly. This TT-AS approach is more efficient than the conventional data lock based approaches since kernel threads except MT are periodic threads and frequent changes of the running thread can occur under the latter approaches but not under the former approach.

2.5 Two-level scheduling

Since there are two types of concurrent computation-units, i.e., AP's and kernel threads, which the DREAM kernel deals with, scheduling of concurrent computation-units occurs in two different layers of the kernel. *The process time-slice should never be smaller than the thread time-slice.* Normally, the process time-slice should be multiple times as long as the thread time-slice is.

In L2 (the thread-support machine), the thread scheduler schedules the four threads, ICT, OCT, WRMT, and MT, for use of the hardware machine. Then in L4, the MT, when it is executing its core (i.e., AP Scheduler) (rather than an AP), schedules various AP's for use of the hardware machine. When an AP releases the control over the hardware machine voluntarily or under a mandate upon expiration of its process time-slice, the control goes to another thread (non-MT) or the core (AP scheduler) of the MT.

Also, in a manner analogous to the use of TT-AS's, AP's can use process-to-process atomic sections (PP-AS's). Therefore, synchronization between AP's can be realized via three different mechanisms: PP-AS, semaphore, and CREW monitor. Unlike in the case of kernel-threads, dynamic creation of AP's is also supported by the DREAM kernel.

Some advantages of this two-level scheduling approach are as follows. Scheduling of kernel-threads in a manner which is not interfered in any way by the AP scheduling makes it trivially easy to ensure that periodic kernel-threads provide essential services to the AP's precisely at the times determined at design-time. Also, the design of the MT-Core as a mediator among AP's considerably reduces the implementation complexity and also the context switching overhead to some extent over a direct AP-to-AP context switch design.

3. DREAM kernel as both a process execution engine and a real-time object execution engine

The remaining features of the DREAM kernel that do not belong to the process execution engine part, are the mechanisms directly related to supporting RTO.k objects. After a brief review of the essence of the RTO.k object structuring scheme in Section 3.1, those support mechanisms are discussed.

3.1 The RTO.k object structuring scheme

An initial abstract framework of the *RTO.k object model*, also called the time-triggered real-time object (TT_RTO) model, came out of the attempt by the first co-author and Hermann Kopetz at the Technical University of Vienna to find a proper extension of the basic object model which is highly cost-effective in development of *hard-real-time* application systems. Based on the initial abstract framework formulated in late 1980's [Kop90], a concrete syntactic structure and execution semantics was developed in recent years [Kim94a, Kim94b, Kim94c].

The basic structure of an RTO.k object is depicted in Figure 2. It is an extension of the conventional object model(s) and two most important and unique extensions are the following:

(a) *Two clearly separated groups of methods:*

For some methods of an RTO.k object, a real-time clock serves as the mechanism for triggering the method executions as the clock reaches some values specified at design time and such methods are called time-triggered (TT-) methods, also called the spontaneous methods (SpM's), and *clearly* separated from the conventional service methods (SvM's) triggered by messages from clients. The two types of methods in an RTO.k object are different not only in the way their executions are triggered but also in that

> "actions to be taken at real times *which can be determined at the design time* can appear only in SpM's".

Therefore, actions of the type "at constant-clock-value do S" or the type "sleep-until constant-clock-value" can appear only in SpM's. Incorporation of SpM's means introducing the potential for the following two new types of concurrent executions of object methods in addition to the potential for concurrent executions of SvM's that exist in conventional objects.

- (Type I) Concurrency among SpM executions: This concurrency is specified in an implicit but natural manner, e.g., two SpM's designed to be triggered at 10 am.
- (Type II) Concurrency between SpM executions and SvM executions.

(b) *Basic concurrency constraint (BCC):*

In order to dramatically reduce the designer's efforts in guaranteeing timely service capabilities of RTO.k objects, the execution rule which prevents conflicts between SpM's and SvM's is incorporated. Basically, *activation of an SvM triggered by a message from an external client is allowed only when potentially conflicting SpM executions are not in place.* To be exact, when a message-triggered SvM is not free of conflict with an SpM in accessing the same portion of the object data space (ODS), execution of the former method (SvM) must not be allowed in a time zone earmarked for a TT-execution of the latter method (SpM). This restriction is called the basic concurrency constraint (BCC). Therefore, executions of SpM's are not disturbed by SvM executions and triggering times of SpM's are fixed at the design time. If a statement of the type "at 10am do S" appears in an SpM, its timely

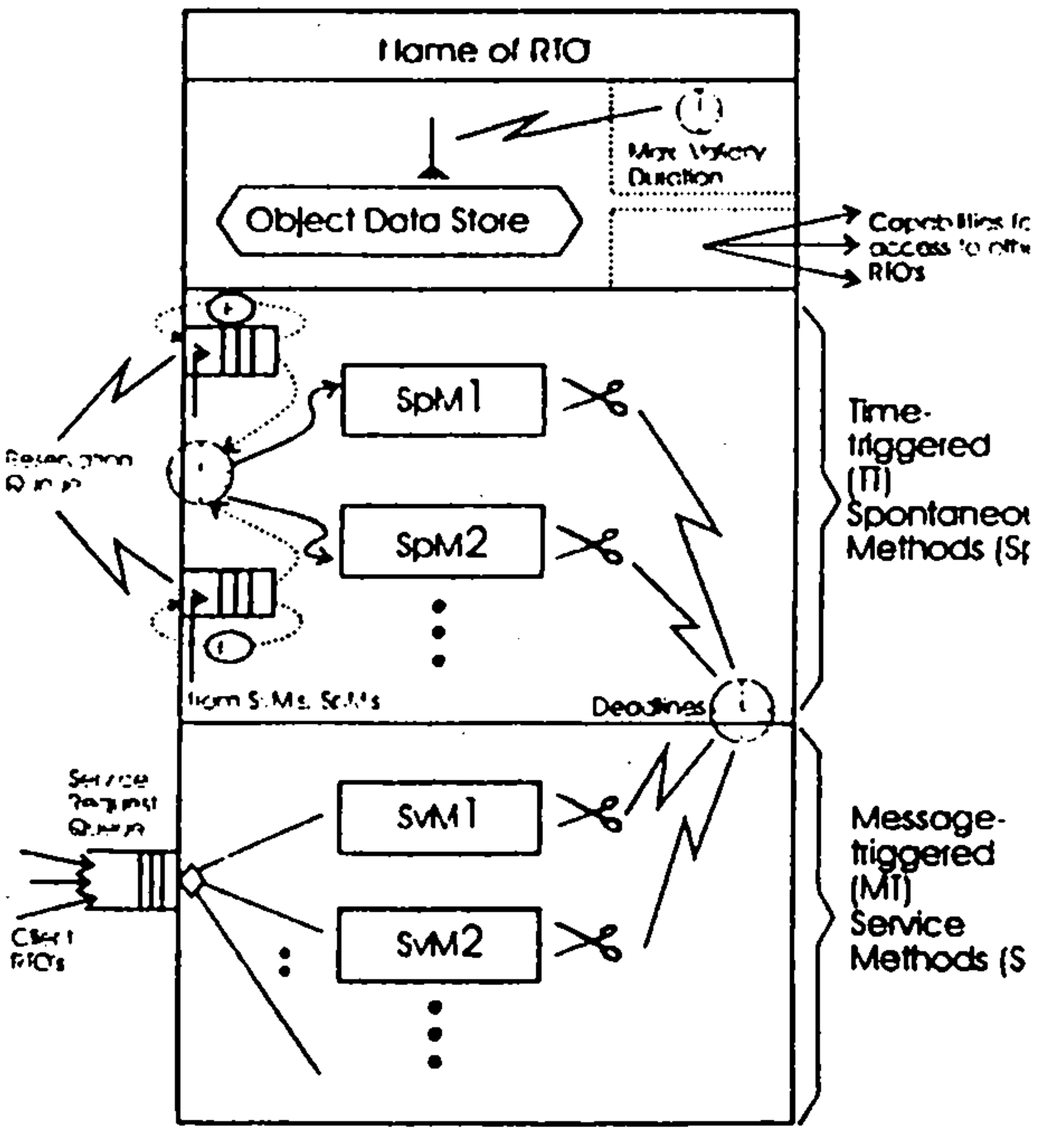


Figure 2. Structure of the RTO.k object model (Adapted from [Kim94c])

execution can be easily assured.

The above two features make the RTO.k object model clearly distinguished from other proposed real-time object models [Att91, Ish92, Tak92]. In addition, the RTO.k object contains the following features not found in the conventional object model(s):

(c) For each execution of a method of an RTO.k object, a deadline is imposed.

(d) Real-time data contained in an RTO.k object become invalid after the interval called the maximum validity duration passes.

Triggering times for SpM's must be fully specified as constants during the design time. Those real-time constants appear in the first clause of an SpM specification called the autonomous activation condition (AAC) section. An example of an AAC is:

"for t = from 10am to 10:50am every 30min
start-during (t, t+5min) finish-by t+10min"

which has the same effect as

{"start-during (10am, 10:05am) finish-by
start_time+10min",
"start-during (10:30am, 10:35am) finish-by
start_time+10min"}

A provision is also made for making the AAC section of an SpM contain only candidate triggering times, not actual triggering times, so that a subset of the candidate triggering times indicated in the AAC section may be dynamically chosen for actual triggering. Such a dynamic selection occurs when an SvM within the same RTO.k object requests future executions of a specific SpM.

An underlying design philosophy of the RTO.k object model is that an RTCS will always take the form of a network of RTO.k objects. The designer of each RTO.k object provides a guarantee of timely service capabilities of the object by indicating the *deadline for every output* produced by each SvM (and each SpM which may be executed on requests from SvM's) in the specification of the SvM (and some relevant SpM's) advertised to the designers of potential client objects. Before determining the deadline specification, the server object designer must convince himself/herself that with the object execution engine (hardware plus operating system) available, the server object can be implemented to always execute the SvM such that the output action is performed within the deadline. Again, the BCC contributes to major reduction of these burdens imposed on the designer.

The RTO.k object model is effective not only in the multiple-level abstraction of real-time (computer) control systems under design but also in the accurate representation and simulation of the application environments. In fact, it enables uniform structuring of control computer systems and application environment simulators [Kim94b] and this presents considerable potential benefits to the system engineers.

3.2 A mapping of an RTO.k object to a PCD-program

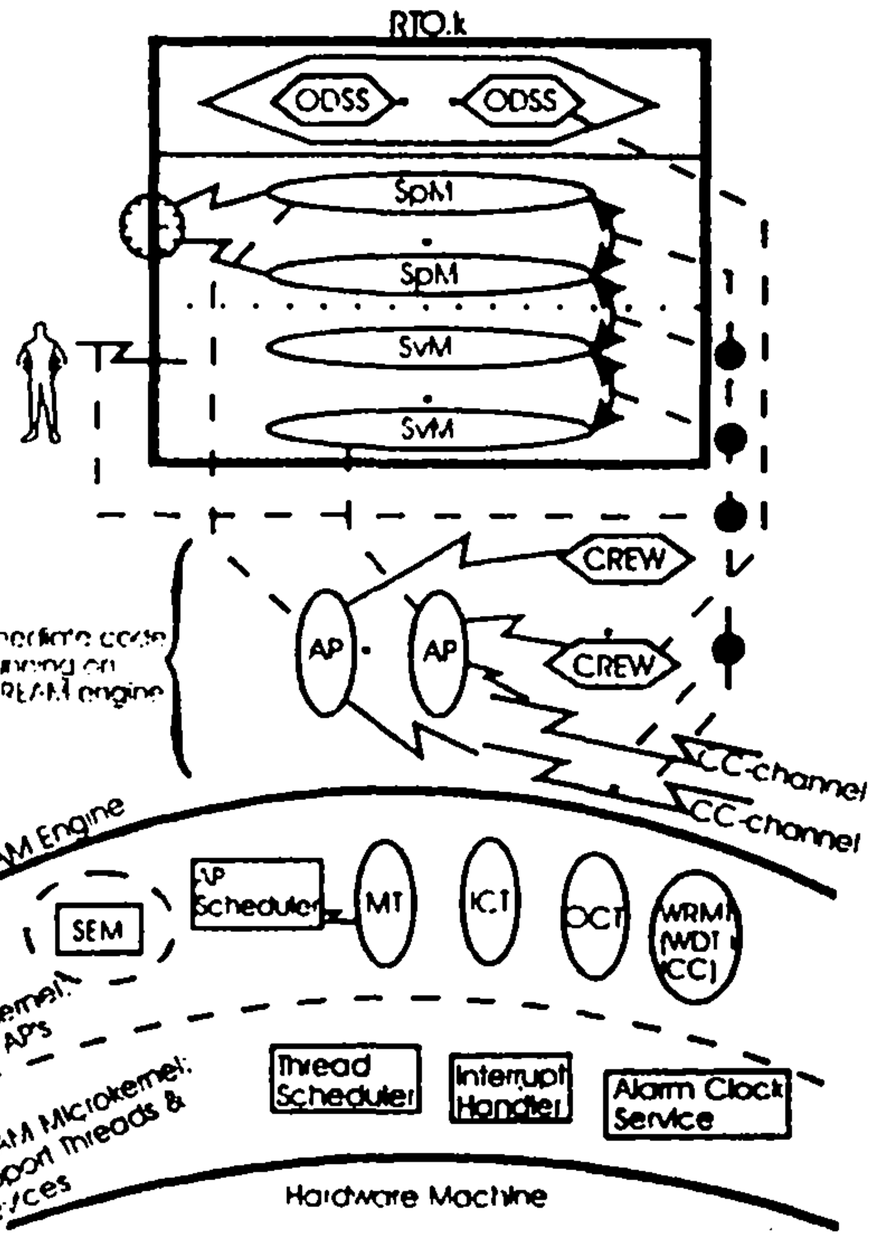


Figure 3. Mapping an RTO.k object to a PCD-program

When the DREAM kernel executes RTO.k objects, it actually executes equivalent PCD-programs, i.e., programs composed of processes, CREW monitors, and data fields (i.e., CC-channels). Figure 3 depicts the mapping relationship between components of an RTO.k object and the corresponding components of an equivalent PCD-program. As shown,
(1) object methods, both SpM's and SvM's, are mapped to processes,
(2) ODS segments (ODSS's) to CREW monitors,
(3) access paths to SvM's to CC-channels, and
(4) result-return paths to clients to CC-channels.
This way of utilizing CC-channels facilitates the transparency of object locations.

When an SpM is mapped to a process, the process must present to the DREAM kernel (1) the AAC of the SpM, (2) the associated deadline specification, and (3) access rights of the SpM for ODSS's. Similarly, when an SvM is mapped to a process, the process must present to the DREAM kernel (1) the associated deadline specification, (2) access rights of the SvM for ODSS's, and (3) other information related to its pipelined execution possibilities.

As mentioned earlier, this execution approach has the advantages of utilizing a general-purpose kernel (such as the DREAM kernel) supporting various types of concurrent and distributed application software.

3.3 The watchdog and real-time object management

thread (WRMT) supporting RTO.k objects

Of the kernel-threads shown earlier in Figure 1, the WRMT provides services needed mainly to support RTO.k objects. It is a periodic kernel-thread responsible for
(1) timely activation and future reservation of SpM's,
(2) activation of SvM's upon receiving of the corresponding service requests,
(3) enforcing the basic concurrency constraint, and
(4) deadline checking of the methods and invoking of an appropriate exception handler upon detection of a deadline violation.

A queue in layer L4, called the Reservation Queue (RvQ), holds the activation schedules of SpM's during next t time units. Reservations of SpM's are done by inserting the earliest start time and the latest start time of each SpM into the RvQ, which is to be activated during the next time period, p. The WRMT activates SpM's from the RvQ at their scheduled times by inserting the SpM's into the Ready AP Queue (RAPQ) and makes future reservations into the RvQ. The WRMT also activates an SvM upon receiving a service request message by inserting the SvM into the Ready AP Queue only if there is no possibility for the SvM to run into an ODS-conflict with any SpM. If an SpM which can run into an ODS-conflict with a SvM requested by an external client object is to be activated in d time units where d is less than the worst-case execution time of the SvM, the WRMT will not activate the execution of the SvM.

Use of the WRMT for supporting RTO.k object methods is an approach striking a good balance between the response time and the overhead, especially in view of the possibilities of using a dedicated process for similar functions or using the real-time alarm clock in L0 for some parts of the functions (e.g., TT-activation of an SpM).

3.4 Extensibility: incorporation of additional kernel-threads and system processes

As new I/O and networking devices are incorporated, the DREAM kernel can be expanded in several ways. First, device interrupt handlers must be inserted into layer L1 or L3 depending upon the data transfer bandwidth of the new device and the required response to a request of the device for an attention. Then, a decision must be made whether to impose additional responsibility for supporting the new device on the ICT or to introduce a new dedicated kernel-thread or to introduce a new dedicated AP.

4. A prototype implementation of the DREAM kernel and implementation techniques adopted

The first prototype DREAM kernel, v.D2, was implemented by the co-authors to run on a network of PC's connected via Ethernet and equipped with Intel 80486 processors, DOS-BIOS device drivers, and the Packet Ethernet driver. Services of the DREAM kernel including process management services can be requested from within a C++ program (representing an implementation of an RTO.k object) via calls for DREAM library routines.

4.1 Internal RTO.k object structuring

All components of the DREAM kernel v.D2 were structured in OO-forms, many times in RTO.k object forms. Initially, the fact that the naturalness and the representational power of the RTO.k object model were prominently shown even in this context looked very interesting. However, through subsequent reflections, we have come to the realization that a kernel is nothing but another kind of a real-time program and thus the applicability of the RTO.k object structuring scheme to the implementation of the DREAM kernel v.D2 should have been expected if the RTO.k scheme were indeed an appropriate approach for structuring NG RTCS's.

4.2 Inter-process-group communication facilities

In the HU-DF scheme [Kim95a], a process can freely connect itself to or disconnect itself from a multicast logical channel, which is somewhat restricted in the original DF scheme.

The DREAM kernel not only supports interprocess communication through CC-channels but also supports communication through ports [Ras89]. A port is a message communication channel with only a single AP as a receiver. An AP may notify its intention to the kernel to function as a receiver to a specific port. In such a case, the kernel makes sure that no other AP's are granted the receive rights to that port.

4.3 Layer L0 services

Layer L0 provides the following essential services to the upper layers.
(1) time-slice generation,
(2) alarm clock service to upper layers,
(3) context switch. It is essential to include the context switch routine in L0 because during a context switch, a timer interrupt service routine must not be invoked. In fact, the context switch routine is the only routine in the entire DREAM kernel v.D2 that is written in an assembly language.

4.4 An RTO.k structured application running on DREAM kernel v.D2

A preliminary version of the prototype DREAM kernel was used in an experimental development of a defense system with its application environment simulator [Kim94b] and has since been replaced by the current version (v.D2). This real-time distributed application software consists of nine different types of tailorable RTO.k objects and runs on a network of five or more PC's. In this experimental effort, translation of the RTO.k object structured program into an equivalent PCD-program was done manually. Some real-time fault tolerance capabilities were also implemented. The

application software consists of approximately 30,000 lines of C++ code. This experiment gave us a considerable amount of confidence in the appropriateness of the architecture of the DREAM kernel as well as the power of the RTO.k object structuring scheme.

5. Conclusion

At present, the DREAM kernel is little more than a minimal-functionality model of a timeliness-guaranteed operating system kernel supporting real-time processes and RTO.k objects. It can be extended in several major directions. The most obvious among them is to extend it to utilize multi-processor architectures and highly parallel machine architectures. Adapting the DREAM kernel to commercial micro-kernel environments is another meaningful direction to pursue. We plan to implement in the near future another version of the DREAM kernel which uses the IBM micro-kernel as its component. Realization of NG real-time computing will require investment of sizable efforts by industry to revise and extend their existing operating system products to possess guaranteed timely service capabilities and it is our hope that industry find some of the approaches and principles exploited in the DREAM kernel useful in such efforts.

Acknowledgment: The research work reported here was supported in part by US Navy, NSWC Dahlgren Division under Contract No. N60921-92-C-0204, in part by the California Transportation Department via the UCI Institute for Transportation Studies, in part by the University of California MICRO Program under Grant No. 93-080, in part by Hitachi, Ltd, in part by Postech, and in part by ETRI. The efforts of L. F. Bacellar were also supported by Coordenação de Aperfeiçoamento de Pessoal de Nivel Superior - CAPES and by Universidade Federal do Rio de Janeiro - UFRJ both from Brazil.

References

[Att91] Attoui, A. and Schneider, M., "An Object Oriented Model for Parallel and Reactive Systems", *Proc. IEEE CS 12th Real-Time Systems Symp.*, 1991, pp. 84-93.

[Bri77] Brinch Hansen, P., "The Architecture of Concurrent Programs," Prentice-Hall, 1977.

[Ish92] Ishikawa, Y., Tokuda, H., and Mercer, C. W., "An Object-Oriented Real-Time Programming Language", *IEEE Computer*, October 1992, pp. 66-73.

[Kim94a] Kim, K.H. et al., "Distinguishing Features and Potential Roles of the RTO.k Object Model", *Proc. 1994 IEEE CS Workshop on Object-oriented Real-time Dependable Systems (WORDS)*, Oct. '94, Dana Point, pp.36-45.

[Kim94b] Kim, K.H. and Kopetz, H., "A Real-Time Object Model RTO.k and an Experimental Investigation of Its Potentials", *Proc. 1994 IEEE CS Computer Software and Applications Conf. (COMPSAC)*, Nov. 1994, Taipei, pp 392-402.

[Kim94c] Kim, K.H., "A Utopian View of Future Object-Oriented Real-Time Dependable Computer Systems", (Invited paper) *Proc. 1st Int'l Workshop on Real-Time Computing Systems and Applications*, Seoul, Korea, Dec. 1994, pp. 59-69.

[Kim95a] Kim, K.H., Mori, K., and Nakanishi, H., "Realization of Autonomous Decentralized Computing with the RTO.k Object Structuring Scheme and the HU-DF Inter-Process-Group Communication Scheme", *Proc. 1995 IEEE CS Int'l Symp. on Autonomous Decentralized Systems (ISADS)*, April 1995, Phoenix, pp.305-312.

[Kim95b] Kim, K.H., "Toward New-Generation Real-Time Object-Oriented Computing", *Proc. IEEE CS 5th Workshop on Future Trends of Distributed Computing Systems (FTDCS)*, Cheju Island, Aug. '95, pp.520-529.

[Kop89] Kopetz, H., Damm, A., Koza, C., Mulazzani, M., Wolfgang, S., Senft, C., and Zainlinger, R., "Distributed Fault-Tolerant Real-Time Systems: The Mars Approach", IEEE Micro, Feb. 1989, pp. 25-39.

[Kop90] Kopetz, H. and Kim, K.H., "Temporal Uncertainties in Interactions among Real-Time Objects", *Proc. IEEE Computer Society's 9th Symp. on Reliable Distributed Systems*, Huntsville, AL, Oct. 1990, pp.165-174.

[Mor93] Mori, K., "Autonomous Decentralized Systems: Concept, Data Field Architecture, and Future Trends", *Proc. IEEE CS Int'l Symp. on Autonomous Decentralized Systems (ISADS 93)*, Mar. 1993, Kawasaki, Japan, pp. 28-34.

[Ras89] Richard Rashid, Robert Baron, Alessandro Forin, David Golub, Michael Jones, Daniel Julin, Douglas Orr, Richard Sanzi, "Mach: A Foundation for Open Systems", *Proc. the IEEE Second Workshop on Workstation Operating Systems (WWOS2)*, September 1989, pp.109-113.

[Tak92] Takashio, K., and Tokoro, M., "DROL: An Object-Oriented Programming Language for Distributed Real-Time Systems", *Proc. OOPSLA*, 1992, pp. 276-294.

제 3 편

제2차년도 연차보고서

여 백

목차

제 2 차년도
연차 보고서

RTS 기술개발

RTS Technology Development

거시적 시뮬레이터 구축 기술 개발

연구기관
포항 공과 대학교

과 학 기 술 처

여 백

제 1 장 거시적 실시간 시뮬레이터 구축 기술 개발

1 절 서론

컴퓨터 하드웨어의 성능이 급속하게 발전함에 따라 컴퓨터의 응용분야 역시 그 영역을 빠르게 넓혀 가고 있다. 컴퓨터의 고성능화로 가능해진 응용 분야는 일반적으로 대규모의 행동이 복잡한 실시간 컴퓨터 시스템을 요구하고 있다.

실시간 시스템이란 시스템의 행동의 올바름이 기능적인 측면뿐만 아니라 시간적인 측면에 의해서도 결정되는 시스템으로 원자력 발전소 제어 시스템, 기상 인공위성 제어 시스템, 미사일 제어 시스템, 그리고 교통 정보 시스템 등이 이에 속한다. 이런 실시간 시스템은 그 규모가 방대하고, 행동이 복잡하며, 시간적 제약이 엄격한 특징을 가진다. 이러한 복잡한 행동 양식을 보이는 실시간 시스템의 개발에 있어서, 사용자 요구 사항의 분석은 매우 어려울 뿐만 아니라 시스템 개발 완료 시까지는 충분한 유효성 검사가 힘든 실정이다.

요구 사항의 명세에는 두 일단이 참여하게 된다. 하나는 시스템을 사용하는 단말 사용자이고 다른 하나는 시스템을 분석하는 분석자이다. 일반적으로 단말 사용자는 컴퓨터 시스템에 대한 지식이 없고 분석자는 개발될 특정 시스템에 대한 기반 지식이 없기 때문에 요구 사항 명세(Requirement Specification)는 비정형적이거나 모호하거나 불완전한 경우가 많다. 이러한 문제점을 해결하기 위해서 정형적인 방법(Formal Method)이 사용되지만 단말 사용자는 정형적인 명세서(Formal Specification)를 이해하기 힘들기 때문에 요구 사항 명세서의 유효성 검사를 하기가 힘들다. 만약 요구 사항 명세서의 오류가 소프트웨어 생명주기 후반기에 발견된다면 이 오류를 초기에 고치는 것에 비해 비용이 수백 배에 이른다는 것이 잘 알려져 있다.

사용자 요구 사항의 유효성 검사를 위한 방법은 크게 3가지 부류로 나뉘어 진다. 즉, 정형적인 방법, 명세의 수행과 시뮬레이션, 프로토타이핑 등이 있다. 이들 각각에 대해 아래에서 언급한다.

정형적인 방법에서는 사용자 요구 사항이 수학적 이론과 방법을 적용하여 정형적으로 명세화된 후에 안전성, 일치성과 완전성과 같은 시스템 성질이 수학적으로 검증된다. 그러므로 정형적인 방법은 시스템의 올바름(Correctness)를 보장할 수 있다.

시스템이 복잡해지고 실시간 성질을 가짐에 따라, 분석되어야 할 시간적 공간적 범위가 극도로 증가하게 된다. 그러므로 이러한 방법을 전체 시스템 명세에 적용하기에는 비실용적인 면이 있다. 또한 이러한 방법은 최종 시스템이 어떠한 모습으로 행동하는지를 보고 느낄 수 있는 방법을 제공하지 못하기 때문에 사용자로 부터 시스템의 유효성을 검증 받는 방법이라기 보다는 시스템의 분석에 더 적합한 방법이다.

시스템 명세를 수행시키기 위한 많은 노력이 있어 왔고 많은 명세 수행 방법이 제안되었다. 이 방법은 사용자의 유효성 검사를 위해 시뮬레이터를 이용하여 명세를 해석하고 시스템 행동을 시각적으로 보여 준다. 또 실시간 시스템을 위해 추계적인(Stochastic) 모델을 사용하여 실시간 시스템의 행동을 명세하고 시뮬레이션하고 유효성을 검증하기도 한다.

이러한 방법의 몇 가지 장점은 다음과 같다.

□ 시스템의 모습을 보고 느낄 수 있고 시스템의 설계와 구현 전에 사용자로 부터 빠른 검증을 받을 수 있다는 것이다. 이러한 방법을 사용하면 사용자가 요구 사항 명세 단계에 적극적으로 참여하여 요구 사항 추출과 검증 사이의 간격을 극도로 좁힐 수 있고 많은 선택 사항이 경제적으로 평가될 수 있다.

□ 정형적인 분석 방법으로는 적합하지 않은 분야를 보완할 수 있다. 예를 들면, 실시간 요구 사항의 행동적 민감성 테스트는 정형적인 방법으로는 불가능하다. 즉 명세를 수행시켜 전체 시스템 행동을 시뮬레이션 함으로써 실시간 요구 사항에 민감한 프로세스를 쉽게 찾아낼 수 있다.

하지만 명세의 수행 방법은 일반적으로 명세에 대해 올바름의 신뢰도를 높여 주는 역할은 하지만 이를 보장하지는 않는다. 시스템의 어떤 성질의 올바름을 보장하기 위해서는 가능한 모든 경우에 대해 검사를 해봐야 한다. 이것은 시스템의 크기가 커다란 경우에는 거의 불가능하다. 그러므로 정형적인 분석 방법과 명세의 수행을 통한 방법 모두가 요구 사항 분석을 위해 필수적이다.

하지만 정형적인 방법과 명세의 수행 방법 모두를 제공하는 방법은 그리 많지 않다. 개발중인 ASADAL(A System Analysis and Design Aid tooL system) CASE 도구는 이들 두 가지 방법을 모두 지원한다. 이 도구의 특징을 요약하면 다음과 같다.

□ 위험 요소가 있는 시스템 성질- 안전성, 생존성, 실시간 반응성-의 검증을 위해 정형적인 분석 방법을 제공한다.

□ 명세의 수행을 통해 생명 주기 초기 단계에 요구 사항의 유효성 검사 방법을 제공한다.

세부 과제 "거시적 실시간 시뮬레이터 구축 기술 개발"은 이 ASADAL CASE 도구상에 실시간 시스템의 명세와 그 실시간 시뮬레이션을 돕는 기술을 개발하는 것을 목표로 하고 있다.

1 차년도(1994.9-1995.8)에서는 거시적 실시간 시뮬레이션을 위한 시스템의 모델링 방법인 ASADAL 요구 분석 방법론을 개발하였고 그 방법론에 따라 RTET(Real-time Event Trace), TES(Time Enriched Statechart), DFD(Data Flow Diagram)을 그리는 그림 도구를 만들고 시뮬레이터 프로그램으로서의 이 그림들을 분석, 데이터베이스에

의미 있는 형태로 저장할 수 있게 하였다. 또, 이들 명세 언어[1]를 이용한 시스템의 명세를 시뮬레이션하는 정형적인 방법과 추계적인(Stochastic) 시뮬레이션을 하기 위한 기본프로세스 명세 언어와 시뮬레이션 드라이버 프로그램도 정의되었다.

당해년도(2 차년도)에서는 이를 더욱 발전된 형태로 개발하였는데, 우선 DFD 의 분할(Decomposition)과 TES 명세간의 관계, 분할된 DFD 의 시뮬레이션 방법 등에 대한 연구가 진행되었고 사용자 대화형 시뮬레이터를 비롯하여 추계적인 시뮬레이터를 위해 정의된 기본프로세스 명세와 시뮬레이션 드라이버 프로그램을 해석하여 시뮬레이션하는 프로그램을 개발, 기존의 시뮬레이터와 연결하여 추계적인 배치 모드의 시뮬레이션을 가능하게 하였다. 그리고 ASADAL 명세 언어로 만들어진 명세가 여러 실시간 제약 조건을 만족하는지의 여부를 검증하는 기술을 개발하였다. 또한 지능형 생산 공장을 시뮬레이션하기 위한 모델을 구축, ASADAL 방법론을 실제 환경에의 적용성을 높였으며 실제 그 모델을 이용하여 참여 기업(POSCON)에서 제시한 냉간 압연 시스템의 자동 철판 두께 제어기(AGC; Automatic Gauge Controller) 및 장력 제어기(SC; Speed Controller)를 명세, 시뮬레이션 하였다.

앞으로 이 장의 2 절에서는 더욱 상세화된 시뮬레이션 방법을 정리하였고 3 절에서는 사용자 대화형 시뮬레이터, 4 절에서는 추계적인 배치 모드의 시뮬레이터에 대해 말하고 정형적인 실시간 성질 검증 방법은 5 절에, 지능형 생산 공장 시뮬레이션 모델과 AGC 및 SC 의 예에 대해서는 6 절에서 설명하도록 하겠다.

2 절 시뮬레이션 방법

1 차년도 연구의 결과에 의해 DFD 와 TES, RTET 간의 연결 방식 및 그 의미, 그

---

[1] RTET, DFD, TES 그림들을 ASADA:L 명세 언어라 한다.

리고 그것을 시뮬레이션하는 방법이 연구, 개발되었다. 당해년도에는 이것에 하나 더 나아가 거대한 시스템 분석에 필수적인 단계적인 상세화(Stepwise Refinement)를 실현시키기 위한 분할(Decomposition)된 DFD 를 시뮬레이션하는 방법을 개발, 구현하였다.

이 절에서는 DFD 의 분할 상세화와 그 때 RTET, TES 와의 관계, 그리고 이들의 시뮬레이션에 대해 다룬다.

1. 시스템의 분할 상세화

시스템이 거대하고 복잡해짐에 따라 이를 사람이 다루기가 점점 어렵게 되어가고 있다. 이를 위해 많은 분석, 설계 방법들은 시스템을 단계적으로 상세화 하는 방법, 분할하는 방법 등을 고안하였다.

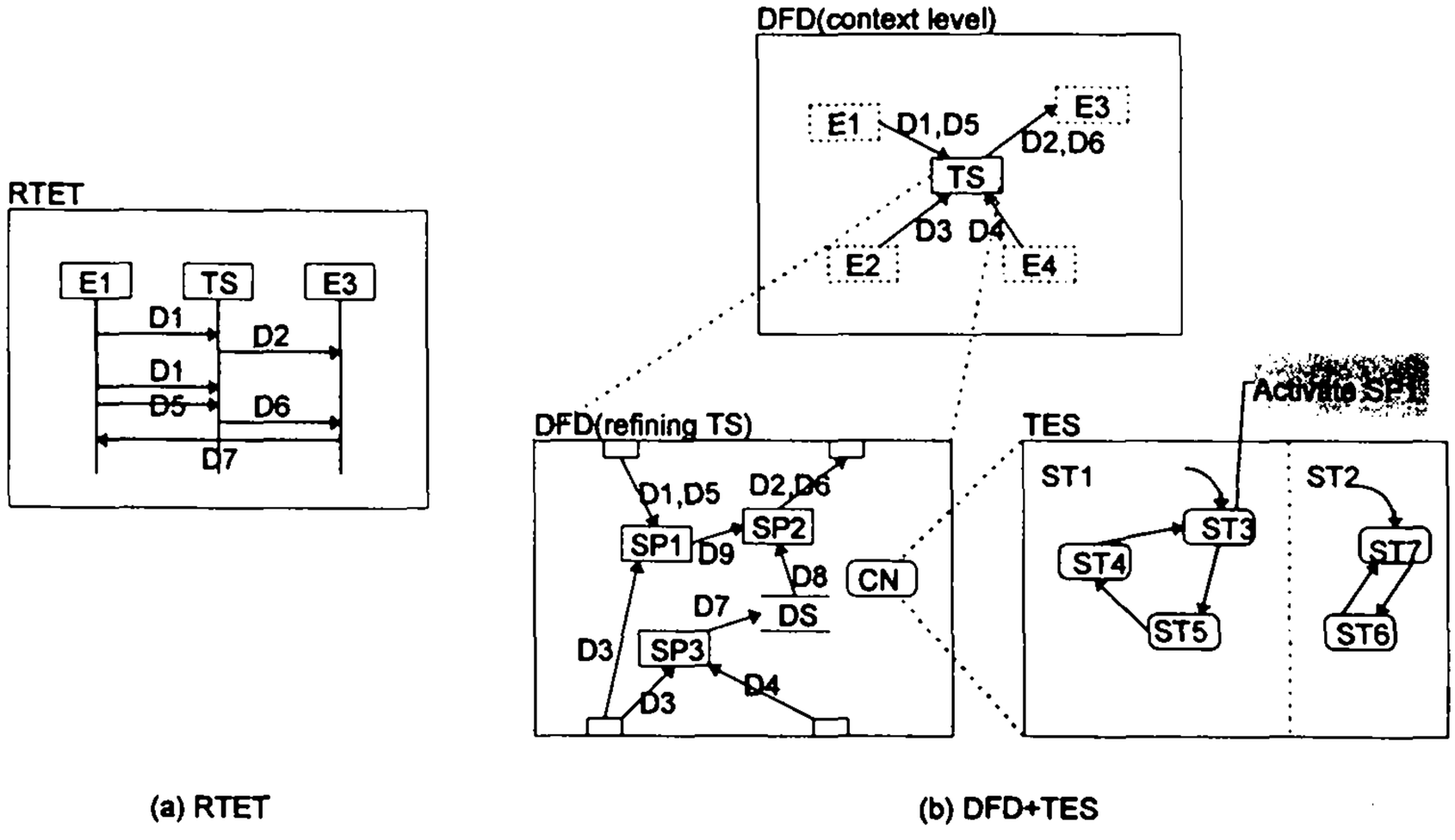


그림 1 ASADAL 명세 언어 및 모델

DFD 는 예전부터 프로세스(Process; Bubble)를 분할, 상세화 하는 방법을 갖추고 있었다. ASADAL 에서는 거대하고 복잡한 시스템의 단계적인 상세화를 DFD 로써

제공한다. 시스템의 구조는 DFD 의 분할 구조(Decomposition Hierarchy)로 표현되어지고 이제 각각의 DFD 에 제어기가 그려지고, 제어기는 하나의 TES 로 명세된다.

그림 1 은 이러한 ASADAL 명세 언어와 그 모델을 보이고 있다. 먼저 외부적 관점에서 본 시스템 명세를 그림의 (a)와 같이 RTET 을 이용해서 하게 된다. 이 그림에는 만들고자 하는 제어 시스템(Target System; TS)외에 시스템 내에 존재하는 관련된 다른 개체들이 표현되고, 제어 시스템과의 메세지 교환 외에 시나리오와 관련된 다른 개체들간의 것도 표현되어 시스템 환경 전체의 시나리오가 진행됨에 따른 메세지 흐름을 볼 수 있다.

이렇게 RTET 이 명세되면 RTET 에서 개체로 표현되었던 제어 시스템 자체에 대한 내부적 관점의 명세를 하게 된다. 이는 DFD 를 통해 이루어진다. 최상위 단계(Context Level)의 DFD 그림에서는 먼저 제어 시스템을 하나의 프로세스(Process; Bubble)로 놓고 그것과 데이터나 제어 신호(Control Signal)의 흐름이 있는 개체들을 외부 프로세스(External Process)로 놓고, 제어 시스템으로의 입력과 출력을 명세하게 된다. 이것이 그림 1 (b) 의 최상위 단계의 DFD 이다. 이 그림에는 제어 시스템으로의 모든 입력과 출력이 명세되어야 하며 따라서 RTET 에서 나타나는 제어 시스템으로의 입력과 출력은 DFD 에 나타나 있는 것이어야 한다.

그림 1 (b)의 최상위 단계 DFD 에는 제어 시스템을 "TS"라는 이름의 프로세스로 명세하였고, 이것이 분할 상세화된 것이 그 그림의 왼쪽 아래에 나타나 있는 DFD 이다. 이 DFD 는 "TS"프로세스가 실은 "SP1", "SP2", "SP3"의 세 프로세스들을 가지고 있으며 "TS"로 들어온 입력이 이들 프로세스에 어떻게 전해지고 또 이들이 어떻게 "TS"의 출력을 만들어 내는지를 보인다. 이 과정에서 "TS"의 입력과 출력이 그것이 상세화된 그림에서 완전히 같은 입력과 출력으로 나타나야 한다는 것은 주지의 사실이다.

분할 상세화 된 DFD("refining TS"라고 설명된)에서 모서리 둥근 사각형으로 표현된 이름이 "CN"인 제어기(Controller)가 있는데, 이 것이 그 오른쪽 그림과 같은 TES 로 상세화 되어 시스템의 상태 변화에 있어서 프로세스들이 어떻게 활성화되는지를 보이게 된다. 예를 들어 이 그림에서 초기 상태인 "ST3"은 보이는 바와 같이 "SP1" 프로세스를 활성화하게 된다.[2] 즉, 시스템이 "ST3" 상태로 될 때 "SP1"의 기능을 수행하고 "ST" 상태에서 다른 상태가 될 때 그 상태는 끝나게 된다.

일반적으로 하나의 DFD 에는 하나의 제어기가 있어서 그 DFD 에 명세된 프로세스들의 행위(Behavior)들이 시스템 상태 변화에 따라 어떻게 일어나는지 명세하게 된다. 하지만 그림에서와 같이 최상위 단계에서는 프로세스가 하나만 있고 따라서 제어기는 무의미하다. 즉, 이 프로세스는 항상 활성(Active)이다. 최상위 단계뿐 아니라 이와 같이 제어기가 없는 DFD 그림의 프로세스들은 모두 동시에(Concurrent) 활성인 것으로 본다. 왜냐하면 이러한 행위를 나타내는 TES 는 쓸 데 없는(Trivial) 것이기 때문이다.

하나의 프로세스는 이를 제어하는 제어기의 TES 그림에서 단 하나의 상태에서만 활성화한다. 그리고 시스템은 항상 어떤 기능을 행하게 되므로 모든 TES 그림에 나타난 상태들은 어떠한 프로세스를 활성화시켜야 한다. 또 여러 프로세스를 활성화시키는 상태가 있을 수 있다. 하지만 이 경우 그들을 연속적으로(Sequentially) 활성화시키는 것으로 보기 때문에 동시에(Concurrently) 활성화되는 프로세스들의 경우 여러 개의 병렬 상태(Parallel State)에서 활성화하도록 명세되어야 한다. 예를 들어 그림 1(b)의 TES 그림에서 "ST3", "ST4"에서 활성시키는 프로세스들은 동시에 활성화될 수 없지만 "ST3"과 "ST6"은 동시에 현재상태로 될 수 있으므로 그들이

---

[2] 그림에서 "activate..."의 표현은 임의의 것으로 ASADAL 명세의 문법(Syntax)은 아니다.

활성화하는 프로세스들은 동시에 수행된다.

그럼 이러한 각 그림들간의 관계와 DFD가 상세화 되었을 때 이들의 시뮬레이션이 어떻게 동작하는 지를 보도록 하겠다.

2. 거대, 복잡한 명세의 시뮬레이션

거대하고 복잡한 명세를 하게 되면 앞서 설명했 듯 DFD의 상세화가 필수적이며 한 프로세스와 그것이 상세화된 DFD간의 관계에 대해서도 앞서 언급했다.

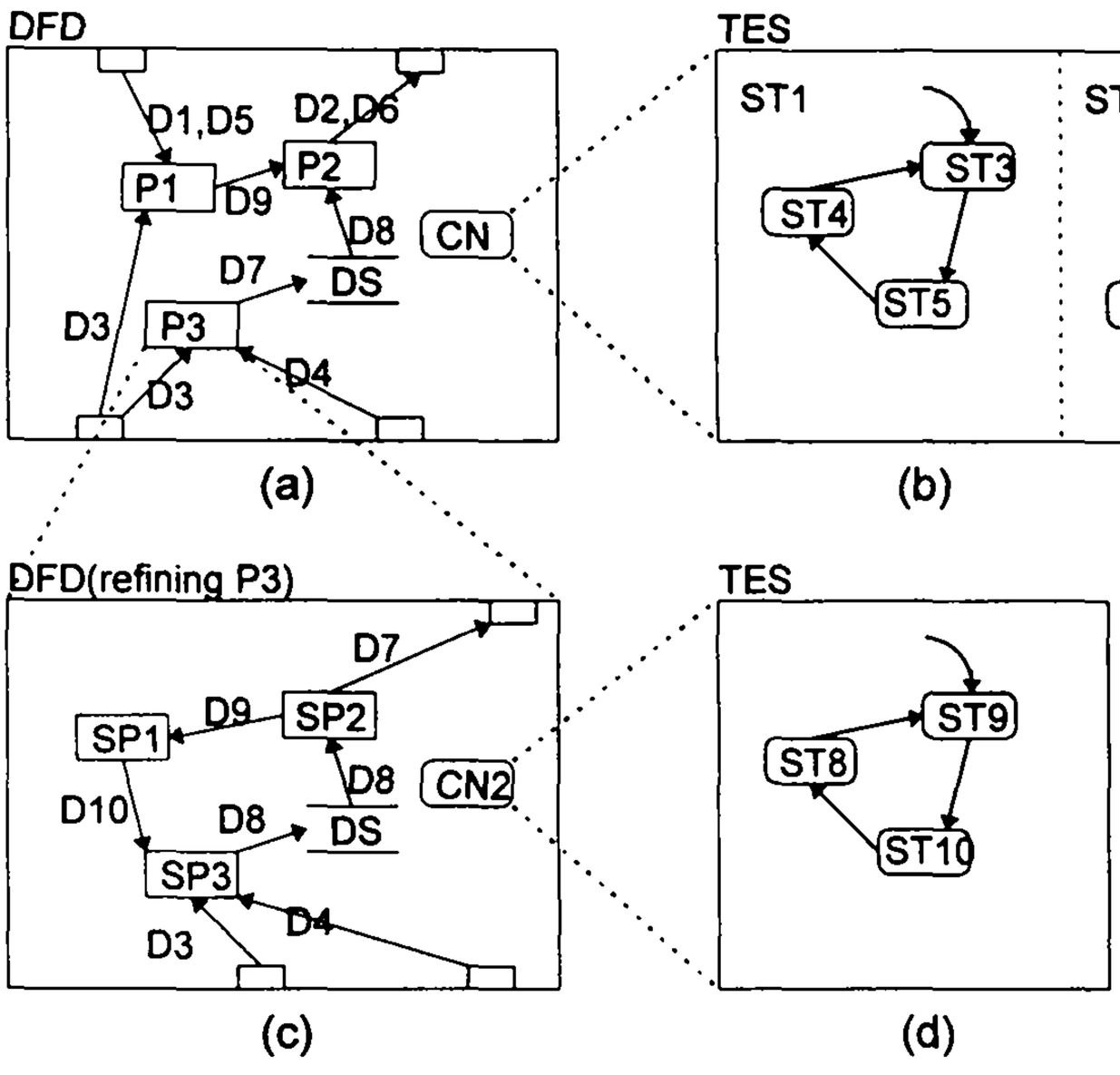


그림 2 DFD 구조와 시뮬레이션

그림 2는 DFD와 그 중 한 프로세스 "P3"을 상세화 한 DFD를 보인 것이다. (a)의 DFD도 어느 한 프로세스를 상세화한 것으로 그 기능들을 세부적으로 "P1", "P2", "P3"로 나눈 것인데, 이들은 제어기 "CN"에 의해 제어되고 "P3"의 경우는 (c)와 같이 다시 상세화된다. 이 때 그림 (a)가 나타내는 프로세스가 활성화될 때 이 그림에

나타나 있는 기능, 행위들은 작동한다. 즉, 비활성일 때 모든 하부 프로세스("P1"과 같은)들도 비 활성이 된다. 또한 제어기 "CN"도 아무런 행위를 보이지 않는다. 이 프로세스가 활성화되면 제어기 "CN"의 시뮬레이션이 시작된다. 제어기에 대한 시뮬레이션이 되면 현재의 상태가 바뀜에 따라 활성화되는 프로세스들이 바뀐다. 결국, "CN"은 프로세스 "P1", "P2", "P3"를 활성화, 비활성화 하는 역할을 한다. 이 제어기 상태변화는 프로세스들의 기능 수행에 의한 데이터나 조건(조건) 변수 값의 변화, 또는 데이터의 발생, 프로세스의 중단(Termination), 시간 경과(Timeout)와 같은 사건(Event)의 발생에 의해 이루어진다. 단, 데이터나 조건의 변수값은 전역으로 어느 곳의 제어기에서도 참조할 수 있지만 사건의 발생은 그 발생 원인이 게어할 DFD 그림 내에 있는 것으로 한정한다. 그림이 너무 목잡해지고, 참조 대상을 찾기 어려워지는 현상을 막기 위해서이며 그것으로도 한 그림을 제어하기 위한 정보는 충분하리라고 생각한다.

그림 2의 (b)에서 상태 "ST3"이 (a)의 프로세스 "P3"을 활성화하면 (c)의 제어기 명세인 (d)가 행위를 보이기 시작한다. 처음엔 초기 상태 "ST9"이 현재상태로 될 것이다. 그러면서 그 상태에 활성해야 할 (c)의 프로세스를 다시 활성화할 것이다. 그 후 상황 변화에 따라 TES에서 상태 전이(State Transition)이 발생할 것이며 그에 맞게 (c)의 프로세스들이 동적으로 활성화된다. 그러다가 어느 순간 (b)에서 "ST3" 상태를 나가면 프로세스 "P3"는 비활성화되고 따라서 (d)도 비 활성화된다. 이 과정에서 (d)에 상태 기록자(History Connector)가 없으면 다음 활성시엔 "ST9"가 다시 현재 상태가 활 것이다. 그리고 (c)의 프로세스들도 비활성화되는데, 이들은 하던 일을 멈추고 그 특성에 따라 다음 활성시에 처음부터 다시 기능을 수행하거나 이전에 멈춘 곳으로부터 계속 수행해 나가게 된다.

그림 2(a)의 "P3"으로 들어가는 "D3", "D4", 그리고 나오는 "D7"의 데이터들은 (c)의 같은 이름의 데이터들과 완전히 같은 것이다. 앞서 말했 듯 (c)는 (a)의 "P3"

율 상세화한 것이기 때문이다. 예를 들어 "P3"으로 "D3"의 데이터가 들어온다는 것은 (c)의 "SP3"에 "D3"이 들어온다는 것과 같은 말이다. 따라서 시뮬레이션을 할 때 (a)와 같은 상위 단계의 프로세스에 전해지는 데이터는 그것을 상세화한 그림에 그대로 전해져야 한다. 마찬가지로 상세화된 그림에서 발생한 데이터도 그 상위 단계에 전해져야 한다. ASADAL DFD 모델에서 데이터 흐름(Data Flow)은 각각 하나씩의 큐(Queue; First In First Out)로 설정되어 진다. 즉, 프로세스에서 나오는 데이터는 그 흐름에 저장되고, 그것은 이 큐안에 계속 쌓인다. 그리고 그것은 도착한 순서대로 차례대로 사용하는 측에 전달된다. 일반적으로 프로그래밍 언어에서는 A 에서 B 로의 데이터 전달이 A 가 B 에게 전해주거나 B 가 A 로부터 가져가는 형태를 취한다. ASADAL 모델에서는 이러한 전달을 단순히 큐로 보고 있으므로 이러한 데이터 전달은 A 가 큐에 넣고, B 가 큐에서 가져가는 형식을 따르게 된다. 이러한 방식은 전달하는 측과 받는 측간의 시간, 공간적인 의존성을 거의 요하지 않으므로 시스템 모델링을 쉽게 만들어준다. 그리고 이러한 모델은 시스템의 큐잉 모델(Queueing Model)에 근거한 분석을 가능하게 해 준다. 이 분석에 대해서는 후에 설명하겠다.

3. 하향식 시뮬레이션

ASADAL 명세는 하향식으로 이루어진다. 요구 사항의 명세를 해 나가는 단계에서 아직 덜 상세화된 단계(Gross Level)의 명세를 시뮬레이션하는 것은 조금이라도 빨리 오류나 문제점을 찾으려는 노력을 뒷받침하는 것으로 꼭 필요하다.

ASADAL 명세를 시뮬레이션하면서 완전히 명세되지 않은 요소들이나 외부 개체들의 시뮬레이션은 사용자 대화형의 경우 사용자에 의해 이루어지고 배치의 경우 시뮬레이션 드라이버에 의해 이루어진다.

프로세스가 DFD 그림이나 프로세스 명세로 아직 상세화되지 않았을 때 시 프로세스를 이용한 시뮬레이션의 결과는 아무런 의미있는 시간적 행위(Timing Behavior)

나 기능(Action)을 보이지 못한다. 따라서 적어도 이러한 프로세스에 대해서 시뮬레이션 드라이버 프로그램이나 프로세스 명세로 수행 시간이 명세되어야 합리적인 시간적 행위를 보일 수 있다. 예를 들어 우리는 간단히 프로세스에 그것이 얼마의 시간 동안 수행될 것인지를 추계적인 방법으로 명세할 수 있다. 또는 프로세스에 그 프로세스를 시뮬레이션할 때 데이터가 들어오면 얼마만큼의 수행 시간이 지난 후에 어떤 데이터가 나가고 또 어느 정도 지난 후에 어떤 데이터가 나가도록 명세할 수도 있다. 이런 식으로 프로세스 명세에 더 자세한 것을 집어 넣을 수록 더욱 자세한 시뮬레이션이 가능해진다.

이렇게 일단 프로세스 명세를 통해 시뮬레이션해 보았던 프로세스에 대해 충분히 상세화할 준비가 되면 그것을 다른 DFD 그림으로 상세화하거나 더 자세한 프로세스 명세로 확장하거나 하면 된다.

이상으로 DFD의 상세화에 의한 복잡한 모델의 시뮬레이션에 대한 설명은 마치기로 하고, 다음 장엔 개발된 사용자 대화형 시뮬레이터의 사용자 인터페이스와 그 실제 작동환경에 대해 알아보겠다.

3 절 사용자 대화형(User Interactive) 시뮬레이터

ASADAL 명세 방법론에서는 세 가지 관점에서 시스템을 분석, 각각 RTET(Real-time Event Trace)와 DFD(Data Flow Diagram), TES(Time Enriched Statechart)의 세 가지 명세 언어로 명세하도록 하고 있다. 지난 해(1 차년도)의 결과 이러한 명세 언어의 그림 도구와 간단한 시뮬레이터가 설계, 개발되었다. 당해년도 연구에서는 2 절의 내용과 같이 상세화 된 시뮬레이션 방법을 구현, 시뮬레이터를 확장하였다.

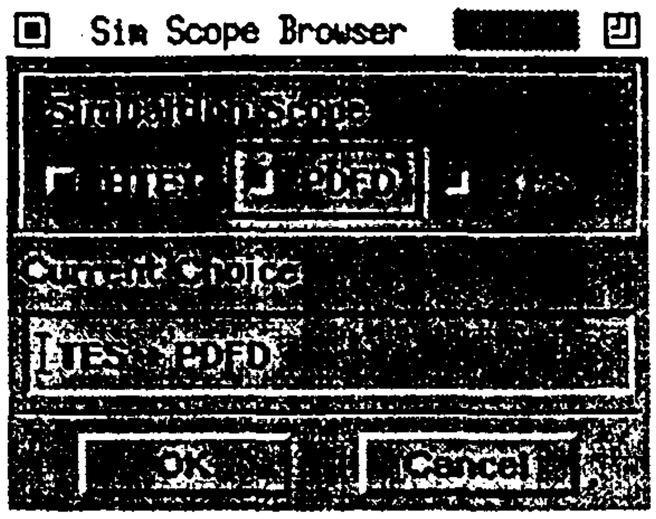


그림 3 시뮬레이션 대상 그림 선택 창

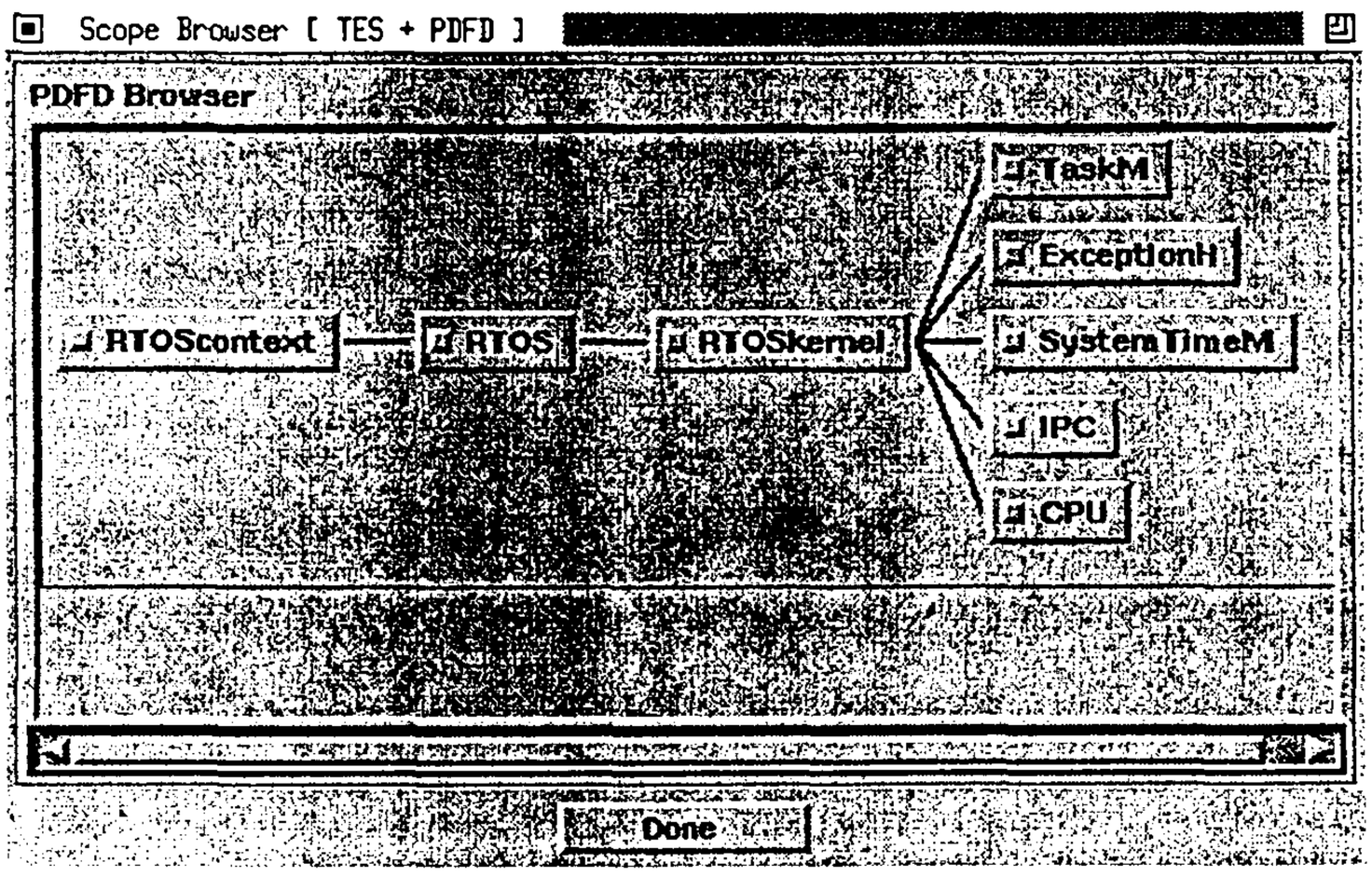


그림 4 시뮬레이션 범위 결정 창

이 장에서는 확장된 사용자 대화형 시뮬레이터의 기능에 대해 설명하고자 한다.

사용자 대화형 시뮬레이터를 실행시키면 우선 시뮬레이션의 범위(Scope)을 결정하기 위한 창을 볼 수 있는데, 이것이 그림 3과 그림 4와 같은 것들이다.

그림 3의 창을 이용하여 시뮬레이션할 대상을 TES, DFD+TES, TES+DFD+RTET의 세 가지중에서 선택할 수 있다. 이들의 선택에 따른 시뮬레이션의 의미는 지난 해의 보고서를 참고하기 바란다.

그림 4의 창을 이용하면 시뮬레이션의 범위를 결정할 수 있다. TES 만을 시뮬레이션할 때는 시뮬레이션하기 원하는 TES의 나무 구조(Tree) 중에서 일부(Subtree)를 선택하고, DFD가 포함된 시뮬레이션에서도 역시 원하는 DFD 나무 구조에서의 일부를 선택하게 된다. 이렇게 시뮬레이션할 범위인 DFD 나무 구조가 결정되면 이 나무의 뿌리(Root Node)로 들어오는 데이터, 제어 시그널들이 시뮬레이션할 때 외부로부터의 입력으로 설정되어 시뮬레이션 된다.

이렇게 시뮬레이션의 범위가 결정되면 처음 만나게 되는 것이 그림 5의 제어

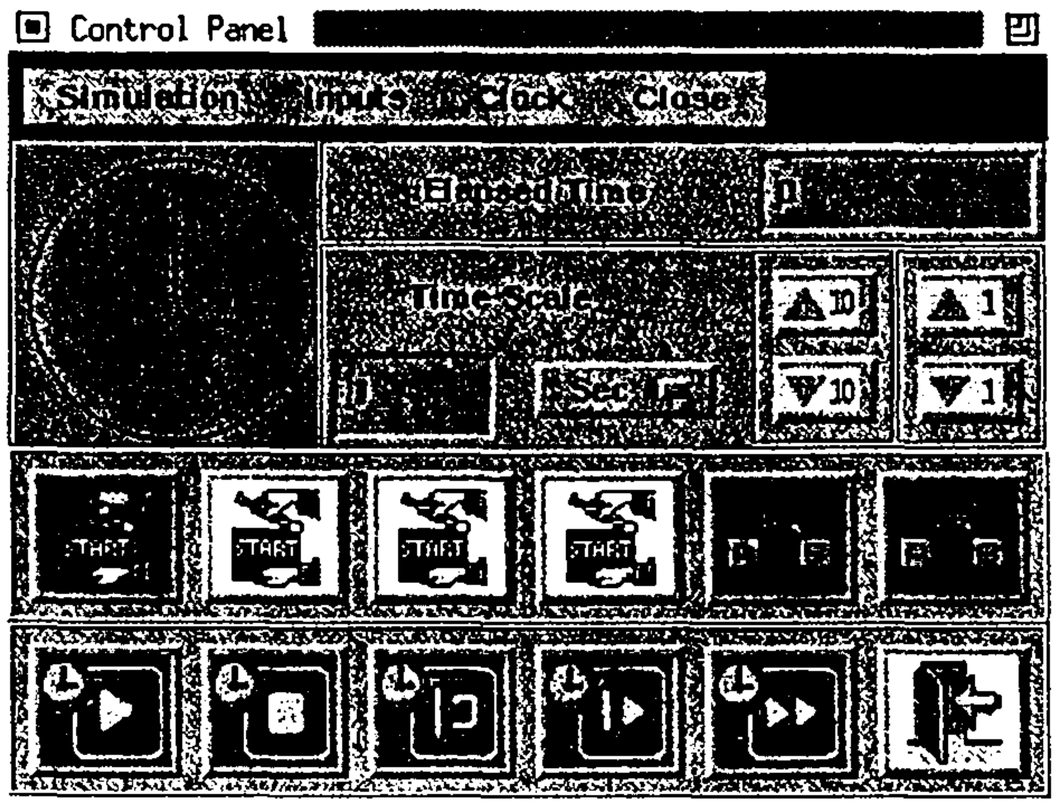


그림 5 제어 파넬

파넬과 그림 6의 나무 구조 보기이다.

사용자 대화형 시뮬레이션시 사용자는 이 제어 파넬을 이용하게 되는데, 그 기능을 간단히 설명하면 다음과 같다.

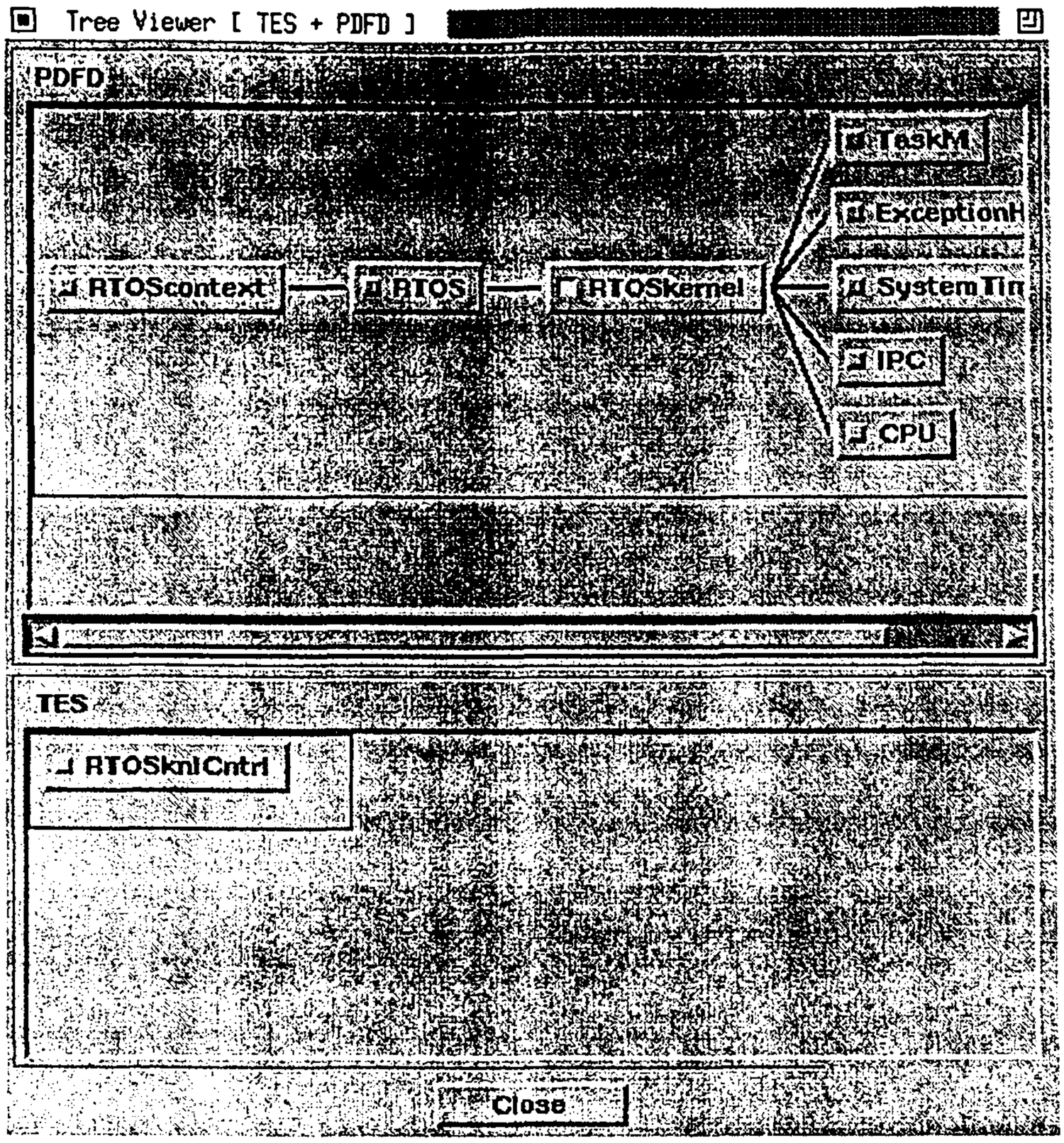


그림 6 나무 구조 보기

□ Step

시스템 시계(System Clock)는 흐르지 않은 상태에서 현재 발생된 사건과 만족된 조건에 따라 상태 전이가 일어 난다.

□ Continue

더 이상의 상태 전이가 일어 나지 않을 동안 Step 을 연속해서 시뮬레이션한 것과 같은 효과이다.

□ **Schedule**

Scheduling 중인 Action 이나 Timeout Event 중 남은 시간(Delta-Time)이 가장 작은 것으로 자신은 물론 다른 것의 남은 시간을 빼 준다. 즉, Continue 과 달리 시간적으로 남은 시간이 영(Zero)가 나오는 것이 있을 때까지 Clock 1 을 계속해 주는 것이다.

□ Clock 1

시스템 시계를 일 타임 단위만큼 증가시킨다.

□ Clock N

시스템 시계를 N 타임 단위만큼 증가시킨다.

물론, 이 각각의 기능들은 아이콘으로 그려져 있고, 원하는 경우 위에 붙어 있는 메뉴로부터 접근할 수도 있다.

ASADAL 시뮬레이터의 시간 발생기는 사용자가 제어 파넬을 이용하여 직접 시뮬레이션 스텝(Step)이나 시간의 흐름을 알리는 형태에서 벗어나, 시간과 시뮬레이션 스텝을 자동으로 증가시켜 주어 사용자는 시뮬레이션에 필요한 데이터 값의 변화나 사건의 발생만을 이용하여 간단하게 시뮬레이션할 수 있다. 제어 파넬은 얼마나 시간이 흘렀는 지를 보이고 자동 시간 발생에 의해 시뮬레이션을 하는 경우 시간과 스텝을 일정한 주기로 만들어 시뮬레이터에 전하는데, 현재 시간이 흐르는 모습과 시간 단위(Time Unit) 1 이 실제 시간 얼마마다 증가되는지를 정할 수 있어서 시스템이 다루는 시간의 크기에 따라 사용자가 원하는 빠르기로 볼 수 있다.

그림 6 의 나무 구조 보기 화면 역시 제어 파넬과 함께 시뮬레이션 도중에 항상 있으면서 사용자 대화형 시뮬레이션을 돕는다. 나무 구조 보기는 위아래의 두 부분으로 나뉘어 있는데 윗 부분은 DFD 의 나무 구조를 보여 주고 아래 부분은 선택된

프로세스를 제어하는 제어기를 보여준다. 이 나무 구조 보기는 시뮬레이션 진행 중 어느 프로세스가, 어느 제어기가 활성이 되었는지를 활성된 프로세스는 빨간 색으로 표시함으로써 보여준다. 또, 이 나무 구조 보기를 이용하면 DFD 나무 구조 상의 어떠한 그림도 원하면 볼 수 있다. 즉, 원하는 프로세스나 제어기를 나타내는 버튼을 누르면 그 그림을 보이는 창이 나타나게 되고, 나무 구조 보기 상에 그 그림이 현재 보여지고 있는 지도 보인다. 이렇게 나타나는 창이 그림 7과 그림 8이다.

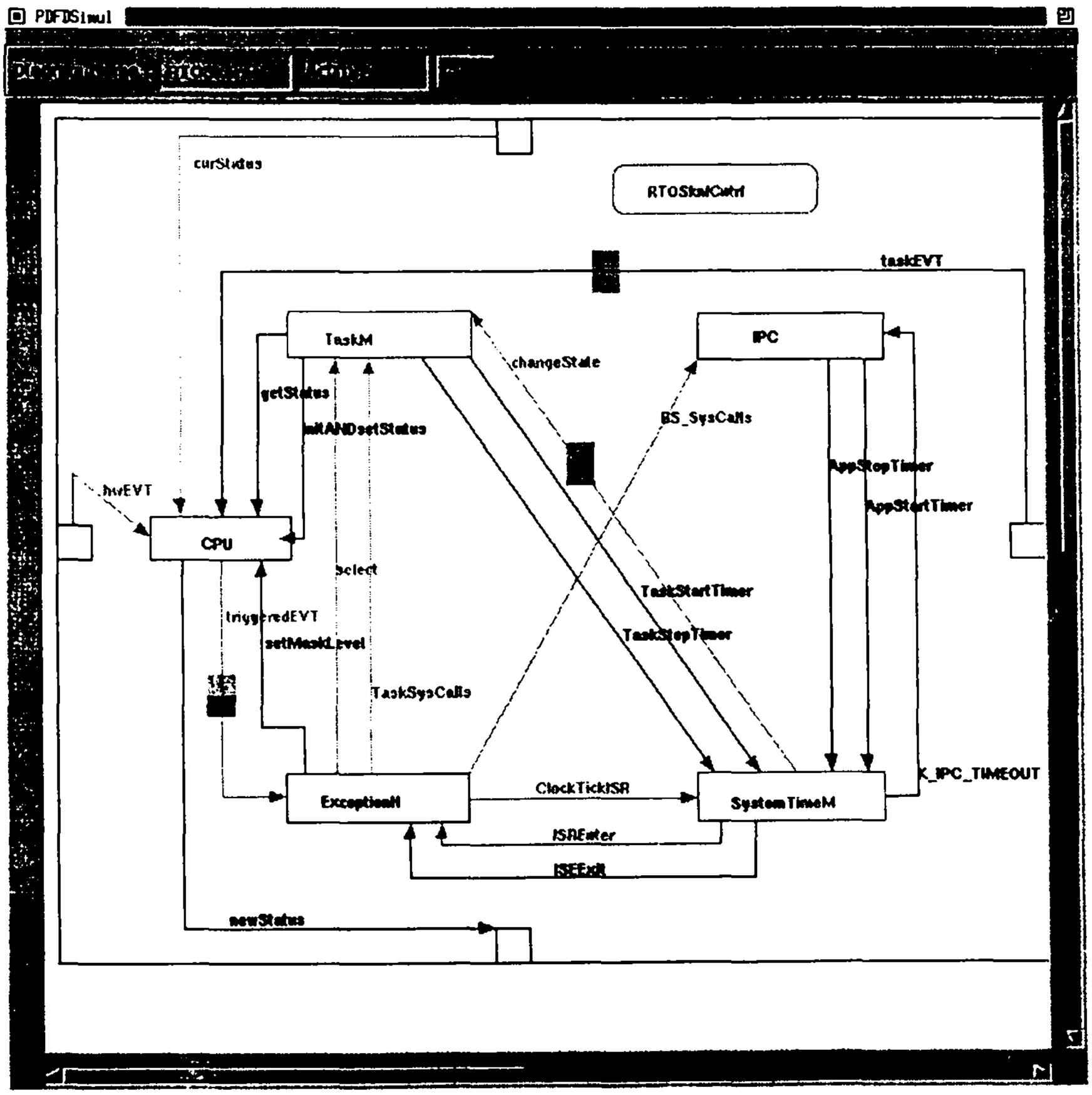


그림 7 DFD 시뮬레이션 창

그림 7의 DFD 시뮬레이션 창은 DFD가 시뮬레이션되는 모습을 보여준다. 이

창을 통해 사용자는 어느 프로세스가 현재 활성화되었는지를 프로세스 색의 변화로써 알 수 있고 데이터의 흐름도 그것의 색 변화로 볼 수 있다. 즉, 프로세스들이 활성화되고, 프로세싱을 거쳐 데이터를 큐(데이터 흐름이 큐인 것은 앞서 설명한 바 있다)에 넣고, 가져가는 등의 모습을 창에서 직접 볼 수 있다. 이 그림에서는 또한 큐에 쌓이는 데이터의 양을 시각적으로 보이기 위하여 큐 길이 표시자(Probe)를 달았는데, 이것은 사용자가 보기를 원하는 데이터 흐름 위에 언제든 만들어 사용할 수 있도록 했다. 이것을 사용하기 위해서는 사용자가 그림의 데이터 흐름 위에 마우스의 커서를 옮기고 오른쪽 마우스 버튼을 눌러서 나오는 메뉴에서 항목을 선택하면 된다.

이 창에서는 또한 그 DFD 그림이 나타내는 프로세스가 현재 활성화되었는지는 창 위쪽의 작은 창을 통해 볼 수 있다. 그 프로세스의 활성화 여부에 따라 그것에 속해 있는 제어기의 활성화 여부가 결정되며 그것도 제어의 색 변화를 통해 확인할 수 있다. 창 위쪽에는 이 외에 그 그림의 이름을 보여 주는 창과 그 그림을 닫을 때 사용하는 버튼 등이 있다.

그림 8은 TES 시뮬레이션 창을 보이고 있다. 이 창에서는 현재 시스템이 어느 상태이고 어떤 상태 전이에 의해 어떤 상태로 바뀌고 있는지가 보인다.

ASADAL 명세는 세 가지 다른 관점에서의 명세 언어들을 제공하고, 이들간의 연관 관계상의 완전성(Completeness), 일관성(Consistency)등의 문제를 해결할 수 있도록 노력을 기울이고 있다. 이 문제의 해결이 바로 시스템 개발에 참여하는 다방면의 사람들 간의 이해를 돕고 시스템에 대한 뷰(View)를 통일시키는 데 크게 기여할 것이기 때문이다. ASADAL 시뮬레이션은 이러한 문제에 대해서 어느 정도 해결책을 마련한다.

예를 들어 ASADAL 시뮬레이션을 이용하여 사용자는 손쉽게 ASADAL 명세상의

오류를 찾아낼 수 있다. 예를 들어 이 도구의 시험 단계에서 DFD 시뮬레이션 시 어떤 프로세스는 계속 활성화되지 않아 들어오는 데이터를 처리하지 못한다거나, 활성화되지만 그 주기가 들어오는 데이터를 만드는 프로세스보다 현저히 작아 미처 그것을 처리하지 못하고 있다거나 하는 것들이 손쉽게 보여졌고, 이것은 TES 명세를 고침으로써 해결되었다. 또는 RTET+TES, RTET+DFD+TES 의 시뮬레이션을 통해 DFD, TES 의 명세가 RTET 이 보이는 시나리오를 따르고 있는지, 시간적인 제약은 지키고 있는지 등을 시뮬레이션을 통해 검증해 볼 수 있다.

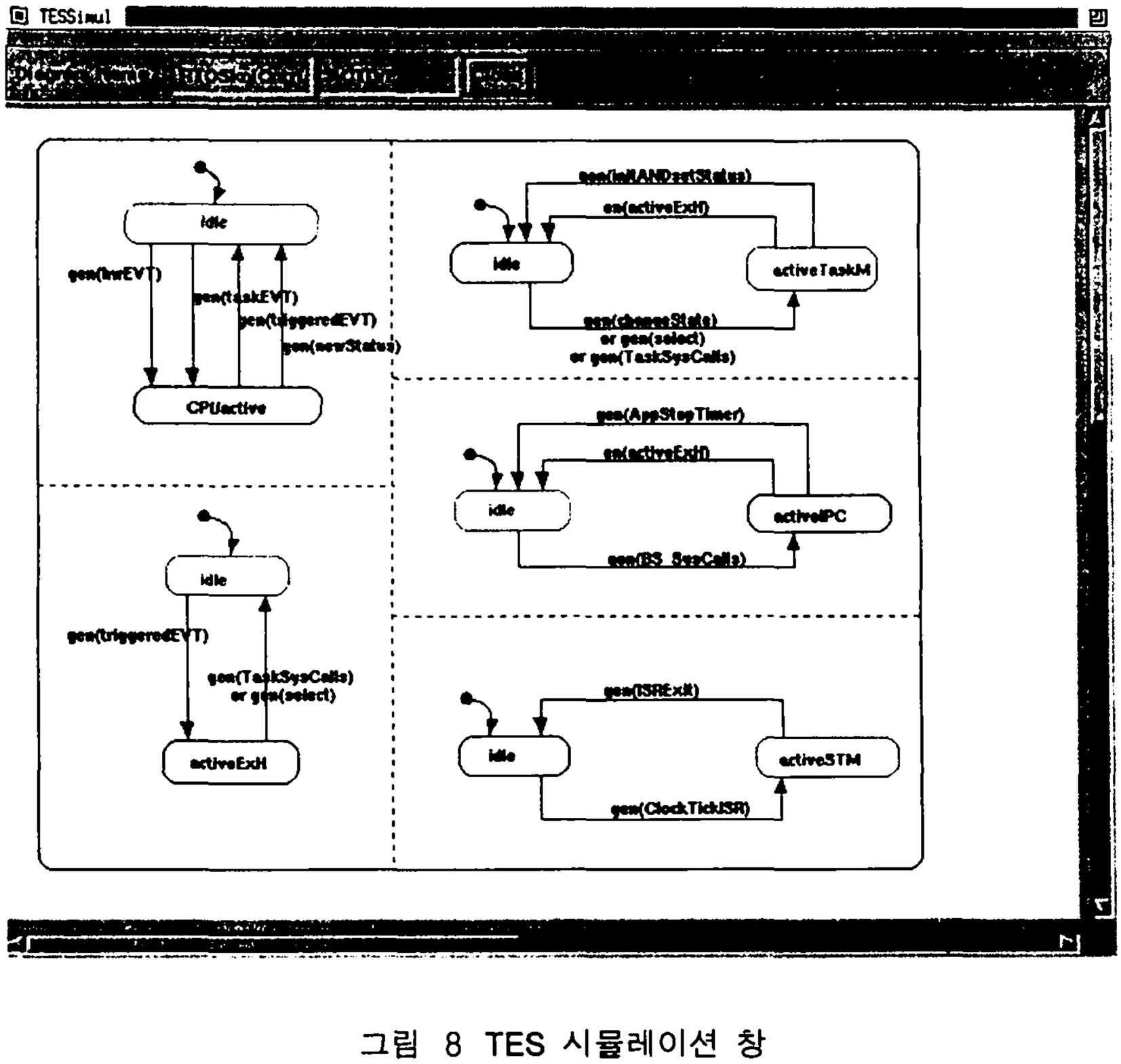


그림 8 TES 시뮬레이션 창

4 절 추계적인 배치형(Stochastic Batch) 시뮬레이터

ASADAL 시뮬레이션은 추계적인 배치형 시뮬레이션을 제공한다. 즉, 외부 입력

에 대한 특성을 명세할 수 있으며 프로세스들에 대한 서비스 특성을 특히 시간적인 특성을 명세하게 된다. 또한 상태의 변화도 시간과 연관해 이루어 지는 것이 많다. 이러한 추계적인 시뮬레이션에 의해 우리는 높은 신뢰도로 실제 이 시스템이 아주 많은 시간 동안 운영되었을 때 어떤 결과를 낳을 것인지를 예측할 수 있으며 이것이 추계적인 배치형 시뮬레이터의 한 가지 목표이다.

당해년도의 연구 결과로서 추계적인 배치형 시뮬레이션이 가능하게 되었다. 이 절에서는 이를 위해 필요한 ASADAL 명세 방법과 배치 시뮬레이션의 결과물, 그리고 이 결과를 분석해 주는 도구에 대한 연구 결과 등에 대해 설명하고자 한다.

1. 추계적인 배치 시뮬레이션을 위한 명세 방법

추계적인 배치 시뮬레이션은 많은 시간 동안 빠르게 이루어져야 하므로 사용자와 대화형으로 진행될 수 없다. 따라서 사용자 개입이 필요했던 많은 부분을 자동으로 시뮬레이션할 수 있도록 명세해 주는 일이 필요하다.

이것을 위한 명세 언어에는 프로세스 명세(Process Specification)와 시뮬레이션 드라이버 프로그램(Simulation Driver Program)이 있다.

□ 프로세스 명세

기본 프로세스(Primitive Process)는 간단한 기능을 수행한다. 그리고 이들에게는 시뮬레이션 묘사 언어(Simulation Description Language)라는 언어를 이용한 프로그램이 명세되는데 이 것이 프로세스 명세이다. 이 명세는 크게 프로세싱 시간(Processing Time)과 계산 알고리듬(Computation Algorithm)을 포함한다.

□ 시뮬레이션 드라이버 프로그램

시뮬레이션 드라이버는 RTET에 기술된 시나리오나 정해진 추계적인 모델에 근거하여 외부 개체들의 행위를 시스템의 시뮬레이션에 반영하는 역할을 수

행하는 시뮬레이터의 한 요소이다. 예를 들어 시스템이 필요로 하는 외부로부터의 사건이나 데이터는 이 것에 의해 발생되어진다. RTET 을 이용한 시뮬레이션의 경우 단지 가능한 하나의 시나리오에 대해서만 시뮬레이션만 행해질 수 있기 때문에 배치 시뮬레이션에는 적당하지 않다. 따라서 시뮬레이션 드라이버 프로그램이 존재하여 외부 개체의 행위에 대한 추계적인 모델을 만든다.

프로세스 명세와 시뮬레이션 드라이버 프로그램에 대해서는 지난 해 결과물로 제출한 바 있으니 참고하기 바란다. 당해년도에는 이를 직접 구현, 시뮬레이터와 통합, 운영되게 하였다.

2. 배치 시뮬레이션을 위한 소프트웨어 구현과 통합

비치 시뮬레이션을 하기 위해 사용자 대화형의 경우와는 달리 여러 소프트웨어 부분들이 추가로 시뮬레이터에 추가되었다.

추가된 것은 앞서 설명했던 시뮬레이션 드라이버로 정리될 수 있는데 이 시뮬레이션 드라이버는 외부 개체들의 행위를 시스템 시뮬레이션에 자동으로 반영하는 역할 뿐 아니라 시뮬레이션 수행 도중 어떤 상황이 되어 설정된 여러 형태의 멈춤점(Breakpoint)에 도달했을 때 시뮬레이션을 멈추어야 할 지를 결정, 시뮬레이션을 멈추고 사용자가 시스템 상태를 검사하고 그것을 고칠 수 있는 여지를 주어 사용자가 시뮬레이션에 참여할 수 있는 기회를 준다. 예를 들어 ASADAL 명세가 완전히 되지 않았을 때 이 기능을 이용하면 명시되지 않은 곳이 필요할 때 시뮬레이션을 멈추고 그 부분에서 일어남직한 일들을 사용자가 대신해 봄으로써 대강이나마 시뮬레이션을 해 볼 수 있다.

그럼 이제 DFD 의 시뮬레이션을 설명하도록 하겠다.

가. 개요

DFD 에 나타나는 internal process 들은 두 가지 종류로 나눌 수 있는데, 그 하나는 decompose 되어서 child DFD 를 가지는 것(derived process)이고, 다른 하나는 더 이상 decompose 되지 않고 아주 기본적인 기능이 할당되는 것(primitive process)이다. DFD 를 시뮬레이션할 때, 전자는 DerivedProcessor 라는 클래스의 오브젝트가 하나 생성되어 시뮬레이션하게되고, 후자는 PrimProcessor2 라는 클래스의 오브젝트가 하나 생성되어 시뮬레이션을 수행하게 된다. 또한 DFD 에는 이러한 internal process 이외에도 external process 도 있는데, 이들의 시뮬레이션을 위해서는 SimDriver 라는 클래스의 오브젝트를 생성한다. (뒤에서 보겠지만 SimDriver 클래스는 external process 를 시뮬레이션하는 기능 이외에도 breakpoint 를 관리하는 기능도 함께 제공한다.)

나. DerivedProcessor 클래스

derived process 들은 그 행동이 DFD 로 나타나 있기 때문에, 자신이 특별한 행동을 취하는 것이 아니라 자식 process 들을 관리해주는 역할만 한다. 즉, 구체적인 행동은 자식 process 들 중에 primitive process 에 의해 행해지고, derived process 는 그들 사이의 데이터 전달이나 activate, deactivate 에 대한 관리만 한다는 것이다. TES 의 각 state 는 DFD 의 하나 이상의 process 들을 activate, deactivate 시키도록 되어 있는데, 이때 TES 를 관리하는 controller 가 자신이 속해있는 DFD 에 해당하는 derived process 에게, 그 activate 와 deactivate 를 요청하게 된다. 또한, 데이터가 전달되는 방법은 다음과 같다. 어떤 primitive process 가 데이터를 만들어냈으면, 그것을 자신의 부모 process 에 알린다. (dataOutFromChild 함수를 이용) 그러면 부모 process 는 자신이 관리하는 DFD 에서 그 데이터의 목적지를 알아내고 그 목적지 process 에 그 데이터를 가져가라는 요청을 한다. (dataInToChild 함수를 이용)

다. PrimProcessor2 클래스

더 이상 decompose 되지 않는 primitive process 들은 아주 기본적인 기능을 수행하는 process 들인데, 이들이 어떤 식으로 시뮬레이션 되어야 하는지는 명세하는 사람이 Process-Spec 을 별도로 명시해 주어야 한다. Process-Spec 이란 어떤 입력 데이터가 왔을 때, 그것을 어떤 식으로 처리(process)해서 어떤 출력 데이터를 내보내야 하는지를 지정해주는 일종의 프로그램인데, PrimProcessor2 가 이를 해석하고 그에 따라 적절한 수행을 하게 된다. 그럼, Process-Spec 을 대략적으로 살펴보고, 그에 따라 PrimProcessor2 가 어떤 식으로 이를 해석하고 수행하는지를 살펴보도록 하겠다.

(1) Process-Spec

Process-Spec 은 크게 네 가지 부분으로 나눌 수가 있는데, 변수 선언부, 초기화부, 행동 명세부, 컨트롤 지정부 등이 그것이다. 변수 선언부는 Process-Spec 에서 임시로 쓸 지역 변수들을 선언하는 곳이고, 초기화부는 해당 process 가 activate 되어 새로운 일을 시작하기 전에 실행하는 명령들을 지정하는 곳이며, 행동 명세부는 Process-Spec 의 가장 핵심적인 부분으로서 어떤 입력 데이터에 대해서 어떤 일을 수행해야 하는 지를 지정한 곳이다. 그리고 마지막으로 컨트롤 지정부는 process 가 activate 될 때 이전에 하다가 중지한 일을 계속해서 할지(SUSPEND_RESUME), 아니면 그냥 무시하고 새로운 일을 시작할지(START_STOP)를 지정해주는 것이다. 그림 1 에 Process-Spec 의 한 예를 보였다.

(2) PrimProcessor2

우선 그 parent process 에 제공되는 인터페이스로는 생성자와 소멸자는 물론, activate, deactivate, 그리고 process 가 있다.

- PrimProcessor2(Supervisor *supervisor, STRING name, DerivedProcessor *parentProc, STRING progName)

```
: Processor(supervisor, name, parentProc)
{
        _suspendedComp = NULL;
        _activeComp = NULL;
        _ex_control = START_STOP;
        _initialize(progName);
}

void PrimProcessor2 :: _initialize(char *scp_name)
{
        char*     str;

        get_var(scp_name, &str);
        _compileVarSec(STRING(str));

        get_init(scp_name, &str);
        _setInitStatements(STRING(str));

        get_behavior(scp_name, &str);
        _compileCompSec(STRING(str));

        _setExecutionControl(get_control(scp_name));
}
```

우선 Processor 의 파생 클래스로서 supervisor, name, parentProc 을 Process 클래스의 생성자에 전달해 주고는 progName 을 이용해 자기 자신을 초기화한다. 즉, 가장 먼저 get_var 를 호출해서 Process-Spec 의 변수 선언부를 얻는다. 그리고는 _compileVarSec 함수를 호출해서 변수 선언부를 분석(parsing)해서 변수들에 대한 정보를 얻고 각 변수들을 Data 클래스의 오브젝트로 만들어 _localDataList 에 저장한다. 두 번째로는 get_init 를 호출해서 Process-Spec 의 초기화부를 얻고 그것을 _initStatements 에 문자열로 저장해둔다. 세 번째로는 get_behavior 를 호출하여 Process-Spec 의 행동 명세부를 얻고 _compileCompSec

함수를 호출해서 각 행동 명세에 대한 정보를 얻어낸다. 그리고는 그 실행 제어기를 설정한다.

프로세스 명세(Process Specification)로부터 정보를 얻어내는 부분은 후에 설명할 것인데 Lex 와 Yacc 도구를 이용하여 개발하였다.

□ void activate()

```
{
    Computation*    comp;

    _state = ACTIVE;
    if (_ex_control == START_STOP) {
        _executeInitSec();
        if (comp = _getTriggeredComp())
            _activeComp = comp;
        else _activeComp = NULL;
    }
    else { // in case of RESUME_SUSPEND
        if (_suspendedComp)
            _activeComp = _suspendedComp;
        else if (comp = _getTriggeredComp())
            _activeComp = comp;
        else _activeComp = NULL;
    }
}
```

PrimProcessor2 가 activate 될 때에 필요한 처리를 실행한다. 이때 두 가지 경우가 있는데, 실행 컨트롤이 START_STOP 이냐 아니면 SUSPEND_RESUME 이냐에 따라 달라진다. 실행 컨트롤이 START_STOP 인 경우는 먼저 초기화부를 실행한 다음, 실행 조건이 만족된 Computation 이 있는 지를 검사해서 있으면 그 Computation 을 _activeComp 에 설정한다. 그리고, 실행 컨트롤이

SUSPEND_RESUME 이라면 _suspendedComp 를 검사해서 하다가 중지된 Computation 이 있으면 그것을 _activeComp 로 설정하고 그렇지 않다면 다시 실행 조건이 만족된 Computation 이 있는지를 검사해서 있다면 그것을 _activeComp 로 설정한다.

□ void deactivate()

```
{
    if (_ex_control == START_STOP) {
        _activeComp = NULL;
        _suspendedComp = NULL;
        _state = INACTIVE;
        }
    else { // in case of RESUME_SUSPEND
        _suspendedComp = _activeComp;
        _state = INACTIVE;
    }
}
```

deactivate 는 그 프로세스의 프로세싱을 멈추는 일을 하는데, Control 이 START-STOP 방식인지, RESUME-SUSPEND 방식인지 보고 RESUME-SUSPEND 인 경우는 그 때의 상태를 저장하고 멈춘다. 이 저장된 정보는 물린 후에 활성화될 때 그 상태로부터 계속 시작하기 위한 것이다.

□ void process(DTIME t)

실제로 프로세싱을 행하는 부분이다. 프로세싱은 다시 그 프로세스중 활성화된 컴퓨테이션이 있을 경우 그 컴퓨테이션의 process()를 호출, 프로세싱을 하게 한다. 그리고 그 되돌림 값(Return Value)가 COMP_PROCESSED 로서 프로세싱이 끝났음을 알리면 Trigger 된 다음 컴퓨테이션이 있는지 판단하여 다음 프로세싱할 컴퓨테이션을 설정한다.

컴퓨테이션을 행하는 것도 컴퓨테이션 섹션의 명세를 다음에 설명할 Yacc 문법 해석기의 Action 과 Inherited Attribute 등을 이용하여 실행함으로써 만들어졌다.

(3) 프로세스 명세의 해석

프로세스 명세를 해석하기 위한 방법으로 Lexical Analyzer 로는 Lex(컴파일러는 Flex 를 사용하였다), 문법 해석기로는 Yacc(컴파일러로는 Bison 을사용하였다)을 이용, Lexical 규칙과 문법 규칙을 작성하고, 필요에 따라 문법 해석 도중 기능을 행하도록 Action 과 Inherited Attribute 을 기술하였다.

다음이 Lex 으로 구성한 Lexical 규칙이다.

```
Alpha           [A-Za-z]
digit           [0-9]
optsign         [+\-]?
Integer         {digit}+
frac            \.{digit}+
delim           [ \t]
ws              {delim}+
numreal         {digit}+{frac}?
Id              {alpha}({alpha}|{digit}|"_")*
string          \"([^']|"'")*\"
%%
"("             { return (LPAR) ; }
")"             { return (RPAR) ; }
"or"            { return (OR) ; }
"and"           { return (AND) ; }
"in"            { return (IN) ; }
"not"           { return (NOT) ; }
"if"            { return (IF) ; }
"then"          { return (THEN) ; }
"else"          { return (ELSE) ; }
```

```
"while"          { return (WHILE) ; }
"do"             { return (DO) ; }
"On"             { return (ON) ; }
"Available"      { return (AVAILABLE) ; }
"Begin"          { return (SBEGIN) ; }
"End"            { return (SEND) ; }
"TIME"           { return (TIMELABEL); }
"Gen!"           { return (GEN) ; }
"Wstore!"        { return (WSTORE) ; }
"Rstore!"        { return (RSTORE); }
"True!"          { return (TRUE); }
"False!"         { return (FALSE); }
"Rand"           { return (RAND) ; }
"RandExp"        { return (RANDEXP); }
"RandUniR"       { return (RANDUNIR); }
"RandNor"        { return (RANDNOR); }
"RandPoi"        { return (RANDPOI); }
"RandUni"        { return (RANDUNI); }
"RandBin"        { return (RANDBIN); }
"Type"           { return (TYPE); }
"integer"        { return (INTT); }
"boolean"        { return (BOOLT); }
"real"           { return (REALT); }
"string"         { return (STRT); }
"record"         { return (RECORD); }
"oneof"          { return (ONEOF); }
"="              { return (EQUAL) ; }
":="             { return (ASSIGN) ; }
"!="             { return (NOTEQUAL) ; }
"+"              { return (PLUS) ; }
"-"              { return (MINUS) ; }
"*"              { return (MULT) ; }
"/"              { return (DIV) ; }
";"              { return (SEMI) ; }
```

```
","             { return (COMMA) ; }
":"             { return (COLON) ; }
{id}            { strcpy(VARIval.sval,VARtext) ; return (IDTOKEN) ; }
{string}        { strncpy(VARIval.sval, VARtext+1, strlen(VARtext)-2) ;
                  VARIval.sval[strlen(VARtext)-1] = '\0' ;
                  return (STRCONST) ; }
{integer}       { VARIval.ival = atoi(VARtext) ;
                  return (NUMBER) ; }
{ws}            ;
```

위는 "%%"으로 두 부분으로 나뉘어 있는데 각각 정의 부분(Definition Section)과 규칙(Rules Section)이라 불린다. 정의 부분에서 각각의 규칙에 맞았을 때 "{}"안에 있는 C 코드를 실행하는데 여기서 "return()"를 이용, 인수로 넘기는 것이 토큰으로서 문법 해석기가 받아들이게 된다.

Yacc 도 정의 부분과 규칙 부분이 있는데 문법 해석기를 위한 Yacc 명세 규칙 부분에서 규칙만 보인 것이 다음과 같다.

```
computations :   computation computations
                 | computation
                 ;
computation :    trigger_expr process_time body
                 ;
trigger_expr :   ON triggers DO
                 ;
triggers :       trigger AND triggers
                 | trigger
                 ;
trigger :        AVAILABLE LPAR IDTOKEN RPAR
                 ;
process_time :   TIMELABEL COLON time_val
                 ;
time_val :       NUMBER
```

```
                    | REALNUM
                    | rand_fun
                    ;
body :              SBEGIN stmt_list SEMI SEND
                    ;
stmt_list :         stmt
                    | stmt_list SEMI stmt
                    ;
stmt :              assign
                    | if_stmt
                    | while_stmt
                    | gen_fnc
                    | wstr_fnc
                    | rstr_fnc
                    | tr_fnc
                    | fs_fnc
                    ;
assign :            IDTOKEN ASSIGN expr
                    ;
if_stmt :           IF LPAR condition RPAR THEN stmt ELSE stmt
                    | IF LPAR condition RPAR THEN stmt
                    ;
while_stmt :        WHILE condition DO stmt
                    ;
gen_fnc :           GEN LPAR IDTOKEN RPAR
                    ;
wstr_fnc :          WSTORE LPAR IDTOKEN RPAR
                    ;
rstr_fnc :          RSTORE LPAR IDTOKEN RPAR
                    ;
tr_fnc :            TRUE LPAR IDTOKEN RPAR
                    ;
fs_fnc :            FALSE LPAR IDTOKEN RPAR
                    ;
```

```
condition :     cterm morecterms
                ;
morecterms :    AND cterm morecterms
                | OR cterm morecterms
                |
                ;
cterm :         factor
                | NOT factor
                | IDTOKEN EQUAL STRCONST
                | IDTOKEN NOTEQUAL STRCONST
                | IDTOKEN EQUAL expr
                | IDTOKEN NOTEQUAL expr
                | IN LPAR IDTOKEN RPAR
                ;
expr :          term moreterms
                ;
moreterms :     PLUS term moreterms
                | MINUS term moreterms
                |
                ;
term :          factor morefactors
                ;
morefactors :   MULT factor morefactors
                | DIV factor morefactors
                |
                ;
factor :        LPAR expr RPAR
                | IDTOKEN
                | rand_fun
                | num_val
                ;
num_val :       REALNUM
                | NUMBER
                ;
```

```
rand_fun :      RAND LPAR num_val RPAR
                | RANDEXP LPAR num_val RPAR
                | RANDUNIR LPAR num_val COMMA num_val RPAR
                | RANDNOR LPAR num_val COMMA num_val RPAR
                | RANDPOI LPAR num_val RPAR
                | RANDUNI LPAR num_val COMMA num_val RPAR
                | RANDBIN LPAR num_val COMMA num_val RPAR
                ;
```

라. SimDriver 클래스

앞에서 잠시 언급되었듯이 SimDriver는 외부 개체들(External Entity)의 행동을 제어함과 동시에 시뮬레이션하는 사용자가 특정 조건이나 상황에 설정한 breakpoint를 관리하면서 breakpoint가 만족되는 경우에 supervisor에게 시뮬레이션을 멈추게 하고 컨트롤을 사용자에게 넘기게 하는 역할을 한다. 외부 개체의 행동을 정의한 것과 breakpoint를 설정하는 것을 합쳐서 시뮬레이션 드라이버 프로그램(Simulation Driver Program : SDP)이라고 하는데 다음에 그 형식을 설명하였다.

(1) 시뮬레이션 드라이버 프로그램

```
Driver    prog_name

[Variable declaration]
[Initiation section]
[External Behavior section]
[Breakpoint section]
```

시뮬레이션 드라이버 프로그램의 구조는 위에서 보는 바와 같이 지역 변수 선언부,. 초기화부, 외부 행동 선언부, 멈춤점 선언부 등으로 구분된다. 여기서 지역 변수 선언부와 초기화부는 앞에서 설명한 Process-Spec과 동일하므로 반복해서 설명하지 않고, 외부 행동 선언부와 멈춤점 선언부에 대해서만 살펴보도록 하겠다. 우선

외부 행동 선언부는 그 이름 그대로 외부 개체들의 행동이 어떤 식으로 나타나는지를 정의해놓는 곳인데 두 가지 종류로 나눌 수 있다. 하나는 반응 행동으로서 어떤 사건(event)의 발생이나 조건(condition)의 만족에 의해 유도되는 행동이고 다른 하나는 주기적 행동으로서 다른 개체의 행동에 관계없이 일정 간격으로 데이터를 생성, 발생시키는 행동이다. 자연히 두 행동은 지정하는 방법이 다른데, 다음에 그 차이를 보였다.

□ 반응 행동의 정의

```
DEFINE [ <behavior_name> : ] <event> | <condition> Do
Begin
        <statement_list>
End
```

□ 주기적 행동의 정의

```
DEFINE [ <behavior_name> : ] <condition> | Every <time_expresision> Do
Begin
        <statement_list>
End
```

다음은 breakpoint 의 설정에 관해 설명하겠다. breakpoint 는 시뮬레이션 도중 어떤 특별한 상황이 발생했을 때 미리 지정해두었던 명령들을 실행하게 하는 것으로서 breakpoint 가 만족되게 되면 그 시점에서 지정된 명령들이 실행되고 경우에 따라서는 사용자에게 컨트롤을 넘길 수도 있는 것이다. 따라서 이를 잘 이용하면 완성된 시스템 모델을 디버깅할 때나 완성되지 않은 시스템 모델을 부분적으로 시뮬레이션 해볼 수도 있다. 다음에 그 설정 형식을 보였다.

□ Breakpoint 의 설정

```
Breakpoint <trigger_expression> Do
 Begin
        <statement_list>
 End
```

(2) SimDriver

SimDriver 의 public 멤버 함수로는 생성자와 소멸자, 그리고 initialize, processEX, processBP 등이 있다.

□ void initialize(STRING prog_name)

```
{
        char* prog;

        prog =  getDrivProg(prog_name);
        parseDriver(prog);
}
```

PrimProcessor2 가 시뮬레이션을 위해서 Process-Spec 을 미리 분석(parsing)하여 자신을 초기화했던 것과 마찬가지로 SimDriver 도 생성시킨 뒤에 initialize 멤버 함수를 호출하여 초기화를 시켜주어야 한다. 시뮬레이션 드라이버 프로그램에도 이름이 있는데 이를 initialize 멤버 함수에 인자로 넘겨주어야 한다. getDrivProg 함수는 드라이버 프로그램 이름에 해당하는 드라이버 프로그램을 찾아서 문자열로서 돌려준다. 그리고 그렇게 받은 프로그램을 parseDriver 를 이용하여 분석(parsing)해서 적절한 초기화를 수행하게 되는 것이다.

□ void processEX(DTIME t)

```
{
        QueueIterator<ExEvent*>         schedExIter(&_listOfSchExEvent);
```

```
QueueIterator<ExEvent*>          totalExIter(&_totalExEventlist);
ExEvent *ev;

if (!_listOfSchExEvent.empty()) {
        ev = schedExIter.first();
        if (ev->process(t) == EX_TERMINATED)
            _listOfSchExEvent.remove(ev);
        while (!schedExIter.atEnd()) {
            ev = schedExIter.next();
            if (ev->process(t) == EX_TERMINATED)
                _listOfSchExEvent.remove(ev);
        }
}

if (!_totalExEventlist.empty()) {
        ev = totalExIter.first();
        if (ev->isTriggered())
            _listOfSchExEvent.insert(ev);
        while (!totalExIter.atEnd()) {
            ev = totalExIter.next();
            if (ev->isTriggered())
                _listOfSchExEvent.insert(ev);
        }
    }
```

외부 개체들의 행동을 시뮬레이션하기 위한 함수로서_totalExEventlist 는 하나의 시뮬레이션 모델에서 발생할 수 있는 모든 외부 사건들의 리스트이다. 이 리스트에 들어있는 외부 사건들 중에서 그 발생이 예약된 사건들은 _listOfSchExEvent 에 넣어두고 다음에 발생될 시간이 되면 그 사건을 _listOfSchExEvent 에서 제거한다.

□ void processBP()

```
{
        QueueIterator<BreakPoint *> breakPntIter(&_totalBreakPointList);
        BreakPoint *bp;

        if (_totalBreakPointList.empty()) return;
        bp = breakPntIter.first();
        if (bp->isTriggered())
                bp->execute();
        while (!breakPntIter.atEnd()) {
                bp = breakPntIter.next();
                if (bp->isTriggered())
                        bp->execute();
        }
}
```

시뮬레이션 드라이버 프로그램에서 지정한 breakpoint 를 처리하기 위한 함수이다. 드라이버 프로그램이 지정한 breakpoint 들을 리스트로 가지고 있다가 각 breakpoint 들이 활성화되었는지를 검사해서 활성화되었으면 실행시킨다.

3. 배치 시뮬레이션의 결과물

배치 시뮬레이션을 하면 그 결과로 먼저 완전한 시간에 따른 시스템 상황 변화에 대한 기록 파일(Log File)을 얻을 수 있다. 이 파일엔 다음과 같은 내용들이 포함된다.

- □ 사건의 발생과 그 시각
- □ 프로세스의 활성화, 비활성화
- □ 데이터, 제어 시그널의 발생, 그 값
- □ 데이터의 사용

□ 조건 값의 변화

□ 어떠한 상태에 들어간 것, 나간 것

이들 내용들은 모두 그것이 발생한 시각과 함께 기록된다.

4. 시뮬레이션 결과 분석 도구

이 절에서는 계획되고 있는 시뮬레이션의 결과 분석 도구에 대해 설명하고자 한다. 추계적인 배치형 시뮬레이션을 하는 경우뿐 아니라 사용자 대화형 시뮬레이션을 함에 있어서도 단지 시뮬레이션을 한다는 것에는 의미가 없다. 시뮬레이션을 통해 얻어야 할 목표가 있을 것이며 이를 위해서는 시뮬레이션 도중이나 후에 그것을 의미 있는 형태로 보거나 분석할 수 있는 도구가 반드시 필요하다.

당해년도에는 이를 위해 먼저 어떠한 분석을 배치 시뮬레이션과 함께 행할 수 있겠는지, 어떠한 분석을 사용자가 요구하겠는 지에 대한 연구가 다방면으로 진행되었다.

가. 시뮬레이션 결과 분석

시뮬레이션이란 일정 시간 동안 시스템의 모형을 실행시켜 보고 그 결과들을 바탕으로 시스템의 궁극적 행동(Long-Run Behavior)을 예측해 보는 것이다. 그러므로, 시뮬레이션 결과를 분석하여 의미를 추출해 내지 않고 끝낸다면 그 시뮬레이션은 별로 유용하지 않을 것이다. 그래서, ASADAL 시뮬레이터는 시뮬레이션 결과를 통계적이고 추계적으로 분석해 주고 그 의미를 그래픽컬하게 보여 주는 도구를 제공하려고 한다.

이 분석 도구는 시뮬레이션 도중에는 데이터 값, 프로세서의 서비스 시간, 데이터가 큐에서 기다리는 시간 등의 평균, 분산의 변화를 그래프를 통해 동적으로 보여 준다. 그리고, 시뮬레이션이 끝나면 수행된 동안 나타난 시스템의 결과를 바탕으

로 예측된 최종의 결과를 보여 줄 것이다.

이 절에서는 앞으로 제공될 예정인 결과 분석 도구가 어떤 것들을 제공할 것인지에 대해서 설명한다.

나. 분석 내용

ASADAL 시뮬레이터에서 TES 명세와 DFD 명세의 시뮬레이션은 TES 명세에 의해 시스템의 상태(State)의 전이가 일어나며 그 때 각 상태에 명시되어 있는 DFD 명세의 프로세스가 활성화 되어 정해진 일을 하고 비활성화 되면서 진행된다.

시뮬레이션 도중 각 데이터나 제어 신호는 특정 확률 분포 함수에 따라 프로세스가 가지고 있는 큐로 들어오며, 큐에 쌓여 있던 데이터는 기본 프로세스가 수행될 때마다 하나씩 소비된다. 큐는 기본 프로세스의 특성에 따라 길이가 다르며, First-come First-served 방식이다. 기본 프로세스는 서비스를 받으러 들어 오는 데이터에 대해서 서비스를 제공한 후 서비스가 끝나면 내보내며 프로세스가 데이터를 서비스하는 시간도 특정 확률 분포를 따른다.

이렇게 시뮬레이션이 진행되는 동안 어떤 데이터가 언제 어떤 프로세스의 큐에 들어갔고 언제 프로세스에 의해 서비스 받았으며 언제 프로세스 밖으로 나왔는지 그 값이 무엇인지를 기록하는 Output File 을 만든다.

그 Output File 로부터 다음과 같은 것들을 분석할 수 있을 것이다.

□ 시간 제약 검사(Time Constraint Checking)

RTET 명세에는 어떤 사건이 일어나고 그 다음에 어떤 사건은 얼마의 시간 안에 일어나야 한다는 시간 제약 등을 명세할 수 있다. 그래서, 분석 도구는 RTET 명세에 나타나는 두 사건 사이의 시간 제약들이 시뮬레이션 도중에 잘 맞는지를 계산해 준다. 그리고, 시뮬레이션이 끝나고 나서는 시간 제약을 지

키는 데 대한 신뢰도를 계산해서 보여 준다.

□ 데이터 값 분석

각 데이터나 제어 신호에 대해서 현재 어떤 값이 들어오고 있는지를 보여 준다. 그리고, 시뮬레이션 도중 현재까지의 값의 평균과 분산의 동적인 변화를 그래프를 통해 보여 준다.

□ 큐 길이 분석

큐에서 서비스를 기다리는 데이터의 개수의 평균과 분산을 구하여 그래프로 보여 준다.

□ 서비스 시간 분석

각 데이터에 대해서 프로세스에 들어가서 서비스를 받고 프로세스를 나가는 데 걸리는 시간의 평균과 분산을 계산하여 그래프로 동적인 변화를 보여 준다.

□ 기다리는 시간 분석

각 데이터에 대해서 큐에 들어와서 프로세스에 들어가기 전까지 큐에서 머무는 시간의 평균과 분산을 계산하여 그래프로 동적인 변화를 보여 준다.

□ 상태의 활성 비율

시뮬레이션 동안 각 상태에 있는 비율이 얼마나 되는지를 계산하여 보여 준다. 그리고, 시뮬레이션 결과를 바탕으로 상태의 전이 확률을 계산하고 Markov Chain 을 이용하여 궁극적으로 어떤 상태에 있을 확률이 얼마나 되는가를 계산한다.

□ 프로세스의 Busy/Idle Proportion

프로세스가 시뮬레이션 도중 얼마나 바빴는가와 얼마나 놀았는지를 계산하여 보여 준다. 이 결과로부터 프로세스가 거의 놀지 않고 서비스를 했다면 프로세스를 하나 늘리는 것을 고려해 볼 수 있다.

5 절 ASADAL CASE 도구 기능의 확장

ASADAL CASE 도구는 이번에 상용 데이터베이스 소프트웨어인 UniSQL 을 사용하던 것을 필요한 연구 기관이나 기업 등에 배포하기 쉽고 사용상의 편의를 돕기 위해 무상의 공공 데이터베이스로 교체하였고, 다른 도구들과의 데이터 교환 등의 상호 작용을 돕기 위해 네트웍 인터페이스를 마련하였다.

1. 데이터베이스

이번에 ASADAL 시뮬레이터 및 그림 도구는 이전 버전에서 사용하던 UniSQL 데신 무상 소프트웨어를 이용한 데이터베이스 부분을 자체로 개발하였다. ASADAL 의 그림 도구 및 시뮬레이터는 기본적으로 많은 회사나 연구소에 무상으로 배포, 실제 현장에 적용하거나 다른 연구에 응용되도록 노력하고, 또 그럼으로써 외부로부터의 아이디어나 조언들을 얻도록 제작되었을 뿐 아니라 외부에서 잦은 데모가 있는데, 여러 곳에서 설치한 결과 시스템 사용을 위해 UniSQL 을 사야 하는 문제점이 큼을 알았다. 또한 데모시에도 직접 UniSQL 이 설치된 웍스테이션을 가져가야 하는 문제점이 대두되었다. 따라서 이번 데이터베이스의 개발, ASADAL CASE 도구와의 통합은 우리의 이러한 활동을 더욱 촉진시켜줄 것으로 기대된다.

이 데이터베이스가 어떻게 구현되어 있으며 그 운영 방식에 대해 성명한 후에 기존 UniSQL 을 사용했을 때와 비교를 하도록 하겠다.

가. 데이터베이스의 구현

□ 이번 데이터베이스 부분의 두 번째 버전에서는 기존에 UniSQL 을 이용했던 것을 자체 내의 간단한 데이터베이스로 바꾸자는 것이 목적이었으므로 UniSQL 의 인터페이스를 그대로 유지하면서 같은 기능을 하도록 바꾸었다.

ASADAL CASE 도구의 설계 철학상 데이터베이스 부분이 바뀌기 쉬운 부분으로 파악, 철저히 추상화(Abstraction)될 수 있게 초기에 설계되었으므로 이번 작업은 CASE 도구의 다른 부분은 건드리는 일 없이 데이터베이스를 숨기고(Information Hiding)있던 모듈만 바꾸어줌으로써 완료되었다.

□ 실제로 파일에 읽고 쓰는 것은 <ndbm.h>라는 라이브러리를 이용하여 구현을 했으므로, 클래스 스키마(Schema) 관리에 중점을 두었다.

□ 여러 가지 테스트를 하는 과정에서 ndbm 의 파일 크기의 한정이 있어서 다시 "gdbm.h"로 바꾸었다. ndbm 과 gdbm 은 인터페이스가 거의 같기 때문에 바꾸는데는 5 분이면 되었다.

□ database.h, database.c 파일을 보면 쉽게 이해가 갈 것이다.

나. 스키마의 정리 및 관리

```
struct Attribute {
        int type;
        char name[MAX_STR];
        int index;
};

class Schema {
        char classname[MAX_STR];
        Attribute *attrList;
        ....
public:
```

```
};
```

- 어떤 클래스의 스키마가 정의가 될 때에 그 클래스의 attribute 들이 attrList 에 의해서 관리가 된다.

- 위의 구조체 Attribute 로 보면 type 은 그것의 타입을, name 의 이름을 저장하고 있다. 그리고, index 는 그것이 실제로 저장될 곳의 번지수라고 생각을 하면 된다.

- 하나의 instance 는 그것의 스키마를 찾아서 각 attribute 들이 저장될 곳의 index 를 참조하여 읽고 쓰게 된다.

- sequence 타입은 그 크기가 유동적이므로 따로 관리하여 읽고 쓰게 된다.

(1) 스키마의 정의

예를 들어, 아래의 예제 클래스가 있다고 생각해 보자.

```
extern DataManager data_mngr;


class Cls {
public:
    int i_val;
    double d_val;
    char s_val[128];
    list<int> l_val;
};
```

스키마의 정의는 아래와 같이 한다.

```
data_mngr.CreateClass ("Cls");
```

```
data_mngr.AddAttribute ("i_val", "integer", 1);
data_mngr.AddAttribute ("d_val", "double", 0 );
data_mngr.AddAttribute ("s_val", "string", 0 );
data_mngr.AddAttribute ("l_val", "sequence", 0 );
data_mngr.StoreClass ();
```

이렇게 정의를 하면 attrList가 그림 9와 같이 만들어 진다.

AttrList :

type →	sizeof(int)	sizeof(double)	MAX_STR(=128)	SEQUENCE(=0)
name →	"i_val"	"d_val"	"s_val"	"l_val"
index →	0	4	12	140

그림 9 Attribute의 관리

(2) 데이터의 저장

```
Cls aaa;

aaa.i_val = 14;
aaa.d_val = 3.14;
aaa.s_val = "Rambo";
```

이 예제에서 클래스 Cls의 한 instance aaa를 저장하면 파일에 그림 10과 같이 된다.

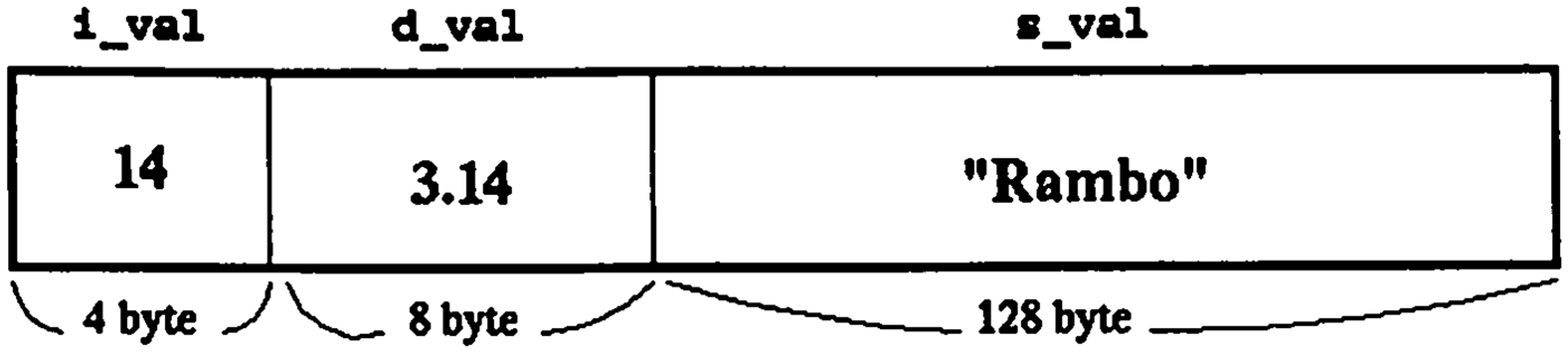


그림 10 Instance의 저장

그림 10에서 보듯이 실제로 한 instance는 파일에서 하나의 데이터 블럭이다. 그래서, 그 instance를 읽고 쓸 때에는 자신의 클래스 스키마로부터 각 attribute가 그 데이터 블럭에서 위치하고 있는 위치를 읽어서 그 곳을 참조한다. 예제에서는 그림 9의 attrList에서 i_val에 있는 곳은 index가 0이고 integer이므로 4 바이트, d_val은 index가 4이고 double이므로 8 바이트, s_val은 index가 12이고 string이므로 128 바이트씩을 차지하고 있다.

다. 생성되는 파일들

- gdbm에 의하여 생성되는 파일은 .dir, .pag의 확장자가 붙는다.
- ***db_name*.sch** - 실행한 프로그램에서 생성한 모든 클래스들의 스키마가 저장이 된다.
- ***db_name*.attr** - 클래스의 attrList들이 이곳에 저장이 된다.
- ***clas_name*.obj** - 각 클래스의 instance들이 저장이 된다.
- ***class_name*.seq, *class_name*.seqc** - 각 클래스의 instance들의 sequence 타입의 attribute들이 저장이 된다.

라. UniSQL을 사용했을 때와의 비교

만들어진 데이터베이스를 이용하여 실험해 본 결과, UniSQL을 사용했을 때보다

오히려 엄청난 속도 향상을 느낄 수 있었다. 처음 Schema 를 만드는 단계에서 10 초 정도 걸리던 것을 1 초 내에 해결하였으며 데이터의 저장과 읽어오기도 2 배 가량 빠른 모습을 보였다. 이 결과는 아마도 ASADAL CASE 도구에서 저장하는 데이터의 양이 그다지 크지 않은 것이기 때문에 부딪힘(Collision)이 별로 없이 빠른 찾기(Search)와 읽기가 가능한 해싱(Hashing) 방식을 새 데이터베이스가 사용하기 때문인 것으로 생각된다.

2. 다른 도구와의 인터페이스

ASADAL 시뮬레이터는 ASADAL 명세의 단독 시뮬레이션뿐 아니라 다른 시뮬레이터나 분석, 시각화(Visualization) 도구들과 쉽게 동시 작업을 수행할 수 있게 설계되었다. 즉, ASADAL 시뮬레이션을 통해 제어기의 시뮬레이션을 하는 것과 동시에 외부 개체들에 대한 시뮬레이션을 다른 시뮬레이터나 추계적인 시뮬레이터 등을 이용하여 하면서 이들로부터 제어기로의 입력을 받아들일 수 있고, 제어기의 출력을 이들 도구에 전해줄 수 있게 설계되었다. 즉, 원하면 시뮬레이션 도중에 얼마든지 ASADAL 시뮬레이터에 값을 전달하거나 값을 가지고 나오는 것이 가능하다.

이것은 실제로 이번 결과물인 AGC 데모에 사용되었는데 이 데모에서는 제어기 외부 환경에 대한 시뮬레이션은 UCI(University of California, Irvine)에서 행하였고 이것에서 제어기의 입력을 받아들여 제어 값을 계산, 이것을 다시 외부 개체에 전하는 식으로 실제 환경과 그 제어기에 대한 실시간 시뮬레이션을 실시하였다.

그뿐 아니라 이러한 방식은 ASADAL 명세로 제어기뿐 아니라 실제 객체들에 대한 명세를 하여 동시에 시뮬레이션하면서 ASADAL 시뮬레이터간의 데이터, 제어 시그널의 흐름을 통해 전체를 시뮬레이션할 수 있는 기반을 마련해 준다. 시스템이 거대해지고 더욱 더 계산의 양이 많아지면 실시간 시뮬레이션을 하기 위해 시스템을 여러 개로 나누고 이것을 여러 대의 컴퓨터에 나누어 시뮬레이션하는 것이 필수

가 될것이므로 이러한 기능은 반드시 필요할 것으로 생각된다. 결국 RTET 명세에서 다루고 있는 제어기를 포함한 각 시스템 개체에 대한 ASADAL 명세를 하고 이들을 여러 대의 컴퓨터 상에서 동시에 시뮬레이션할 수 있을 것으로 기대된다. 또한, 이것은 차후 개발될 동적 시각화(Dynamic Visualization) 도구와 연동하면서 실제 환경과 유사하게 보여주는 데 도움이 될 것으로 생각된다.

6 절 정형적인 실시간 성질 검증 방법(ASADAL/PROVER)

만약 잘못 명세될 경우 심각한 인명 손실이나 재산 손실을 가져오게 될 많은 분야에 실시간 소프트웨어 시스템들이 적용되고 있다. 따라서, 이러한 실시간 시스템은 정형적 방법에 의해 분석할 필요성이 그만큼 증가한 것이다.

ASADAL/PROVER(이하 PROVER)는 그림 11에서처럼 명세 분석을 위해 두 가지 행위를 갖는다. 하나는 TES 명세가 자신에게 요구되는 실시간 성질을 만족하는지 검증하는 것이고, 다른 하나는 외부적 관점에서 명세된 시스템의 외부 행동(RTET)과 내부적 관점에서 명세된 시스템의 내부 행동(TES) 및 기능(DFD) 간의 일관성을 검증하는 것이다.

본 절에서는 위에서 설명한 두 가지 분석 행위를 지원하는 PROVER 의 정형적 검증 방법을 소개한다. 이 중, 특히 실시간 성질의 정형적 분석 방법을 상세히 소개한다. 먼저, 시스템에서 검증되어야 할 시스템의 실시간 성질과 또, 검증을 위해 요구되는 시스템의 공정성에 대해 알아 본다. 그리고, 본 보고서에서는 실시간 성질의 검증을 도달 가능성 분석 방법을 통해 설명하므로, 이에 사용되는 도달 가능성 그래프의 설명과, 끝으로 이 도달 가능 그래프를 바탕으로 만들어진 실시간 성질들의 알고리즘을 각각 절로 나누어 설명할 것이다.

이어지는 1. ASADAL/PROVER 의 정형적 분석 방법에서는 PROVER 에서 제공하

는 정형적 분석 방법을 소개한다.

1. ASADAL/PROVER 의 정형적 분석 방법

PROVER 는 명세 분석을 위해 시스템 상태의 도달 가능성을 분석하는 방법과 RTTL 논리 검증기를 이용한 방법 등의 두 가지 방법으로 수행하며, 다음의 항에서 자세히 설명한다.

□ 도달 가능성 분석 방법은 내부적 관점에서 명세된 시스템의 내부 행동(TES)을 분석하여 시스템이 도달할 수 있는 모든 상태를 나타내는 도달 가능성 그래프를 생성하고 이를 바탕으로 몇 가지 정형적으로 검증된 알고리즘을 적용해 실시간 성질을 검증한다. 그림 11는 이 방법의 개략적인 모습을 보여준다. 이 방법은 검증 과정에서 오류가 발생하면 그 경우를 분석자가 직접 오류 발생 과정을 도달 가능성 그래프를 통해 추적해 볼 수 있으므로 시스템의 오류의 이해와 수정에 도움을 준다.

시스템의 외부적 관점에서 명세된 외부 행동과 내부적 관점에서 명세된 내부 행동 및 기능 간의 일관성을 검증하는 것은 도달 가능성 그래프 생성시에 내부 행동 뿐만이 아니라 그 기능도 함께 분석하면 가능하지만 현재 연구 중에 있다.

□ RTTL 논리 검증기를 이용하는 방법은 ASADAL/SPEC 에 의해 생성된 RTET, TES, DFD 명세를 모두 RTTL 로 변환하여, 이 RTTL 명세를 대상으로 RTTL 논리 검증기를 실행한다. 이 때, RTTL 명세와 TES 명세의 실시간 성질 명세 기본자는 이미 그 의미가 RTTL 로 정의되어 있기 때문에 특별히 RTTL 로 변환시키는 과정을 거치지 않는다.

시스템의 규모가 너무 크고 복잡할 경우, 앞서 설명한 도달 가능성 분석 방법은 공간 폭발(Space Explosion) 문제를 가지고 있어 적용하기가 어렵다. 이런 경우 RTTL 논리 검증기 이용 방법을 사용하면 공간 폭발 문제없이 분석할 수 있다.

그림 11은 RTTL 논리 검증기를 이용하여 명세 분석을 하는 방법의 개략적인 모습을 보여준다.

이어지는 2. 실시간 시스템의 실시간 성질들에서는 정형적 분석의 대상이 되는 실시간 시스템의 실시간 성질들에 대해 설명한다.

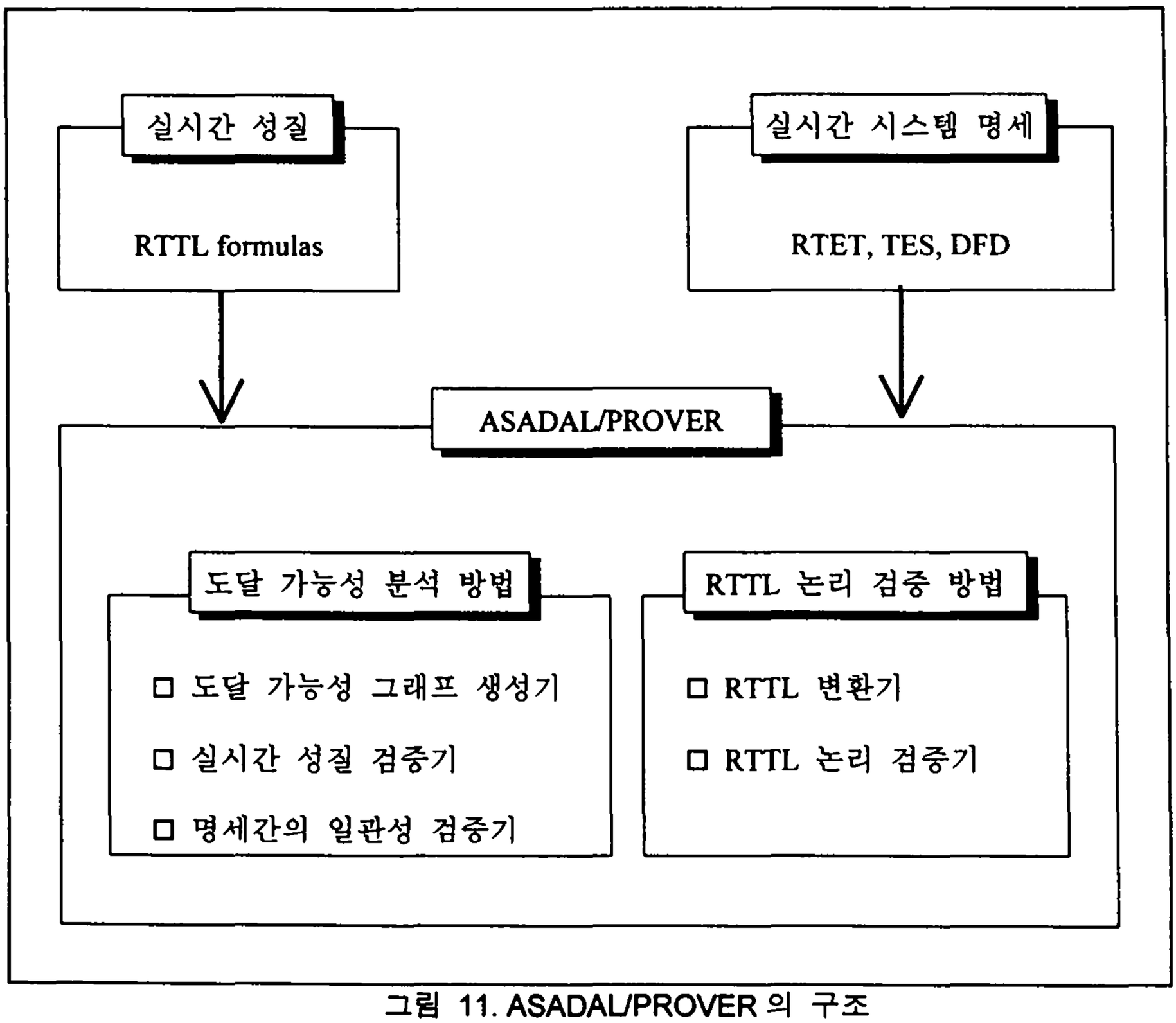


그림 11. ASADAL/PROVER 의 구조

2. 실시간 시스템의 실시간 성질들

시스템의 제약 조건(Constraints)은 시스템의 행동을 제약하는 것으로 시스템 동작 시 반드시 만족되어야 한다. 예를 들어, 실시간 운영체제를 보면, 어떤 실시간

운영체제 시스템은 "Deadlock 이 발생해서는 안된다.", "Starvation 이 일어나서는 안된다.", "요구(Request)에 대한 반응(Response)이 실시간적으로 일어나야 한다." 등과 같은 제약 조건들을 만족하도록 요구받을 수 있다.

이러한 시스템의 제약 조건들은 그 특성에 따라 안전성(Safety), 생존성(Liveness), 지속성(Unless) 등과, 특히 실시간 시스템에서 중요한 실시간 반응성(Real-time response) 등의 실시간 성질들로 나눌 수 있다.

이들 중 가장 기본이 되고 중요한 안전성, 생존성, 그리고 실시간 반응성을 이어지는 항에서 상세히 설명할 것이다.

이에 앞서, 앞으로 설명에 이용될 예로써 "식사하는 철학자들의 문제(Dining philosophers problem)"를 사용한다. "식사하는 철학자들의 문제"는 n 명의 철학자가 원탁에 둘러 앉아 식사를 하는데 각 철학자 사이에 수저가 하나씩 놓여 있고 한 철학자가 식사를 하기 위해서는 자신의 양쪽에 놓인 두 수저가 모두 필요하다고 한다 또한 자신의 양 옆에 있는 철학자 중 한 사람이라도 식사를 위해 수저를 가지고 있다면 자신은 식사를 할 수 없음을 알 수 있다. 여기에서는 문제를 간단히 하기 위해 철학자(phil1, phil2)를 두 사람으로 제한하고 두 철학자 모두 먼저 왼편에 있는 수저(sem1, sem2)를 들고 그 다음 오른편의 수저(sem2, sem1)를 들어 식사를 하기로 한다. 그리고 식사를 마친 뒤에는 두 수저를 자신의 양 옆에 다시 내려 놓는 방식으로 식사를 하는 것으로 가정하였다. 이를 TES 로 명세한 것이 그림 12이다.

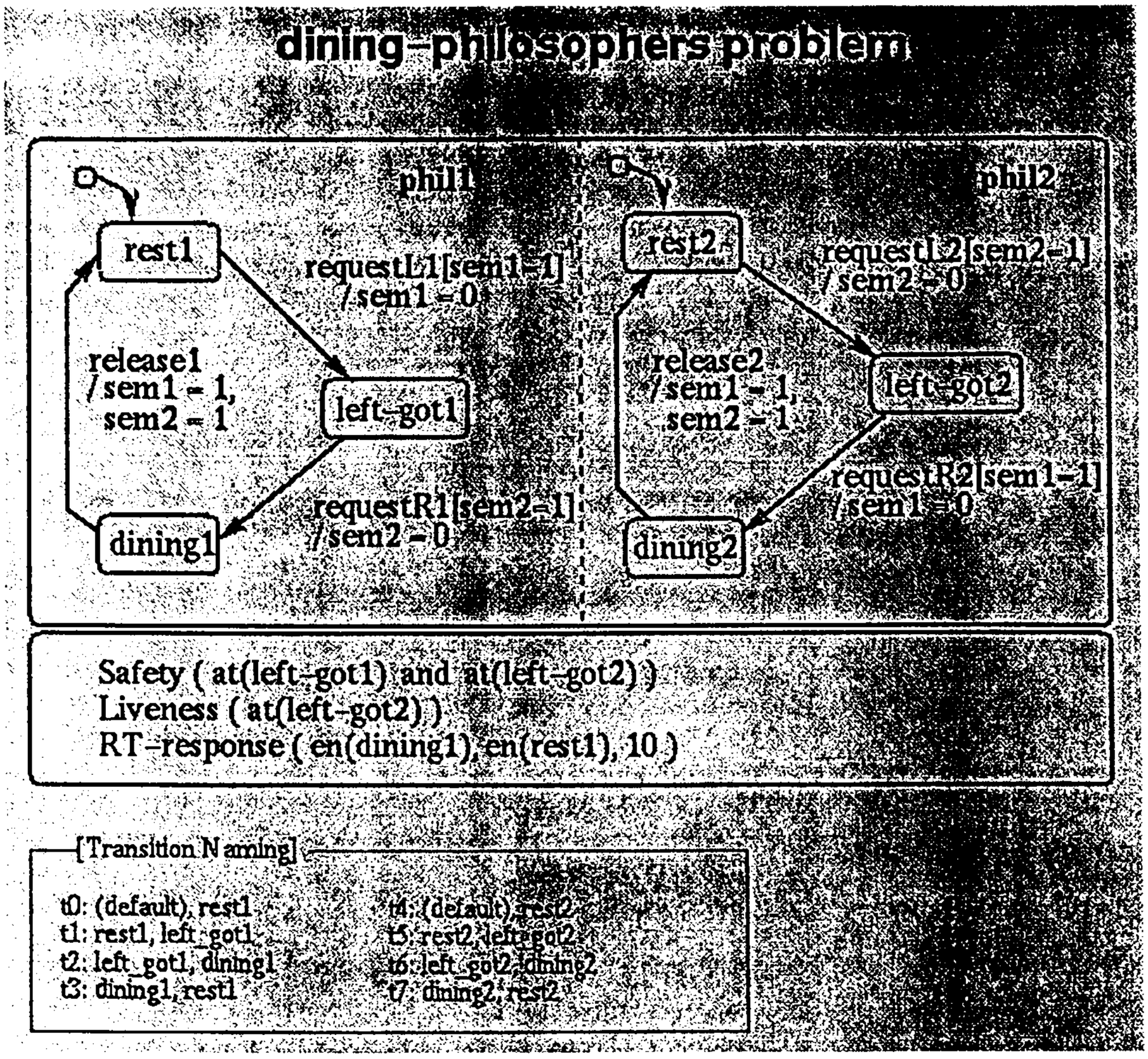


그림 12. 실시간 성질 예제

□ 안전성(p)

: 시스템의 행동이 속성 p 를 만족해서는 안된다.

일반적으로 속성 p 는 시스템의 "나쁜 상태"로서, 앞서 예로든 "Deadlock 이 일어나서는 안된다"에서는 "Deadlock 의 발생"이 바로 그 "나쁜 상태"이다.

그림 12에서 두 철학자가 초기 상태, 즉 각각 rest1 과 rest2 에 있고 sem1 과 sem2 모두 값이 1 로 초기화 되어있다고 가정하자. 이 때, 만약 두 철학자가 모두

자신의 왼쪽 수저를 요구하게 되면(RequestL1, RequestL2) 상태 전이 t1, t5 가 일어나서 철학자 1 의 상태가 left_got1 에 있게 되고 철학자 2 의 상태가 left_got2 에 있게 된다. 또한 상태 전이가 일어나면서 sem1 과 sem2 모두 값이 0 이 되기 때문에 둘 중 아무도 오른쪽 수저(sem1, sem2)를 더 이상 할당받을 수 없는 Deadlock 상태가 된다.

□ 생존성(p)

: 시스템의 행동이 속성 p 를 언젠가는 만족해야 한다.

일반적으로 속성 p 는 시스템의 "좋은 상태"로서, 앞서 예로든 "Starvation 이 일어나서는 안된다"에서는 "진행(Progress)"이 바로 그 "좋은 상태"이다. 그림 12에서는 철학자 2 가 언젠가는 left_got2 에 진입할 수 있어야 한다. 또는 언젠가는 식사를 하는, 즉 dining2 에 진입을 할 수 있어야 한다 등이 생존성에 관한 문제이다.

생존성을 증명하기 위해서는 앞서 설명한 안전성과는 달리 시스템의 진행(Progress)과 밀접한 관련이 있으므로, 증명하고자 하는 시스템이 공정성(Fairness)을 보장해야 한다. 이어지는 3 공정성(Fairness)에서 이에 관해 좀 더 자세히 설명할 것이다.

□ 실시간 반응성(p, q, t)

: 시스템의 행동이 속성 q 를 만족된 이후 t 시간 단위(Time Unit) 내에 속성 p 가 만족되어야 한다.

그림 12에서는 철학자 1 이 rest1 상태에 들어간 이후 10 시간 단위 내에 dining1 상태에 들어가야 한다는 실시간 성질을 나타낸 것이다.

3 공정성(Fairness)

공정성은 인터리빙(Interleaving) 방식의 실행 또는 자원(Resource) 할당 등의 문제에서 서로 독립된 프로세스(Independent Process)에게 균등한 기회를 주어야 한다는 개념으로, 이 성질의 표현 능력에 따라 정당성(Justice)과 배려성(Compassion)의 두 가지로 나뉜다.

각각에 대해서 자세히 설명하기 전에, 앞으로 사용하게 될 상태 전이(State Transition)에 관해 설명한다. 상태 전이는 전이가 일어나기 전의 상태인 출발 상태(Source State)와 전이가 일어난 후의 상태인 도착 상태(Destination State), 전이 시에 만족되어야 할 사건(Event)과 조건(Condition), 그리고 전이 시에 수행하게 되는 행동(Action) 등으로 이루어져 있다. 상태 전이의 출발 상태가 활동 중(Active)에 있고 상태 전이의 조건(Condition)이 만족되면 그 상태 전이가 활성화(Enabled)되었다고 한다. 만약, 조건이 없으면 그 상태 전이는 자동으로 활성화된다. 일단 활성화된 상태 전이는 그것의 사건이 일어나게 되면 도착 상태로 상태 전이가 이루어지고(Taken) 그 상태 전이의 도착 상태가 활동 중에 있게 된다.

□ 정당성(Justice)

: 계속 활성화된 상태 전이는 언젠가는 전이가 이루어진다.

그림 12에서 두 철학자가 초기 상태, 즉 각각 rest1 과 rest2 에 있고 sem1 과 sem2 모두 값이 1 로 초기화 되어있다고 가정하자. 현재 상태 전이 t1 과 t5 가 활성화되어 있음을 알 수 있다. 시스템이 만약 공정한 시스템이라면 언젠가는 사건 requestL1(requestL2)가 발생해 t1(t5)가 일어나게(Taken) 될 것이다. 이처럼 만약 한 상태 전이가 계속 활성화 되어 있으면 언젠가는 그 상태 전이가 일어나는 것을 보장한다면 그 시스템은 그 상태 전이에 대해서 정당하다라고 한다.

하지만 이러한 정당성만으로 시스템의 공정성을 충분히 보장하지 못하기 때문에 보

다 강력한 다음의 배려성이 필요하다.

□ 배려성(Compassion)

: 불연속적이지만 계속 활성화된 상태 전이는 언젠가는 전이가 이루어진다.

그림 12에서 두 철학자가 초기 상태, 즉 각각 rest1 과 rest2 에 있고 sem1 과 sem2 모두 값이 1 로 초기화 되어있다고 가정하자. 이 상태에서 requestL1 이 발생해서 상태 전이 t1 이 이루어지고 연속적으로 requestR1 과 release1 의 사건이 발생해서 철학자 1 이 left_got1 과 dining1 의 상태를 거쳐 다시 rest1 의 상태에 있게 되었다고 하자. 이 때, 상태 전이 t2 이 이루어지면서 sem2 의 값을 0 으로 바꾸어 놓기 때문에 그 순간 상태 전이 t5 는 활성화되지 못한다. 그리고 상태 전이 t3 가 이루어지면 다시 sem2 의 값을 1 로 바꾸어 놓기 때문에 그 순간 상태 전이 t5 는 다시 활성화된다. 즉, 철학자 1 만 이처럼 t1,t2,t3 를 반복해서 전이가 이루어지면 상태 전이 t5 는 불연속적으로 계속 활성화된다.

하지만 이러한 경우에서도 시스템의 공정성을 위해서는 철학자 2 가 언젠가는 진행할 수 있어야 하는데 이러한 성질이 바로 배려성이다. 즉, 이처럼 비록 불연속적이지만 계속 활성화될 경우 언젠가는 그 상태 전이가 이루어지는 것을 보장하는 것이 배려성이고 그러한 시스템을 그 상태 전이에 대해 배려한다라고 한다.

이어지는 4. 도달 가능성 그래프은 시스템의 내부 행동(TES)을 도달 가능성 그래프로 전환하는 방법에 대해 설명한다.

4. 도달 가능성 그래프

도달 가능성 그래프는 시스템의 내부 행동(TES)을 분석하여 만들어진 그 시스템이 도달할 수 있는 모든 상태를 가진 그래프이다.

아래의 단락들에서는 도달 가능 그래프가 만들어지는 방법과 만드는데 있어서 요구되는 사항들에 대해 설명한다.

도달 가능성 그래프에서의 노드(node)는 시스템의 전체 상태와 사상(mapping)되고 에지(edge)는 전체 상태 전이와 사상된다. 이러한 에지는 방향성을 가지므로 도달 가능 그래프는 일종의 방향성 그래프(directed graph)가 된다. 이어지는 항에서는 노드와 에지에 사상되는 구체적인 정보를 설명한다.

□ 노드(Node)

: 다음의 다섯 가지의 튜플(Tuple)들로 이루어져 있다.

1. 모든 활동 중인 상태들의 집합

2. 모든 조건 변수들이 가지고 있는 값들의 집합

3. 현재 발생된 사건들의 집합

4. 현재 스케줄되고 있는 타임아웃(Tim-out) 사건들의 집합

5. 현재 스케줄되고 있는 행동(Action)들의 집합

□ 에지(Edge)

: 다음의 세 가지의 튜플(Tuple)들로 이루어져 있다.

6. 출발 노드

7. 도착 노드

8. 이루어진(Taken) 상태 전이들의 집합

이렇게 해서 만들어진 도달 가능성 그래프 상에서 앞서 설명한 실시간 성질들을

증명하기 위해서는 이 도달 가능성 그래프가 유한한 크기, 즉, 이 그래프를 구성하는 노드의 수가 유한해야 하는 제한 조건이 있다. 따라서 TES에 사용되는 조건 변수의 형(Type)이 정수 또는 자연수여야 하고 또 그 값의 범위가 유한해야 한다. 또한, TES에서 사용되는 사건이 외부에서 발생하는 경우도 있고 내부에서 발생하는 것도 있으므로 도달 가능성 그래프를 만들기 위해서는 시스템의 외부에서 발생하는 외부 사건에 대해 명시해 주어야 한다.

이렇게 명시된 외부 사건과 조건 변수의 범위를 한정함으로써 TES의 도달 가능한 모든 상태를 유한한 범위 내에서 찾을 수 있게 된다.

5. 도달 가능성 그래프에서의 실시간 성질 검증 알고리즘

이 절에서는 앞서 설명한 유한한 도달 가능 그래프 상에서 안전성과 생존성의 실시간 성질들을 검증하는 알고리즘을 소개하고자 한다.

다음의 두 절은 각각 안전성과 생존성을 검증하는 알고리즘을 설명한 것이다.

가. 안전성 검증 알고리즘

도달 가능성 그래프를 시작 노드(Root Node)로부터 깊이 우선 찾기(Depth First Search) 방식으로 노드를 방문하면서 각 노드가 속성 p를 만족하는 가를 검사한다. 만약 p를 만족하는 노드가 없다면 이 시스템은 속성 p에 대해 안전하고, 반대로 p를 만족하는 노드가 하나라도 있으면 속성 p에 대해 시스템은 안전하지 않다고 할 수 있다.

깊이 우선 방식으로 도달 가능 그래프를 방문하므로 이 알고리즘의 복잡도(Time Complexity)는 O(n + e)이다. 여기에서 n 은 도달 가능 그래프의 노드의 개수이고 e 는 에지의 개수이다. 그림 13은 이 알고리즘을 Pseudo-language로 나타낸 것이다.

Procedure ***Safety*** (predicate P, reachability graph RG)

```
BEGIN
 node   nd;
 nd = get the root node of RG;
 DO {
          IF ( nd satisfies the predicate p )
          THEN this procedure fails;
          nd = DFS( RG );
 } WHILE ( nd is a node of RG )
 END
```

그림 13. 안전성 검증 알고리즘

나. 생존성 검증 알고리즘

생존성은 안전성과는 달리 시스템의 진행(Progress)과 관련이 있다. 즉, 시스템이 속성 p에 대해 생존성이 있다는 말은 시스템의 초기 상태로 부터 도달 가능한 모든 패스(Path)에 대해 언젠가는 p를 만족하는 노드에 도달할 수 있다는 뜻이다. 따라서, 시스템의 진행에 대한 성질인 정당성과 배려성을 이용해 생존성 검증 알고리즘을 기술한다.

시스템의 진행을 보장할 수 없는 상태는 다음의 두 가지 경우이다.

□ 한 노드에서 더 이상 진행을 보장할 수 없는 경우

: 노드의 아웃 에지(Out Edge) 중 정당성 또는 배려성을 가진 에지가 없을 때 이 노드에서 다른 노드로의 진행을 보장할 수 없다.

□ 루프(Loop)에서 더 이상 진행을 보장할 수 없는 경우

: 도달 가능성 그래프 내의 루프는 실제 시스템에서 무한히 수행될 수 있음을 의미한다. 그림 12에서 철학자 1만 계속 수행될 수도 있음을 이미 앞에서 보았다. 이때, 이 루프에서 다른 노드로의 진행을 보장하는 방법은 정당성과 배려성에

의해 가능하며 그 진행의 조건은 다음과 같다.

먼저, 정당성에 의해 루프에서 진행을 하기 위해서는, 그림 14와 같이, 그 루프를 이루는 모든 노드들이 이 루프 밖으로 연결된 에지를 가지고 있어야 하는데, 이때 이 에지들 모두에 사상되어 있는 정당성을 가진 상태 전이가 적어도 하나 있어야 한다(그림 14에서 Jt1). 단, 이 상태 전이가 루프를 이루는 에지 중에 사상되어서는 안된다.

다음으로, 배려성에 의해 루프에서 진행을 하기 위해서는, 그림 15과 같이, 그 루프를 이루는 노드들 중에서 이 루프 밖으로 연결된 에지를 가지고 있어야 하고, 이때 이 에지에 사상되어 있는 배려성을 가진 상태 전이가 있어야 한다(그림 15에서 Ct5). 단, 이 상태 전이가 루프를 이루는 에지 중에 사상되어서는 안된다.

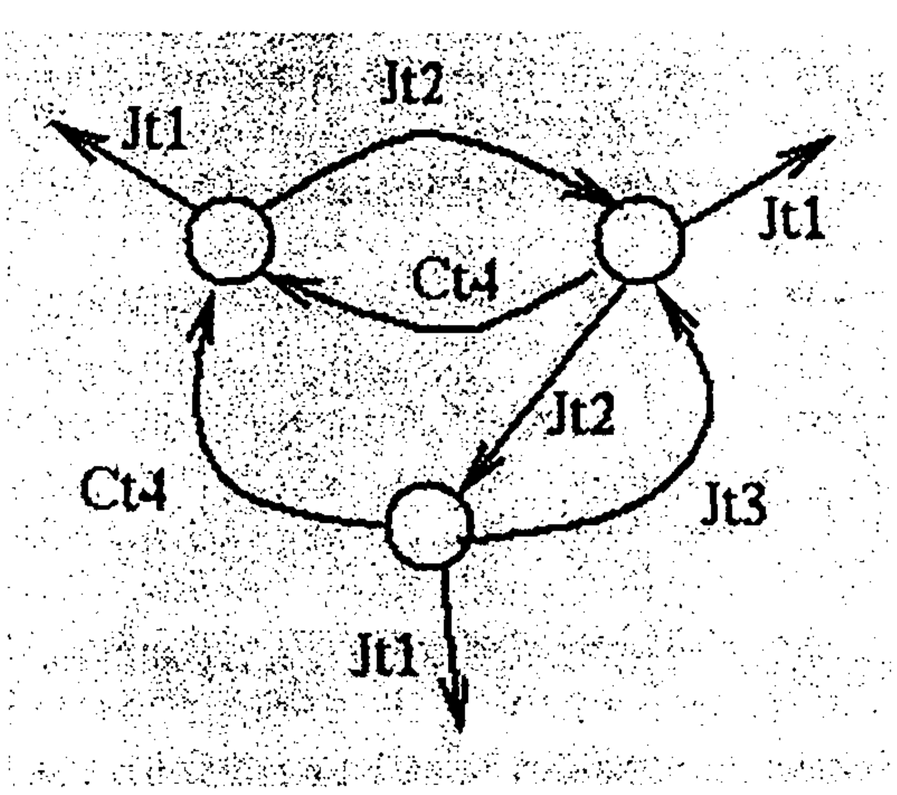


그림 14. 정당성에 의한 진행: Jt1

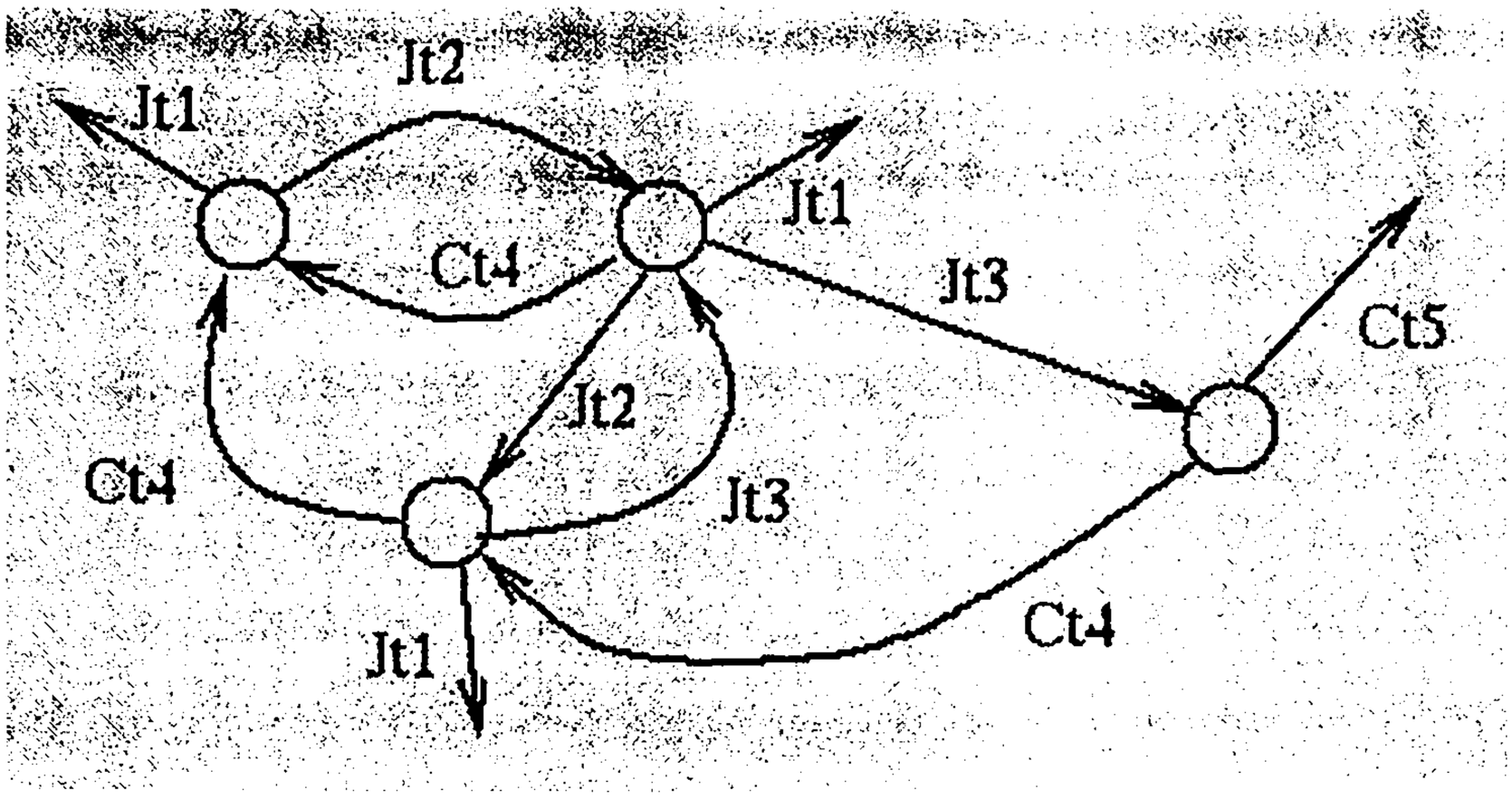


그림 15. 배려성에 의한 진행: Ct5

요약하자면, 생존성을 검증하기 위해서는 각 노드뿐만이 아니라 도달 가능성 그래프에 존재하는 모든 루프를 조사해야 한다는 사실이다. 따라서, 다음에 소개하는 생존성 알고리즘(그림 16)은 이러한 문제를 해결하기 위해 도달 가능성 그래프에 존재하는 루프들을 가능하면 지역화시켜서 문제의 범위를 축소하는 방법을 사용한다. 즉, 도달 가능성 그래프를 최대(Maximum) 강결합 컴포넌트(Strongly Connected Component)들로 나누는 방법이다. 최대 강결합 컴포넌트들 사이에서는 루프가 존재하지 않으므로 검사해야할 루프를 강결합 컴포넌트들로 한정할 수 있게 된다.

이 알고리즘의 복잡도는 강결합 컴포넌트로 나누는데 O(n + e), 그리고 알고리즘의 끝부분인 각 컴포넌트들의 진행 여부는 분할에 의한 풀이 방법(Divide by Conquer)으로 O((n' + e') log n')의 복잡도를 가지므로, 전체 복잡도는 O((n + e) log n)가 된다. 여기에서 n'은 컴포넌트를 이루는 노드의 수이고 e'은 컴포넌트를 이루는 에지의 수이다.

```
procedure   Liveness (predicate P, reachability graph RG)
BEGIN
Reconstruct RG with P such that the new RG has nodes visited with
```

```
DFS, where
the children of nodes satisfying the predicate P are not visited;
Decompose the new RG into its strongly connected components
C1 ... Cm;
FOR i = 1, ..., m DO
        IF Ci is terminal
        THEN this procedure fails
                if Ci doesn't satisfy the predicate P or isn't a node
        ELSE this procedure fails
                if Ci can not progress with Justice and Compassion
END
```

그림 16. 생존성 검증 알고리즘

7 절 지능형 생산 공장 시뮬레이션 모델

이상과 같이 만들어진 명세 언어 및 그 시뮬레이터를 응용하기 위한 지능형 생산 공장의 모델을 만들었다. 지능형 생산 공장이란 공장의 생산 시설 운영에 있어 작업자나 제어기의 오류가 아주 큰 손실을 발생시키거나 재난을 유발할 수 있을 때 이러한 문제들을 신뢰도 높은 고성능 소프트웨어를 활용함으로써 없애서나 빠르게 대처하게 하는 등의 자동화가 이루어 진 공장 형태를 말한다.

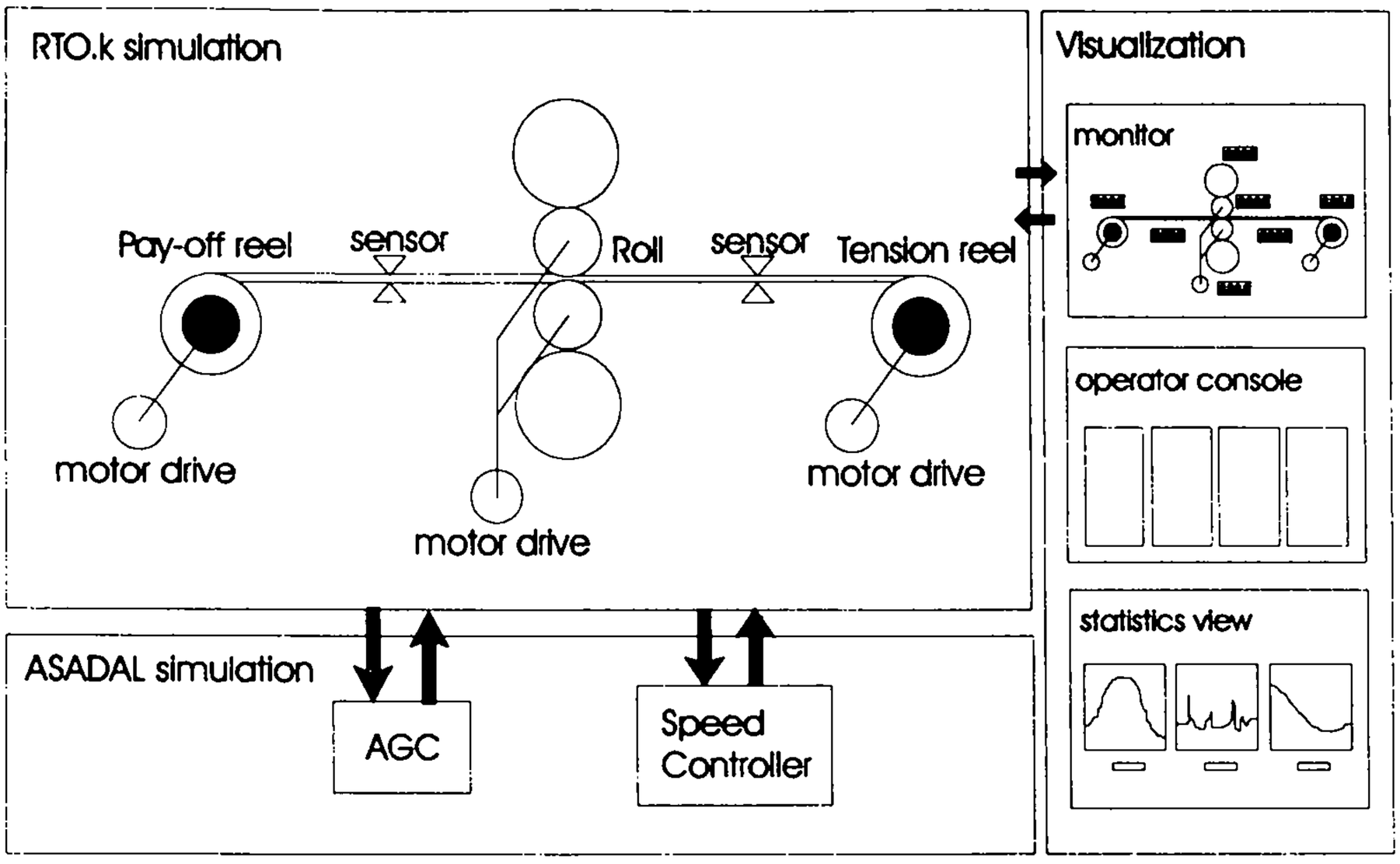
사람이 하던 일을 소프트웨어가 하게 되면 그 것의 실패나 오류 등에 대한 책임을 져야 하는 사람이 명백하지 않으므로 앞서 말한 분야 같은 경우 이러한 소프트웨어의 신뢰도는 특히 중요하다. 검증되지 않은 소프트웨어를 사용하지 않았을 때 발생하는 심각한 문제점들은 외국의 의료기기가 소프트웨어의 잘못으로 인명을 살상한 일이라거나 무인 여객기가 추락하게 된 일 등 사회 곳곳에서 나타나고 있다. 현재의 추세로 볼 때 이러한 소프트웨어를 이용한 사회 전반적인 시스템의 자동화는 계속 추진될 것이며 그 응용 분야도 무궁무진하다. 이러한 사회적인 변화에 발맞추어 소프트웨어의 질적 향상, 특히 안전성에 대한 중요성은 점점 높아질 수 밖

에 없으므로 이러한 시점에서 믿을 수 있는 시간 시스템에 대한 요구가 점차로 증대되고 있는 것이 사실이다.

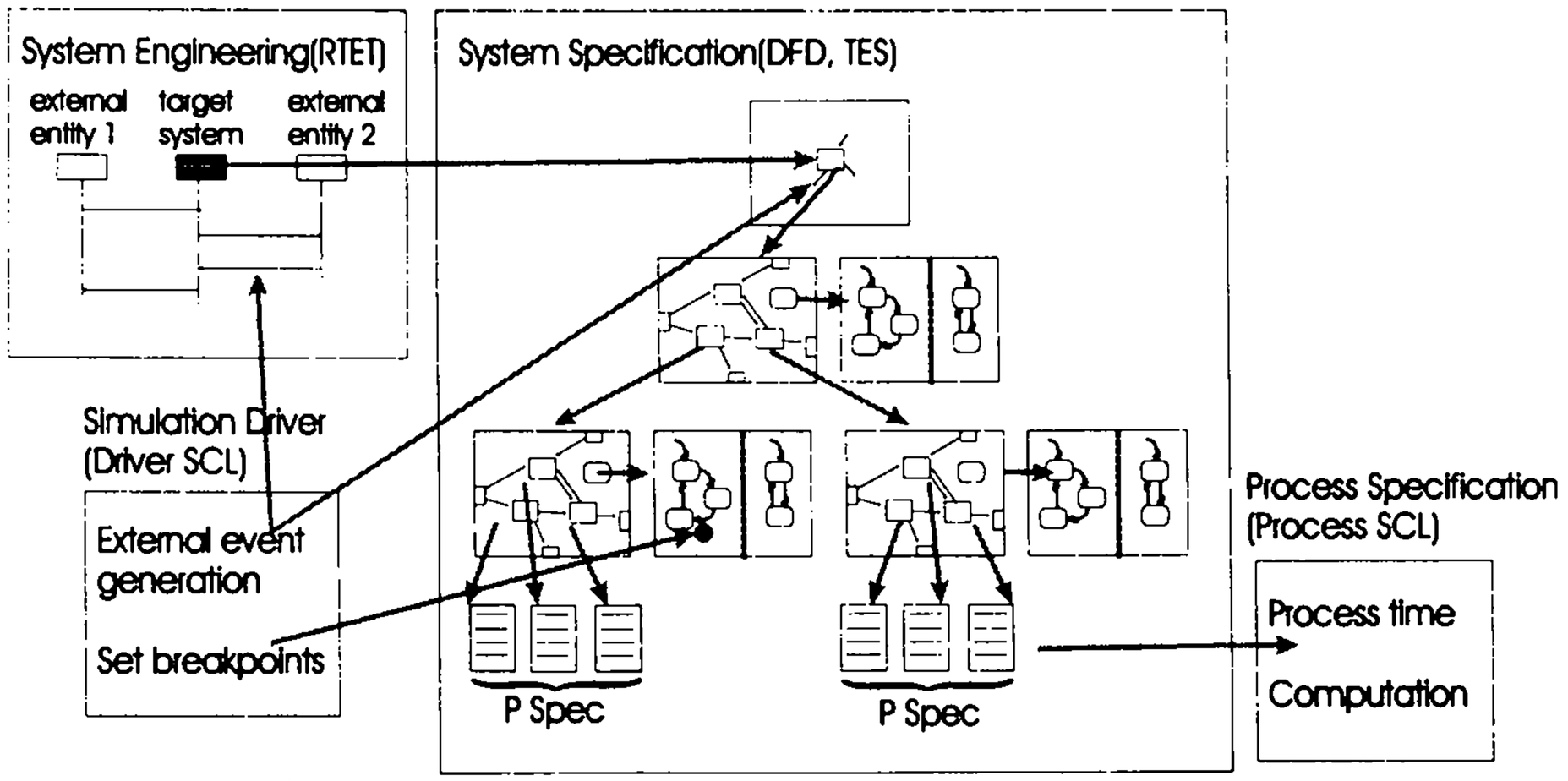
이러한 연구의 일환으로 믿을 수 있는 실시간 소프트웨어 시스템을 만들기 위한 해결책으로서 ASADAL 명세 언어(ASADAL/SPEC), 시뮬레이터(ASADAL/ SIM), 정적인 실시간 성질 검증기(ASADAL/PROVER) 등이 제시되었으며 이의 응용력을 검증하기 위해 지능형 생산 공장의 시뮬레이션 모델이 만들어졌다.

생산 공장의 경우 각각의 실제 기계 부분들의 행위에 대한 특성식들이 이미 주어여 있다. 그리고 이들을 제어하기 위한 제어 이론들도 많이 연구되어 있는 상태이다. 이러한 제어 이론들은 계속해서 현재 시스템의 상태를 센서로부터 읽어오며, 이 정보를 바탕으로 원하는 상태로 계속 다시 공장을 제어하기 위한 시그널을 내보내게 된다. 하지만, 이러한 제어 이론들은 대부분 기계들의 동작에 대한 이해에 기반을 두고 기계에 지금 어떤 값을 줘야 이것이 잘 동작할 지를 계산하는 데 연구가 집중되어 있다. 하지만 현재 시스템이 점점 복잡해질 수록 여러 기계 부분들을 제어하는 제어기 간의 협력적인 기능들도 증가하고, 시스템의 상태에 따른 제어 특성의 변화 등의 요구가 증가하고 있다. 이러한 상황에서 ASADAL 명세를 통해 시스템 및 제어기를 모델하고 그 명세를 직접 시뮬레이션 해 보거나 실시간 성질의 만족성을 검증해 보는 일들은 시스템 개발의 가능성을 조기에 검증해볼 수도 있고 시스템 자체의 안전성도 높여 주는 등 커다란 역할을 할 것이다.

다음 그림의 시뮬레이션 모델은 이 모델을 검증하기 위해 예제 시스템으로 사용된 냉간 압연기 두께 제어기(Automatic Gauge Controller; AGC)의 예를 이용해 보인 것이다. 이 모델에서 AGC 를 포함한 여러 제어기들은 ASADAL 모델로 시뮬레이션되고 외부 개체들은 RTO 로 시뮬레이션되고 있다. 그리고 시스템 환경의 운영 모습을 보이는 부분이 있다. 이들간의 연결은 간단한 네트웍상의 데이터 교환으로 이루어진다.



ASADAL 시뮬레이션을 위한 명세가 제어 시스템 모델이 되는데, 이 모델은 다음 그림과 같이 이루어진다.



이 모델은 먼저 시스템 공학을 통한 시스템 개체의 판별과 그들간의 데이터 흐름의 시나리오를 잡아내고, 이 개체들 중 목표로 삼을 제어 시스템 개체를 선택, 그 명세를 DFD 를 이용, 하향 상세화하고 그 행위는 TES 로 명세한다. 그리고 최하위

단계 프로세스에는 프로세스 명세를 주고 외부 개체들의 행위를 묘사하는 시뮬레이션 드라이버 프로그램을 작성한다.

이러한 단계가 ASADAL 시뮬레이션을 위한 것의 전부로서 이 단계들을 거치면 지능화된 생산 공장을 위한 제어기의 시뮬레이션이 준비된다.

8 절 결론

ASADAL 시뮬레이션을 이용하면 이른 시스템 개발 주기에 빠른 시스템 검증이 가능하며 사용자와 대화형으로 이루어지는 시뮬레이션뿐 아니라 추계적인 방식을 이용하여 외부 환경을 묘사, 배치형으로 시뮬레이션 할 수 있게 하였으며, 실시간으로 시뮬레이션할 수 있도록 시간 개념을 명세 및 시뮬레이터에 넣었고 외부 다른 도구와의 인터페이스를 쉽고 간단히 할 수 있도록 설계, 구현하였다.

ASADAL 명세 언어는 시스템을 외부적 관점과 내부적 관점으로 완전히 분리, 다른 시각에서 시스템을 분석할 수 있게 하고 이들간의 일관성을 검증하여 관점이 분리됨으로 해서 생길 수 있는 불일치를 막았다. 명세 언어로 쓰이는 RTET 은 사용하기 아주 쉬울 뿐 아니라 시스템의 실시간 제약을 사용자 관점에서 기술하기 쉽게 되어 있다.

만들어진 이 명세 언어에 대한 실시간 제약 검증 방법은 실시간 시스템이 가져야 할 여러 성질들을 시뮬레이션 전에 이미 검증, 옳게 되었는지를 파악할 수 있게 해 준다. 이 검증 방법에 의하면 어떠한 성질이 100%만족됨을 증명할 수 있다.

ASADAL 명세 방법 및 시뮬레이터는 신뢰도 높은 시스템 개발에 크게 기여할 것이다.

여 백

제 2 장 실시간 그래픽 사용자 인터페이스 개발

1 절 서론

시뮬레이션(Simulation)의 결과로 발생되는 수많은 양의 수치적 데이터를 그냥 사용자에게 실시간으로 보여준다든지 저장한 후 나중에 사용자로 하여금 검색해 볼 수 있게 한다는 것은 별로 의미가 없다. 방대한 양의 정보를 각계 각층의 다양한 사용자가 잘 이해하고 분석할 수 있도록 하기 위해서는 수치 데이터가 아닌 특별한 지식을 요구하지 않고 쉽게 이해할 수 있는 그래픽으로 나타내 주어야한다. 그래픽으로 나타내는데 있어서 수치적 데이터를 쉽게 보기 위한 히스토리 다이어그램(History Diagram)이나 변하는 양을 실시간에 쉽게 보도록 해 주는 막대 그래프와 같은 단순한 2차원적인 그래픽으로만 나타내어 준다면 사용자는 이러한 그래픽이 나타내는 의미를 이해하기 위해서 상당한 노력과 시간을 들여야 하며, 비 전문적인 사용자는 거의 이해할 수 없게 된다.

따라서 전문적인 사용자뿐만 아니라 비전문적인 사용자도 잘 이해할 수 있도록 하기 위해서는 시뮬레이션하고자 하는 실제 상황(Physical Environment)을 컴퓨터 속의 가상환경(Virtual Environment)에서 3차원 컴퓨터 그래픽스 기법을 이용하여 그대로 구현하고 시뮬레이션되고 있는 상황에 맞게 가상환경을 바꾸어 줄 수 있는 실시간 3차원 사용자 인터페이스가 필요하다.

가상환경속에서 사용자가 실제의 시뮬레이션되고 있는 상황을 보고, 시점을 바꾸고 가상물체(Virtual Object)를 다루는 등의 사용자의 의도나 명령을 가상환경속에 전달하기 위해서는 가상환경과 상호작용(Interaction)을 할 수 있는 방법이 필요하다. 사용자는 이러한 상호작용을 통해서 가상환경에서 일어나고 있는 실제의 시뮬레이

션을 더 잘 이해할 수 있다.

가상환경속에서 사용자의 이해를 돕고 사용자의 의도나 명령을 가상환경 속의 가상물체에 전달하기 위한 상호작용의 종류를 살펴보면 다음과 같다.

- 사용자의 시점을 바꾸기 : 사용자가 현재 보고 있는 위치나 보는 방향을 바꿈으로써 효과적으로 가상환경을 살펴보고 시뮬레이션 된 결과를 더 잘 이해할 수 있다.

- 가상물체를 선택하기 : 가상환경속에서 가상물체와 상호작용을 하기 위해서는 먼저 관심이 있는 물체를 선택할 수 있어야 한다.

- 가상물체를 다루기 : 가상환경속에서 가상물체를 선택한 후 가상물체를 이동, 회전시키고 확대/축소하는 것과 같은 가상물체 다룰 수 있어야 한다.

- 가상메뉴 : 가상환경속에서 가상물체에 사용자의 의도를 전달하고 명령을 주기 위해서는 2 차원 환경에서의 메뉴와 같은 것이 가상환경속에 있어야 한다.

본 과제에서는 실시간 압연공정 시뮬레이터를 통해서 얻어진 데이터를 가상환경에서 실시간에 3 차원으로 사용자와의 상호작용을 통해서 디스플레이할 수 있는 그래픽 사용자 인터페이스를 개발하고 이를 위한 기반 기술을 개발한다. 실시간 그래픽 사용자 인터페이스의 전체적인 개괄도는 그림 17에 나타나 있다. 실시간 그래픽 사용자 인터페이스를 사용할 수 있는 환경을 그래픽 처리를 위한 SGI machinne 과 같은 특별한 기계에 제한할 경우 이식성(Portability)에 문제가 생길 수 있으므로 본 연구과제에서는 PC 나 다른 워크스테이션에서 사용될 수 있도록 C 언어와 OpenGL 을 사용하여 가상환경을 구현하였다. 또한 HMD(Head Mounted Display), StereoGlass 와 같은 디스플레이 장치나 사용자가 6 개의 자유도(Degree Of Freedom)를 제어할 수 있는 Tracking Device 는 보편적으로 사용되고 있지 않으므로 입력장치로

마우스를 이용하였다.

본 보고서에서는 2절에서 가상환경을 구현하고 구현된 환경을 실시간에 가시화하기 위한 방법을 기술하고 3절에서는 가상메뉴를 4절에서 가상도구를 5절에서 시점변화, 물체와 상호작용을 하기 위한 방법을 기술한 후 6절에서 결론과 향후 연구계획에 대해서 기술한다.

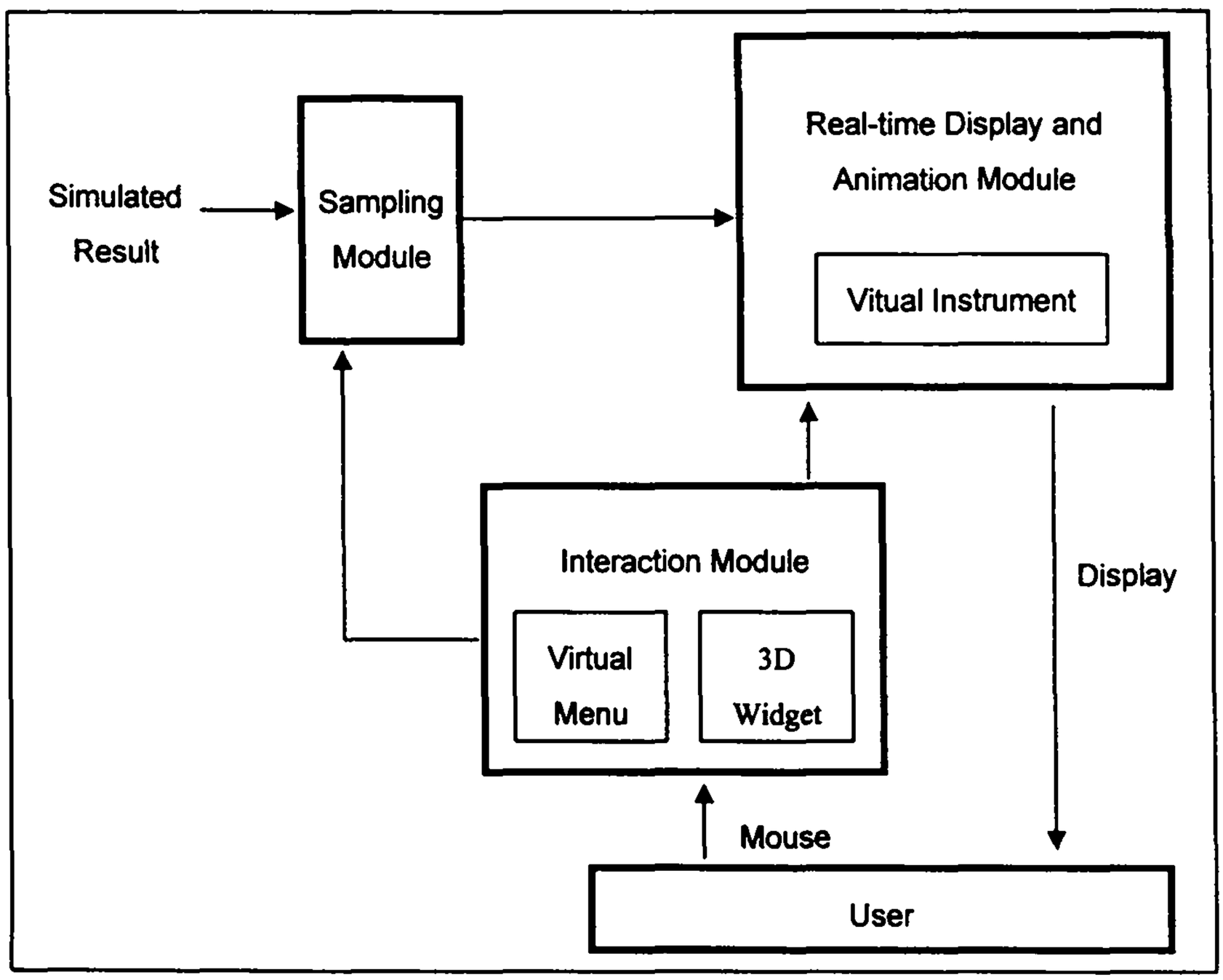


그림 17 실시간 그래픽 사용자 인터페이스의 개괄도

2절 실시간 3차원 가시화

본 절에서는 시뮬레이터를 통해서 나온 결과를 실시간에 3차원으로 가시화하기 위해 샘플링(Sampling)을 하는 방법과 샘플링된 데이터를 받아서 실시간에 가시화하고 에니메이션하는 기술에 대해서 기술한다.

1. 샘플링(Sampling)

현재의 컴퓨터의 성능의 한계 때문에 실시간 시뮬레이터를 통해서 나오는 데이터를 그대로 실시간에 다 보여주기는 어렵다. 따라서 가시화를 위해서는 적절한 양의 정보를 샘플링하는 전처리 과정이 필요하다. 이와는 별도로 사용자가 시뮬레이터를 통해서 나온 결과를 이해하기 쉬운 적절한 양으로 샘플링해서 보는 방법도 필요하다. 이 경우에는 사용자의 샘플링률은 시스템의 샘플링률보다 낮아야 한다.

이러한 샘플링률을 설정하는 방법은 시뮬레이션을 시작하기전에 설정하는 방법과 시뮬레이션동안에 사용자와의 상호작용에 의해서 설정하는 방법이 있다. 사용자와 상호작용하는 방법에 대해서는 5절에 기술한다.

본 연구에서는 실시간 압연공정 시뮬레이터로부터 나오는 데이터(출측/입측의 롤 반경, 롤 속도, 장력, 판 두께, 작업롤의 간격, 반경, 속도 등)를 TCP/IP를 이용해서 받은 후 이 데이터를 일정한 시간 주기로 샘플링하고 샘플링된 데이터를 실시간에 가상공간에 가시화한다.

2. 가상환경의 모델링과 가시화

본 연구에서는 가상환경을 모델링하기 위해서 OpenGL을 사용한다. OpenGL은 단순한 프로그래밍 모델을 가지고 있으면서 진보된 랜더링(Rendering) 특징을 제공하기 때문에 현재 점차적으로 그래픽스의 표준으로 되어가고 있다. OpenGL을 사용한 이유를 기술하면 다음과 같다.

- 랜더링만을 제공 : OpenGL 은 단지 랜더링에 관계된 기능만을 제공하기 때문에 어떠한 윈도우 시스템에서도 사용할 수 있다.
- 하드웨어 독립적 : 단순한 프레임버퍼를 가진 시스템에서 아주 복잡한 그래

픽스 처리 부시스템을 가진 시스템까지 거의 모든 시스템에서 OpenGL 을 사용할 수 있다.

□ 2D, 3D 그래픽스의 기본적인 연산에 대한 쉽고 직접적인 통제 수단을 제공한다.

□ 계층적 구조 : 상위 계층의 그래픽 기능들은 하위 계층을 그래픽 기능으로 구현되어있다.

본 단원에서는 실시간 그래픽 사용자 인터페이스의 모델링과 가시화에 필요한 OpenGL에 대해서 기술한 후 실시간 압연공정 시뮬레이터를 위한 모델링과 가시화에 대해서 기술한다.

가. OpenGL

그래픽 API(Application Programmer's Interface)를 제공하는 소프트웨어 시스템으로 2 차원적인 PostScript, X window system 이 있으며 3 차원적인 PHIGS(Programmer's Hierarchical Interactive Graphics System), PEX, Renderman, HOOPS, IRIS invertor 와 OpenGL 이 있다.

OpenGL 에서는 선택되어진 일련의 모드(Mode)를 통해 기본 요소(Primitive)들을 프레임 버퍼에 그려넣는 기본적인 구조를 가지고 있다. 출력 기본 요소에는 점, 선, 다각형 그리고 픽셀 영역, 비트맵과 같은 이미지들이 있으며, OpenGL 에서의 모드는 각 프레임 버퍼에 대해 독립적으로 변환시킬 수 있다. 따라서 모든 연산은 함수나 프로시저 콜을 통해 행해 진다.

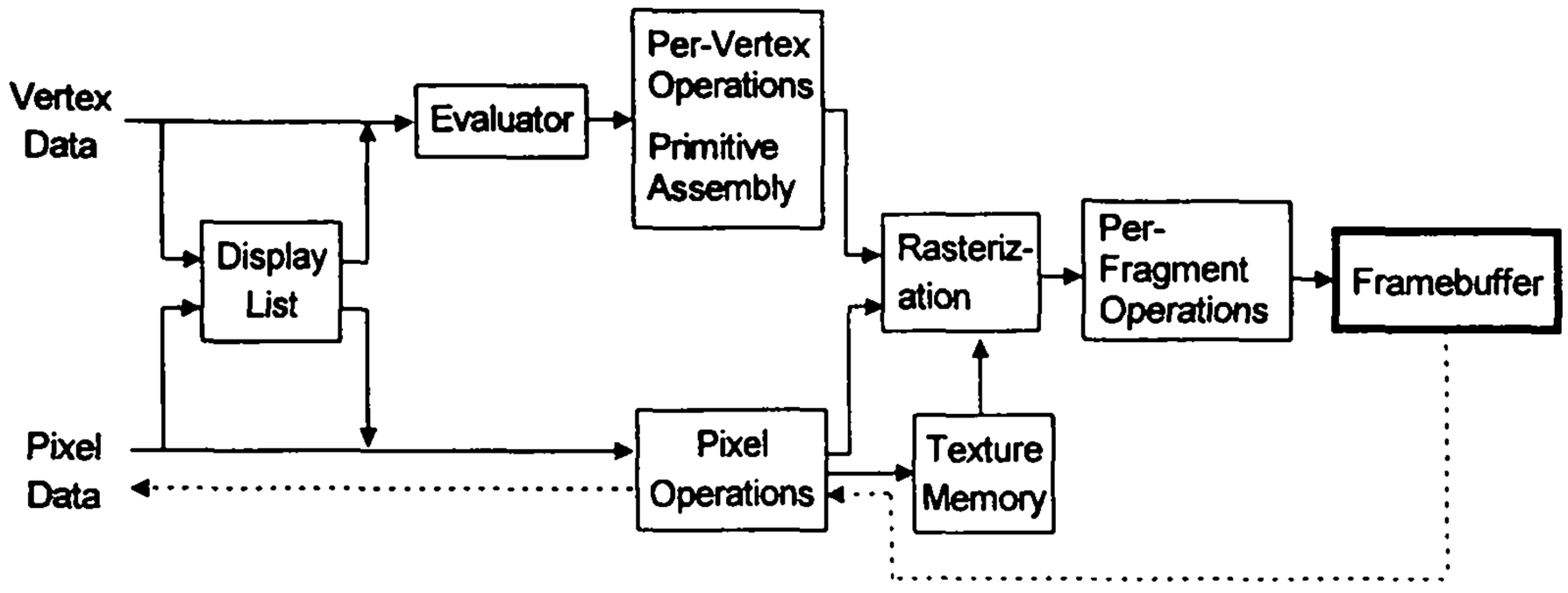


그림 18는 OpenGL의 구성도 이다. 곡선, 곡면과 같은 임의의 객체가 들어올 경우 그 과정을 살펴보자. 우선 대부분의 명령어는 디스플레이 리스트(Display List)에 첨가되어 효과적으로 처리 파이프라인(Processing Pipeline)을 이용하여 수행된다. 다음으로 곡선이나 곡면에 대한 기하적인 정보는 다항식 함수를 이용하여 근사값을 구하는 과정을 거치게 된다. 즉, 기하적인 수치들은 출력 기본 요소인 점, 선, 다각형의 형태로 계산되어지고 경우에 맞는 변환 과정을 거쳐 다음 단계를 위하여 관측 공간(View Volume)에 맞게 절단하게 된다. 이렇게 형성된 기본 요소들은 레스터라이저(Rasterizer)에 의해서 2차원 데이터를 이용하여 프레임 버퍼 상의 주소와 값들을 계산해, 단편(Fragment)으로 생산한다. 이렇게 생산된 각 단편들은 최종의 프레임 버퍼 변경 전까지 연산이 수행될 수 있도록 피드백 된다. 이 경우 수행되는 연산을 보면, 새로 들어온 단편에 대한 값과 전에 저장된 깊이 값(Depth Value)을 비교하여 수정하는 연산, 저장된 색깔과 들어온 색깔간에 배합 함수(Blending Function)를 사용하여 수정하는 연산, 단편들간의 값에 대한 마스킹(Masking), 논리합(OR), 논리곱(AND)과 같은 논리 연산이 있다. 그 외에 프레임 버퍼로 부터 복사된 일부의 픽셀 영역이나 직접 읽혀진 비트맵과 같은 이미지들은 앞에서 말한 기본 요소를 만드는 과정을 거치지 않고 직접 레스터라이저를 거쳐 프레임 버퍼 상에 올 수 있도록 하는 과정도 포함하고 있다.

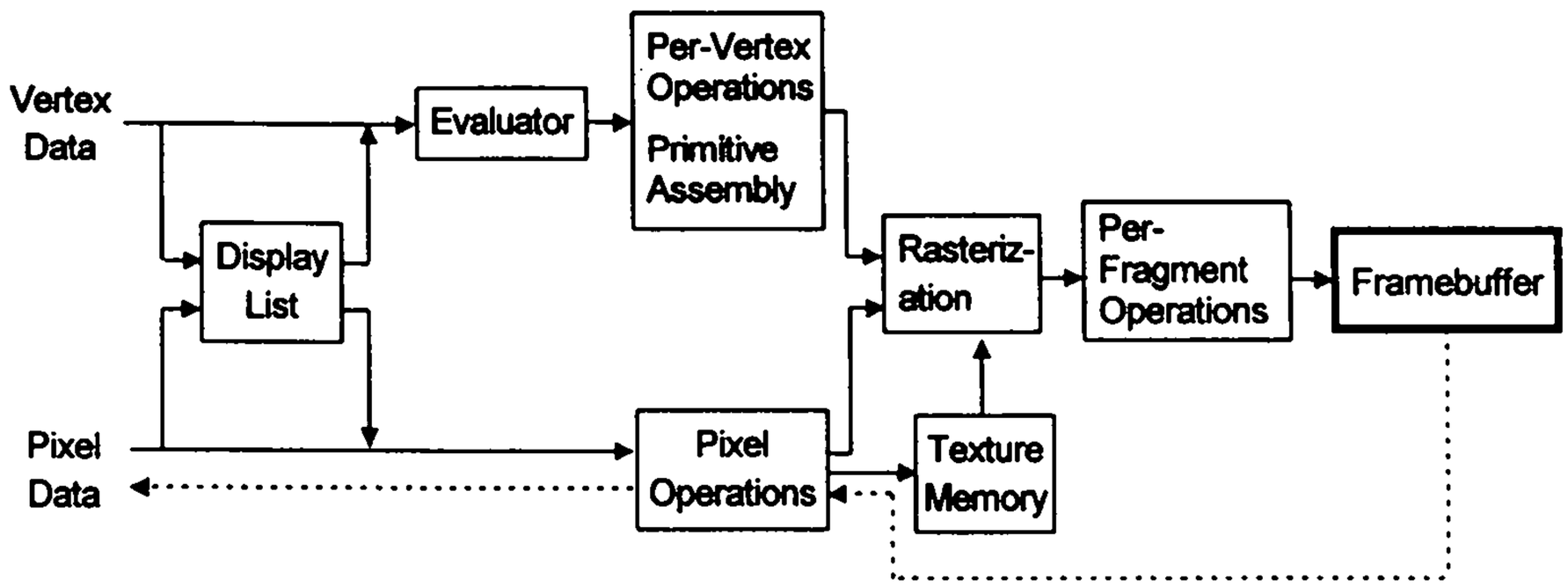


그림 1 8 OpenGL 구성도

그래픽스 API 를 디자인하는 데에는 고려될 사항간의 이율배반성(Tradeoff)을 갖고 있으며, 고려 사항들은 OpenGL 을 디자인하는데 많은 영향을 주게 되었다. 아래에 고려사항들을 나열한다.

(1) 성능(Performance)

상호작용적(Interactive) 3 차원 그래픽스에서의 근본적인 고려 사항은 성능 문제이다. 현재의 기술로는 그리 복잡하지 않은 3 차원 장면을 만드는 데에도 상당한 양의 계산이 필요하다. 따라서, 상호작용적 3 차원 응용 프로그램에서 사용하는 API 는 응용 프로그램에서 사용하는 그래픽 하드웨어의 능력을 효과적으로 접근할 수 있는 공통의 인터페이스를 제공해야 한다. 이 인터페이스는 단속적(On and Off) 렌더링 특징을 부여해야 한다. 그 이유를 기술하면 다음과 같다.

□ 각 부시스템들의 성능 차이 : 몇몇의 하드웨어에서는 어떤 특징을 지원하지 않을 수 있으며, 또한 지원을 하고 있더라도 인정할 만한 성능을 내지 못하는 경우가 있다.

□ 하드웨어 특징들의 조합으로 인해 전체적 성능에서의 감소 현상 발생 : 하나의 이미지를 만들어 내고자 한다면 렌더링 시간이 중요하지 않지만, 객체

에 대한 조작이 있고 관점(Viewpoint)과 장면의 수정이 요구되는 경우, 정교한 한 이미지를 만들어 낸다는 것은 옳지 않다. 이때에는 전체 이미지 렌더링을 하지 못하게 함으로써 전체의 성능을 높인다.

(2) 직교성(Orthogonality)

단속적 특징이 있으면 특징들간의 부작용(Side-Effect)이 생긴다. 예를 들면, 여러 개의 다각형을 면에 따라 색깔을 삽입하는 방법을 채택하지 않고 하나의 색깔로 칠한다고 가정하면 텍스춰 맵핑이나 조명은 의미가 없다. 유사하게 어떤 한 특징을 허락하거나 허락하지 않을 때 렌더링의 결과가 기대하지 않은 결과를 낳는 일관되지 않는 상태(Inconsistent State)를 보여서는 안된다. 이는 특징들간에는 독립성을 가져야 한다는 말과 일맥상통한다. 결국, 이 사항은 프로그래머에게 미리 시험해 보지 않고도 규칙에 맞지 않는 특징의 조합을 알려줄 수 있는 능력을 API가 지니고 있어야 한다.

(3) 완성도(Completeness)

그래픽 부시스템을 가진 한 시스템에서 실행되는 3차원 그래픽스 API는 그 부시스템의 모든 기능을 접근할 수 있도록 해야 한다. 만약, 접근할 수 있는 기능과 접근할 수 없는 기능이 동시에 존재하면, 접근할 수 없는 기능에 대해서는 다른 API를 써야 한다 여러 API간의 상호작용은 응용 프로그램을 복잡하게 하는 요소이기도 하다. 한 하드웨어 플랫폼에서 API가 구현되었다면 그 API를 제공하는 임의의 다른 플랫폼에서도 모두 실행 가능해야 한다.

(4) 정보처리 상호운용성(Interoperability)

많은 계산 환경들은 제조 회사가 다른 컴퓨터들이 네트워크로 연결되어 있다. 이러한 환경하에서 한쪽의 기계는 그래픽 명령어만을 고려하고, 다른 한쪽에서는

그 명령에 대한 실행을 담당하는 형태로 구분되는 것이 매우 유용하다. 이러한 능력을 "정보처리 상호운용성" 이라는 말로 표현하며 정보처리 상호운용성을 갖고 설계된 실행 모델이 클라이언트-서버 모델(Client-Server Model)이다. 클라이언트는 명령어들을 주게 되며, 서버는 그에 대한 실행을 담당하는 형태이다. API 명령어들의 실행은 클라이언트-서버 모델에 따라야 하며 클라이언트와 서버간의 전달에 관한 규약은 미리 정해져 있어야 한다.

(5) 확장성(Extensibility)

3 차원 그래픽스 API 는 새로운 그래픽스 하드웨어 특징이나 알고리즘을 결합할 수 있는 확장성을 가져야 한다. 하지만 API 를 처음 사용하는데 있어 확장성의 성취 정도를 측정한다는 것은 매우 어렵다. 이에 대한 하나의 해결 방안으로 앞에서 설명한 API 의 직교성(Orthogonality)을 고려하면 될 것이다. 그외에 디자인시 일부러 생략한 특징에 대해 차후 이를 삽입할 경우, API 가 어떤 영향을 갖게 될 것인가를 고려해야 한다.

(6) 수용성(Acceptance)

분명하고 일관된 3 차원 그래픽스 API 의 디자인은 디자인 자체에 충분한 목적을 지니고 있다. 하지만 프로그래머가 다양한 응용 프로그램에 대해 사용할 API 를 결정하지 않는다면 API 를 디자인한다는 것은 목적 없는 일이 된다. 따라서 프로그래머의 API 에 대한 수용능력에 대한 고려는 API 디자인시 매우 중요한 일이다.

OpenGL 은 디자인시 위와 같은 사항들을 고려하여 설계되었다. 또한 실리콘 그래픽스(Silicon Graphics)의 IRIS GL 에 바탕을 두고 있으며, IRIS GL 을 통해 얻어진 경험을 통해 프로그래머가 원하는 3 차원 API 가 가져야 특성과 원하지 않는 특성들을 OpenGL 이 보강하고 있다. IRIS GL 을 사용한 모든 프로그래머는 매우 쉽게

OpenGL 을 접할 수 있다. 그외에 OpenGL 이 갖고 있는 디자인 특징을 살펴본다.

(1) 하위 계층(Low Level)

OpenGL 의 궁극적 목표는 하드웨어 상의 기능들을 완전히 접근할 수 있도록 장치에 대한 독립성을 제공하는데 있다. API 가 장치의 독립성을 제공한다면, 그래픽 연산은 가장 낮은 하위 계층에서 수행되어야 한다. OpenGL 은 복잡한 객체 모델링 수단을 제공하지 않고, 객체들을 어떻게 표현할 것인가에 대한 구조(Mechanism)를 제공한다.

하위 계층 API 의 이득은 응용 프로그램이 좀더 상위 계층의 객체들을 표현하는 방법에 대해서 고려하지 않아도 된다는 점에 있다. 즉, 기본적인 OpenGL API 는 보통의 그래픽스 API 들에서 제공하는 기하적인 객체들을 제공하지 않는다. 경우에 따라 OpenGL 은 오목다각형(Concave Polygon)에 대해 렌더링 효과를 제공하지 않는데, 그 이유 중의 하나는 오목다각형 렌더링 알고리즘은 볼록다각형(Convex Polygon)에 대한 렌더링 알고리즘보다 더 복잡하므로, 오목다각형이 한번 이상 그려질 경우 오목다각형을 볼록다각형으로 근사 시켜 표현하는 것이 더 효율적이기 때문이다. 또 다른 이유는 오목다각형에 대한 자료 유지에 있다. 오목다각형은 다각형에 대한 모든 점들을 유지하기 위해 많은 메모리를 요구한다. 그러나 볼록다각형의 경우, 모든 볼록다각형이 삼각형으로 축소될 수 있기 때문에 세 점에 대한 저장 공간만 있으면 표현 가능하다.

OpenGL 상 에서의 하위 계층과 상위 계층간의 구분에 대한 예를 OpenGL 의 수치계산기 (Evaluator)와 NURBS 의 차이점으로 살펴본다. OpenGL 의 수치계산기는 일반적인 다항식 곡선과 곡면을 만든다. 복잡한 NURBS 인터페이스를 사용하는 것과 비교해 볼 때, 특별한 곡선과 곡면에 대한 특성을 다항식의 수치계산기로 접근 가

능하며, 일반적인 곡선과 곡면들을 NURBS로 표현하는 전환 비용을 줄일 수 있다.

IRIS GL을 사용한 프로그래머에게 오목다각형과 NURBS에 대한 계산은 일반적이다. OpenGL 유틸리티 라이브러리에는 기본적으로 오목다각형을 볼록다각형으로 분해하는 방법을 제공하고 있으며, 구, 원추, 원기둥, NURBS 곡선과 곡면을 묘사할 수 있도록 인터페이스를 제공한다. OpenGL의 유틸리티 라이브러리는 다른 라이브러리에서 만들어진 모델과 OpenGL에서 사용하는 모든 기하학적 객체들을 사용할 수 있도록 해준다.

유틸리티 라이브러리 명령어들은 클라이언트에서 OpenGL 명령어로 변환된다. 클라이언트-서버 환경에서 서버의 계산 능력이 뛰어나다면 변환 과정을 서버쪽에서 하는 것이 더욱 효율적일 수 있다는 딜레마가 존재하지만, 이러한 딜레마는 OpenGL 쪽의 문제가 아니라 라이브러리 쪽의 문제로 전환되어야 한다. OpenGL은 근본적으로 그래픽스 부시스템들의 기능 향상을 전체적인 기능의 향상으로 보고 있다. 따라서 서버의 능력이 뛰어나더라도 부시스템의 기능이 효과적이지 않다면 전체적 의미를 상실하게 된다. 만약 미래에 NURBS를 지원하는 효과적인 그래픽 부시스템이 개발된다면 OpenGL의 전체적 성능은 향상된다.

(2) 잘 다져진 제어(Fine-Grained Control)

한 응용프로그램이 임의의 API를 사용하면, 데이터에 대한 표현과 저장을 효과적으로 할 수 있도록 기하학적 객체나 그들에 대한 연산의 성분을 정의한다. OpenGL에서는 아래와 같이 일련의 기하학적 좌표들에 대한 객체를 정의한다.

```
glBegin(GL_POLYGON);

glVertex3i(0,0,0);
```

```
glVertex3i(0,1,0);

glVertex3I(1,0,1);

glEnd()
```

이처럼 데이타를 하나의 프로시저 그룹으로 만들어 놓으면 그래픽스 API 에서 표현되는 형태로 다시 저장할 필요가 없다. 그러나 프로시저 콜에 있어 많은 비용이 든다면 성능이 매우 저하되는 현상을 나타낸다. OpenGL 에서의 프로시저 콜은 시스템 필요시에 따라 또는 사용자가 충분한 정보를 표현하고 있지 않은 경우에 따라 차선의 선택을 취할 수 있으므로 매우 높은 유연성(Flexibility)을 가진다.

(3) 형식을 갖춘 연산(Modal)

OpenGL 은 프로시저 콜의 제어 흐름에 따라 합당한 상태(State), 모드(Mode)를 유지하면서 출력요소들을 어떻게 렌더링 할 것인지를 결정한다. 각 출력요소 연산에 대한 정보를 모두 유지하지 않고, 전체적인 상태 제어로 렌더링을 수행한다. 만약, 수행이 병렬적으로 이루어진다면 상태에 대한 유지 능력이 문제점으로 부각되는데, 이러한 문제점을 해결하기 위해 OpenGL 의 상태변환은 반드시 직렬적으로 행하도록 제약하고 있다. 또 하나의 방법으로 이름지워진 상태(Named State)로 변환을 허락하는 방법이 있을 수 있는데, 상태변환에 대한 벡터를 유지보수해야 하는 단점을 가지고 있다. OpenGL 에서는 이름지워진 상태를 지원하되 모든 명령은 디스플레이 리스트안에서만 행하도록 한다.

OpenGL 에서는 변환행렬을 스택으로 유지하고 있으며 그 종류는 모델-관점 행렬(Model-view Matrix), 텍스쳐 행렬(Texture Matrix), 투상 행렬(Projection Matrix)의 세 종류가 있다. 임의의 한 행렬에 대한 수정 연산이 이루어 지면, 다른 행렬도 상황에 맞는 스택구조로 변환되게 된다. 행렬에 대한 모든 명령들은 같은 명령어로서 수행

가능하여 프로그래머가 쉽게 접근 할 수 있다. OpenGL 의 모든 매개변수(Parameter)들은 get 명령어로 얻을 수 있으며 다른 연산에 방해 없이 스택의 변수와 속성을 얻을 수 있다.

(4) 프레임버퍼(Framebuffer)

OpenGL 명령어의 대부분은 그래픽 하드웨어에 프레임버퍼 기능을 요구하고 있다. 이러한 요구사항은 현재의 모든 상호작용적 그래픽스 응용프로그램들이 프레임버퍼를 가진 시스템에서 수행되고 있으므로 합당한 것으로 보여진다.

프레임 버퍼를 이용하면 한 시점에서 렌더링된 장면이 약간의 다른 시각으로 보여지게 된 경우, 두 프레임 버퍼의 평균을 사용하여 쉽게 장면을 표현할 수 있는 장점을 가지고 있다. 이와 같이 출력요소들에 대해 몇번의 렌더링이 요구될 경우 중복통과 알고리즘(Multipass Algorithm)을 수행하여 우수한 성능을 나타낸다. 매개변수에 대한 작은 변환은 렌더링 과정을 통과하며, 변환된 값들은 상수로 유지하고 있어 변환된 값이 유실되는 부작용을 막고 있다.

OpenGL 의 모든 함수들은 상태를 ON/OFF 시키는 형태로 구현되어 있으며 특정한 매개변수들을 변환시킴으로써 연산을 수행한다. 그 이유는 임의의 프로그래밍 언어를 지원할 경우 하드웨어와 매우 밀접한 관계를 갖는 API 가 구현되게 되며 이에 따른 성능의 성취율이 하드웨어에 의존하게 되므로, OpenGL 의 경우 제어에 관한 특정한 프로그래밍 언어를 제공하지 않는다.

OpenGL 의 명령 수행은 파이프라인으로 수행된다. 짧은 시간안에 복잡한 장면을 그려내기 위해서는 파이프라인 마지막 단계에서 많은 양의 단편들을 통과시킴으로써 프레임 버퍼상에 연산을 제한시킨다. 그러기 위해서는 수행연산들의 적당한 형태의 조합을 이루어야 하는데, OpenGL 에서는 음과 같은 단편들에 대한 연산을 제

공하고 있다.

□ 알파 블렌딩(Alpha blending) : 단편의 색깔은 알파값을 이용하여 해당되는 프레임 버퍼상의 색깔과 조합하게 된다.

□ 깊이 테스트(Depth test) : 단편과 관계된 깊이 값은 기존 프레임 버퍼상의 깊이 값과 비교를 통하여 수정되거나 유지된다.

□ 틀종이 테스트(Stencil test) : 프레임 버퍼에 참조되는 모든 값들은 기존의 프레임 버퍼상에 존재하는 값들과 비교하여 참조되거나 무시되어진다.

알파 블렌딩은 배경색깔이 안티엘리어싱을 구현하는데 있어 매우 유용하게 쓰인다. OpenGL 의 파이프라인에서는 점, 선, 다각형, 점들이 모두 꼭지점(Vertex)의 형태로 취급되므로 레스터라이저에 의해 이미지가 생성될 경우, 단순하며 통합된 형태로 연산을 수행할 수 있다.

(5) 직접 모드(Immediate Mode)와 디스플레이 리스트(Display Lists)

OpenGL 명령어는 서버가 요구를 받는 즉시 수행되는 직접모드 형태로 구현되어 있다. 점에 대한 처리를 하는데 있어 모든 점이 명세되지 않아도 출력요소를 명세화 하기도 한다. 이러한 직접 모드 형태의 명령어를 지원하는 API 는 상호작용 요구를 갖는 응용프로그램 제작시 매우 유용하다. 만약 매개변수나 객체들이 수정되지 않을 경우, 의미없는 연산이 연속으로 수행될 수 있는 단점을 갖고 있다. OpenGL 은 디스플에이 리스트를 갖고 있어 명령어의 흐름을 제어함으로써 이 문제를 해결한다. 디스플레이 리스트는 응용프로그램에서 정의한 함수 고유명으로 관리하여 서버에 저장된다. 만약 클라이언트에서 같은 명령을 수행한다면 디스플레이 리스트를 살펴봄으로써 여러번의 수행을 막을 수 있다.

텍스춰 맵핑과 같은 명령이 디스플레이 리스트에 첨가될 경우, 크기가 큰 텍스춰에 대한 자료의 유지 보수 능력을 감안하여, OpenGL 에서는 텍스춰에 대한 연산이 명세화되면 서버에 이를 복사함으로써 재호출될 경우 복사본을 사용함으로써 성능을 높이는 디스플레이 리스트 최적화를 구현한다.

OpenGL 에서는 디스플레이 리스트 수정 연산을 중요시 여기지 않으며, 다만 디스플레이 리스트에 대한 재정의만을 허락한다. 이는 디스플레이 리스트에 대한 메모리 관리의 부담을 덜어주기 위한 방법이다. 디스플레이 실행과 관련된 모드들은 시스템 자체적으로 저장하지 않으며, 디스플레이 리스트에 대한 정보는 특정한 명령어를 통해서만 저장가능하다. 이러한 간단한 제어 형태는 외부적으로 부터 영향을 받지 않는 디스플레이 계층구조를 만들게 된다. 하지만 디스플레이 리스트를 병렬적으로 처리하는데 있어서는 문제점을 안고 있다.

(6) X 와의 결합

X 는 윈도우를 만들거나 2 차원 객체들을 윈도우 상에 그리기 위한 네트웍크 프로토콜과 인터페이스를 제공한다. OpenGL 은 X 와의 결합환경을 제공하기 위해 X 확장자인 GLX 를 제공한다. GLX 는 X 를 호출할 수 있는 내장된 형태의 함수들로 구성되고 있다. 아래

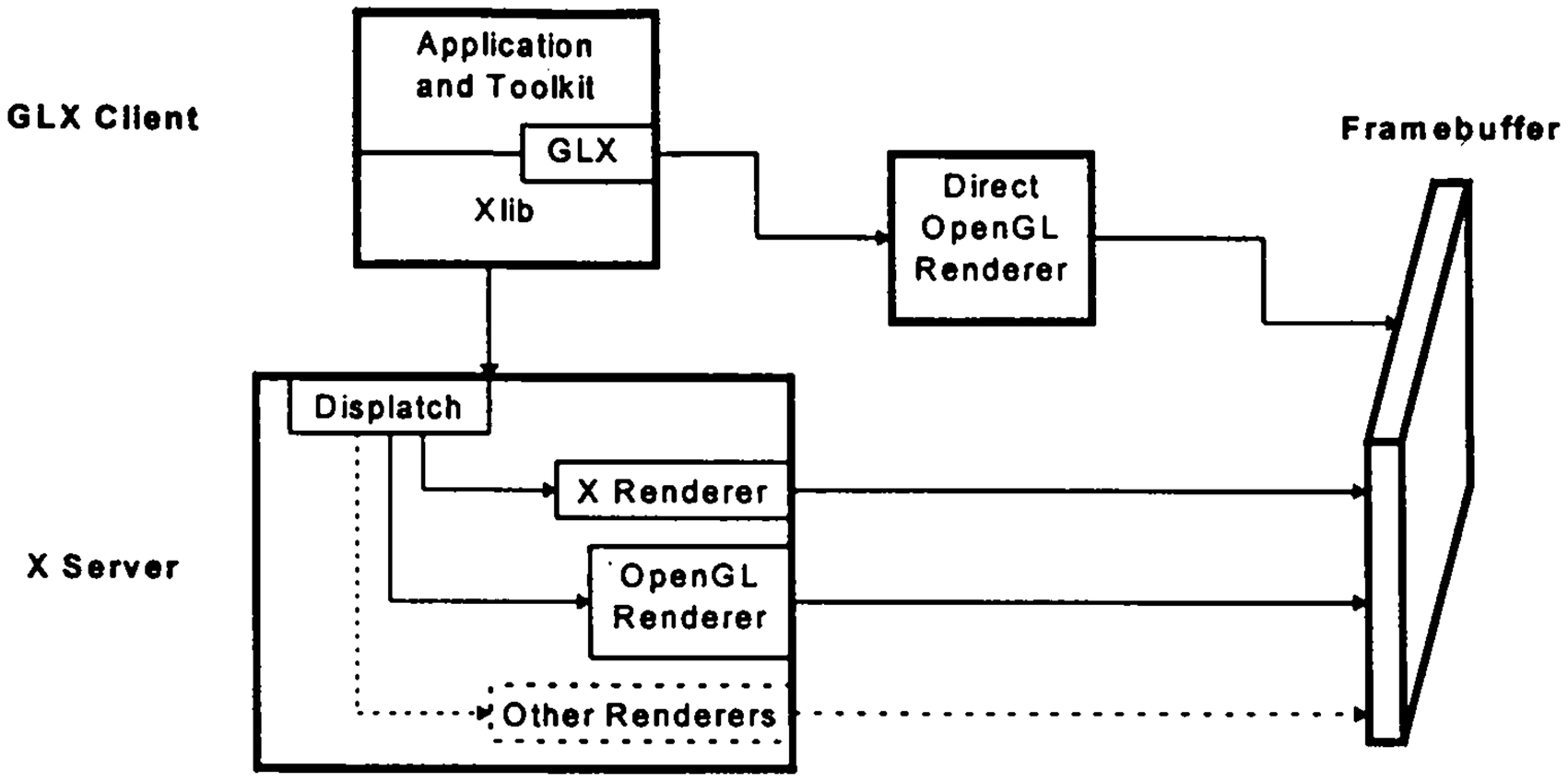


그림 19은 GLX를 사용하고 있는 응용프로그램의 개념도를 보여준다.

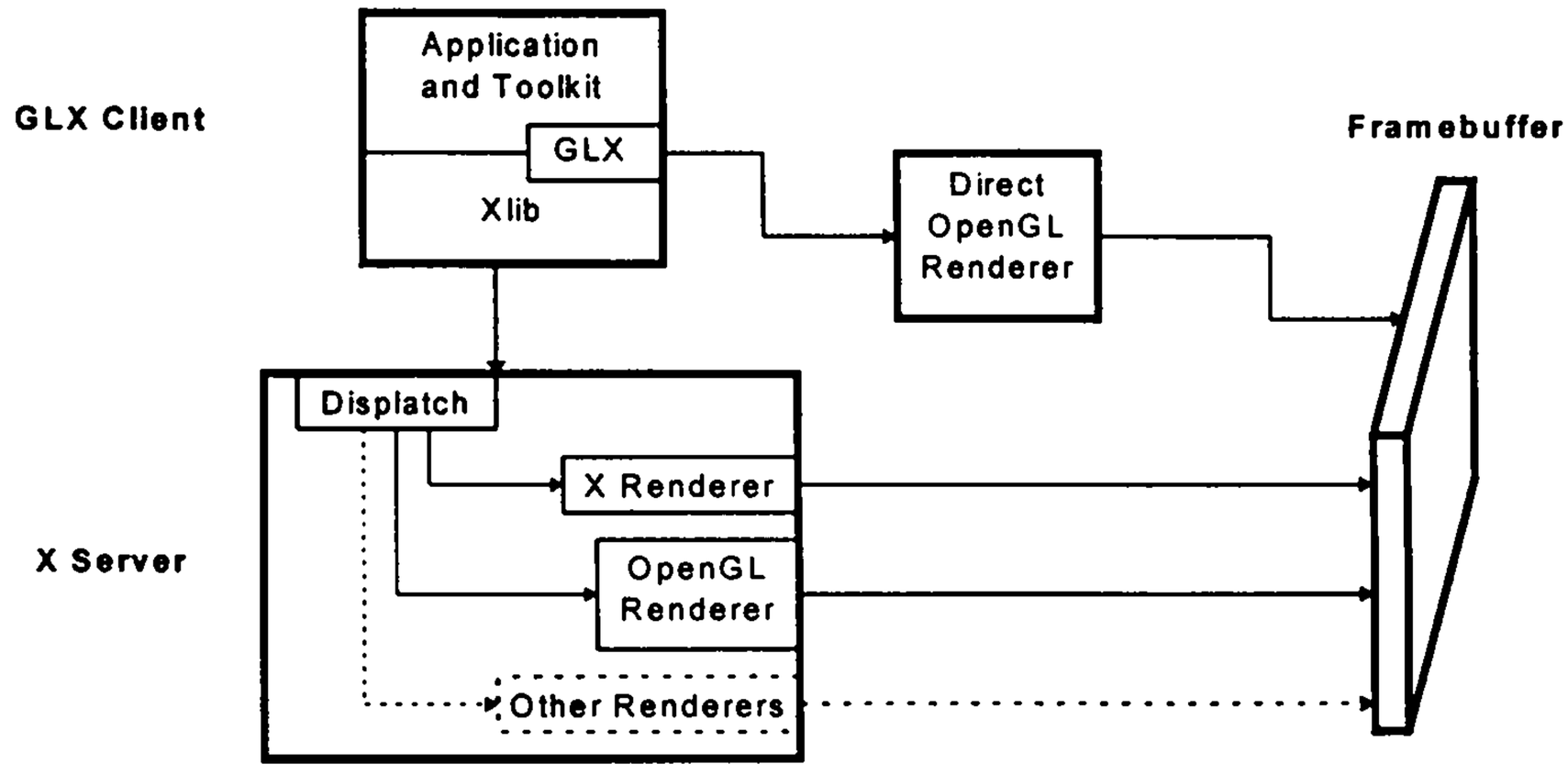


그림 19 GLX client, Server, and OpenGL renderers

나. 실시간 압연공정 시뮬레이터의 모델링과 가시화

실시간 압연공정 시뮬레이터에서는 시뮬레이션하고자 하는 실제 상황과 같게 가상환경을 모델링함으로써 사용자에게 친숙감과 이해의 정도를 높일 수 있다. 실시간 압연공정 시뮬레이터에서 사용하는 가상환경은 그림 20에 나타나 있다. 가상물체를 모델링함에 있어서 이미지의 해상도를 높이기 위해 많은 다각형(Polygon)을

사용할 수록 이미지를 랜더링하는데 많은 시간이 소비되어 실시간 상호작용을 할 수 없게 된다. 따라서 실시간 상호작용을 하기 위해서는 적절한 질의 이미지 해상도로 가상환경을 구성하는 것이 필요하다.

가상환경에는 가상물체뿐만 아니라 가상도구도 존재한다. 가상도구를 실제 도구와 유사하게 모델링함으로써 사용자의 직관과 이해를 높일 수 있다. 그러나 가상도구를 실제 도구와 유사하게 모델링할 경우 랜더링 시간이 많이 소비되고 실제에 존재하지 않는 도구의 경우에는 모델링하는데 어려움을 가질 수 있다.

가상환경을 가시화하는데 있어서 적절한 조명모델과 음영처리는 이미지를 사실적으로 만들기 때문에 꼭 필요하다. 그러나 필요 이상의 음영처리는 실시간 상호작용을 할 수 없게 하므로 좋지 않다.

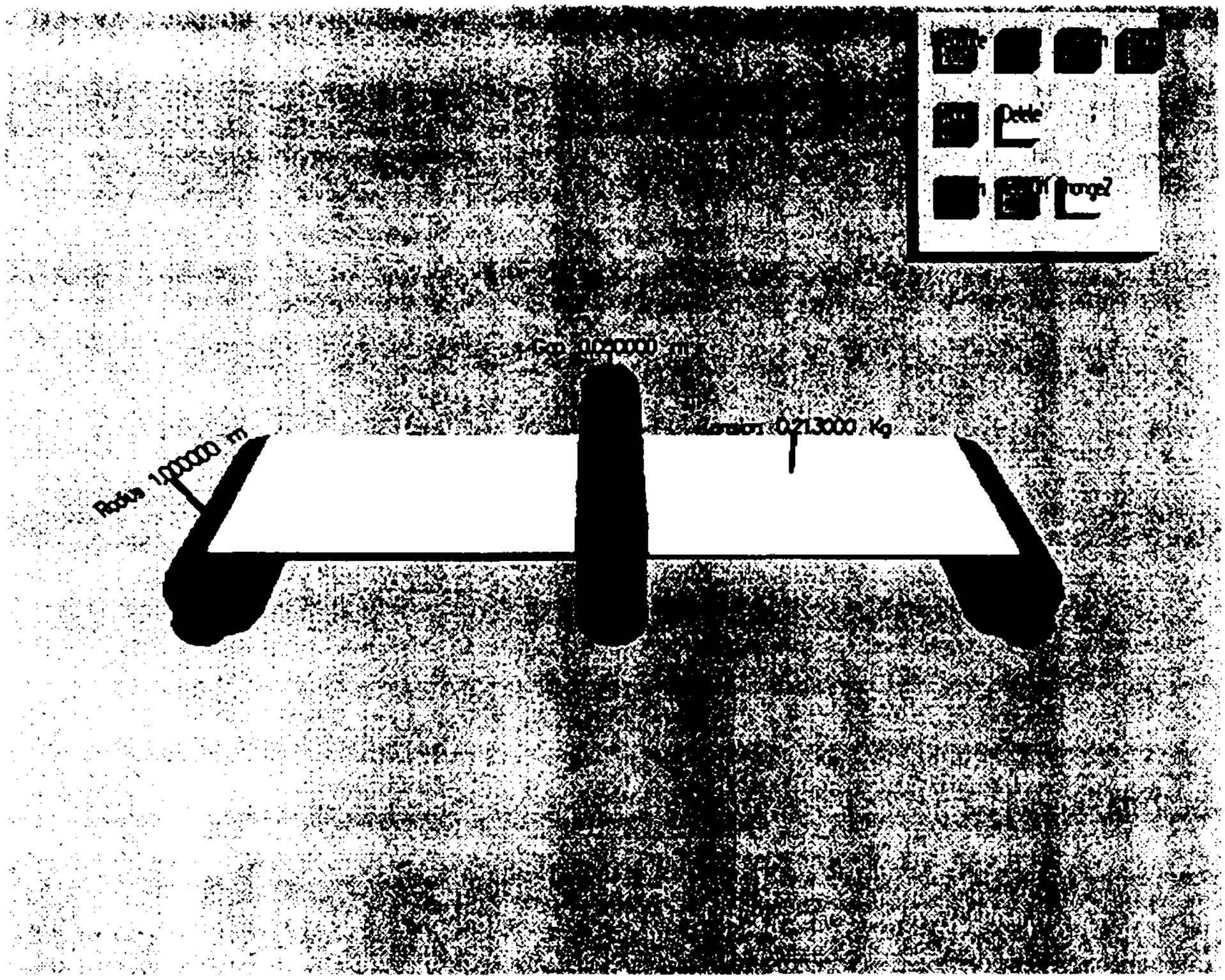


그림 20 실시간 압연공정 시뮬레이터의 가상환경

3. 애니메이션(Animation)

실시간 시뮬레이터를 통해서 나오는 데이터에 따라 가상환경을 실시간에 에니메이션해 줌으로써 사용자의 이해를 증진시킬 수 있다.

본 연구에서는 압연공정 시뮬레이터를 통해서 나오는 데이터에 맞게 가상환경을 그려준다. 즉 입측롤의 두께가 변하면 변한 두께에 맞게 가상환경을 그려주고 속도가 변하면 변한 속도에 맞게 입측롤을 회전시켜 준다. 실제 압연공정과 유사하게 가상환경을 구성하고 시뮬레이션된 결과에 맞게 가상환경을 재구성시켜줌으로써 사용자는 압연공정에 대한 이해도를 높일 수 있다. 또한 가상도구를 이용하여 관심이

물체의 정보를 원하는 형태로 실시간에 에니메이션할 수 있다.

3 절 가상메뉴(Virtual Menu)

가상환경과 상호작용을 하기위한 인터페이스는 과학과 의학분야의 가시화(Visualization), 로봇의 원격조정, 자동차 시뮬레이션, 건축, 소프트웨어 디자인 등 여러분야에서 이용되고 있다. 가상환경의 사용자들은 가상환경뿐만 아니라 가상환경에 있는 물체들과 상호작용을 하기 위해서 다양한 방법들을 필요로 한다. 이러한 상호작용의 방법은 여러 응용분야에서 많은 발전을 이루어 왔으며, 그 중 메뉴 방식이 가장 많이 이용되고 있다.

일반적인 메뉴방식을 3차원으로 이루어진 가상환경에서의 사용자 인터페이스를 위해 사용하면 가상환경에서의 특별한 상황 때문에 좋지 않은 사용자 인터페이스가 된다. 따라서 본 보고서에서는 가상환경에서의 메뉴의 구현에 대해서 기술한다.

1. 가상메뉴의 특징

메뉴는 사용하고자 하는 메뉴 항목들을 선택하기만 하면 되기 때문에 복잡한 명령어들을 기억하고 학습해야하는 불편함을 없애주고, 잘못 입력할 가능성을 줄여준다. 이러한 메뉴는 논리적으로 비슷한 항목들을 하나의 그룹으로 묶고, 모든 항목들을 다 포함하며 각각의 메뉴 항목들의 의미가 겹쳐지지 않도록 의미있고 체계적인 구조를 이루어야 한다. 또한 메뉴의 제목이나 메뉴 항목의 이름을 의미있게 만들어서 사용자가 쉽게 배워서 사용할 수 있게 해야 하며 이외에도 메뉴 항목들의 표시 순서, 자주 사용하는 메뉴 항목들의 지름길(shortcut) 등을 고려해서 메뉴를 구현해야 한다.

이와같은 일반적인 메뉴의 특징과 요구사항들은 가상환경 속의 가상메뉴에도 적

용되지만 일반적인 메뉴와 다른점을 나타내면 다음과 같다.

□ 일반적인 메뉴는 사용자앞에 있는 컴퓨터 화면에 2 차원적으로 나타나지만 가상메뉴는 사용자의 위치와 방향에 따라서 가상환경이라는 3 차원 공간에 나타난다. 이러한 가상환경에서의 공간적인 특징으로 인해서 가상환경에서 사용자와 가상메뉴 사이에 어떤 물체가 존재할 수도 있고 사용자와 가상메뉴 사이의 거리가 너무 멀거나, 가까워서 사용자가 가상메뉴를 보고 읽는데 어려움이 있을 수도 있다. 또한 이러한 3 차원 공간상의 어려움은 가상메뉴를 사용자가 이동시키거나 회전시킬 수 있는 경우 가상메뉴를 잘못 이동시켜서 적당한 위치에 다시 놓이게 하는데 어려움이 있을 수 있다.

□ 가상환경에 있는 가상메뉴는 평평한 2 차원적인 물체에 제한되지 않는다. 일반적인 메뉴는 보통 사각형의 형태에 문자들로 이루어진 메뉴항목들의 순차적인 리스트이다. 그러나 가상메뉴의 형태는 일반적인 메뉴 형태뿐만 아니라 그외의 3 차원 공간상에서 구현할 수 있는 모든 형태로 나타낼 수 있다. 예를 들면 구면체(Sphere)를 새로 생성시키는 메뉴항목을 나타내기 위해서 2 차원적인 메뉴에서는 문자로 나타내든지 구면체를 표시하는 아이콘으로 나타낼 수 있다. 이에 반해 가상메뉴에서는 2 차원적인 메뉴방식으로 나타낼 수 있을 뿐만 아니라 구면체를 생성시키는 메뉴항목을 나타내기 위해서 구면체를 나타냄으로써 사용자의 메뉴에 대한 이해를 증진시킬 수 있다. 즉, 가상메뉴에서는 메뉴항목을 문자로 나타낼 수 있을 뿐만 아니라 물체로도 나타낼 수 있다.

□ 변형된 가상메뉴의 형태로 가상공간상에 존재하는 3 차원 위젯이 있다. 3 차원 위젯을 사용함으로써 사용자는 명령을 직관적이고 직접적으로 가상환경에 줄 수가 있다. 3 차원 위젯에 대해서는 5 절에 자세히 기술한다.

- 가상메뉴와 사용자와 상호작용하는 방법이다. 일반적인 메뉴에서 사용자와의 상호작용 장치로는 마우스외에 응용분야에 따라서 키보드, Light Pen, Touch Sensitive Screen, 조이스틱등이 사용되고 있다. 이러한 상호작용 장치들은 사용자들이 3 차원 공간인 가상환경에 있는 물체와 상호작용하기에는 불편하고 자연스럽지 못하다. 따라서, 사용자가 가상환경에서 자연스럽게 상호작용하기 위해서는 3 차원 공간상에서 사용자의 위치와 방향, 행동 등을 인식할 수 있는 DataGlove 와 같은 장치나 Polhemus Fasktrak 과 같은 3 차원 추적장치들이 필요하다. 그러나 이러한 장치들은 고가의 장비이고 널리 사용되고 있지 않으므로 입력장치로써 부적절하다. 본 과제에서는 마우스를 이용한 3 차원 공간상에서 상호작용을 하는 방법을 개발하여 3 차원 입력장치들을 사용하지 않고 상호작용한다. 이러한 방법에 대해서는 5 절에 자세히 기술한다.

가상메뉴를 구현하는 메뉴의 형태(Paradigm)는 여러 가지 방법으로 정의할 수 있으며 메뉴의 형태에 따라 메뉴 동작이 다르게 정의될 수 있다. 여러 형태의 메뉴에서 많이 사용되는 메뉴의 동작들은 다음과 같다.

- Invocation - 메뉴를 처음에 어떻게 나타나게 할 것인가?
- Location - 메뉴가 어디에 위치하는가?
- Reference Frame - 메뉴를 이동할 수 있는가?
- Cursor - 사용자가 가리키는 곳을 나타내는 커서가 있는가?
- Highlighting - 다른 항목들과 구분해서 어떻게 강조되는가?
- Selection - 강조된 메뉴 항목을 어떻게 선택할 것인가?
- Removal - 메뉴 사용 후에 메뉴상태를 어떻게 할 것인가?

이러한 메뉴의 동작들을 원하는 가상 메뉴형태에 따라 적절히 정의하여 사용한다.

실시간 압연공정 시뮬레이터의 가시화를 위한 가상메뉴는 비교적 단순하기 때문에 본 단원에서는 가상공간상에서 3차원 메뉴인 가상메뉴를 구현하는데 필요한 기반 기술에 대해서 자세히 기술한 후 실시간 압연공정 시뮬레이터의 가시화를 위한 가상메뉴의 구현에 대해서 기술한다.

2. 가상메뉴를 위한 기반기술

가상메뉴를 위한 기반기술로는 가상메뉴를 가상환경속에서 구현하고 구현된 가상메뉴를 선택, 이동, 회전시키는 기술이 있다.

가. 가상메뉴의 모델링

가상메뉴는 일반적인 2차원 메뉴의 특성뿐만 아니라 가상환경속에서 동작한다는 특성을 첨가해서 구현해야 된다. 가상메뉴가 가져야 하는 일반적인 2차원 메뉴의 특성으로는 다음과 같은 것이 있다.

- □ 항목을 선택하기 위한 버튼 : 가상메뉴상의 특정 항목을 선택하기 위해서는 이를 위한 버튼이 필요하다. 버튼의 종류로 푸시버튼(PushButton), 토글버튼(ToggleButton), 라디오버튼(RadioButton)등이 있다. 버튼을 선택했을 경우에 그 버튼의 기능에 맞게 새로운 메뉴가 나타난다든지 값이 입력되는 등의 정해진 기능을 수행하는 콜백(Callback)함수를 제공해 주어야 한다.

- □ 항목을 나타내기 위한 레이블(Label) : 가상메뉴의 각각의 항목을 구분할 수 있는 레이블이 필요하다. 이러한 가상메뉴상의 레이블은 3 차원 공간상에서 나타난다는 점을 고려해야 한다. 2 차원 공간에서는 글자를 나타내기 위해서

비트맵(Bitmap)을 사용한다. 그러나 비트맵은 확대나 축소가 되지 않으므로 3차원 공간상에서 이동과 회전을 하여야하는 가상메뉴에는 적절하지 못하다. 가상공간에서는 확대와 축소가 되는 스트로크(Stroke)글자나 각각의 글자를 3차원으로 모델링하여서 나타내어야 한다. 글자를 3 차원으로 모델링하여 나타낼 경우 사용자가 상대적으로 인식하기는 쉽지만 많은 수의 다각형(Polygon)이 필요하게 되며 결과적으로 전체화면을 랜더링(Rendering)하는데 많은 양의 시간을 소비하게 된다. 따라서 가상메뉴의 용도로는 스트로크 글자가 가장 적절하다.

□ 값을 입력하기 위한 스케일 위젯(Scale Widget) : 가상환경에서 값을 입력하기 위해서 키보드를 사용하는 것은 비효율적이다. 사용자가 스케일 위젯을 선택한 후 마우스를 드래깅(Dragging)함으로써 최대값과 최소값 사이의 원하는 값을 입력할 수 있다. 그러나 이러한 방법은 정수를 입력하기에는 좋지만 실수를 입력하기에는 부적절하다는 단점이 있다. 따라서 가상 키보드와 같은 입력 수단의 개발이 필요하다.

가상메뉴는 이와같은 2차원적인 특성뿐만 아니라 가상공간이라는 3차원적인 특성을 이용함으로써 사용자가 이해하기 쉽게 효과적으로 사용할 수 있는 메뉴가 될 수 있다. 가상메뉴가 가질 수 있는 3차원적인 특성으로는 다음과 같다.

□ 메뉴항목을 3 차원으로 구현 : 메뉴를 가상공간에 나타내기 위해서는 입방체로 나타내어야 한다. 이러한 것 외에도 메뉴항목을 3 차원으로 모델링하고 모델링된 항목에 레이블을 첨가함으로써 사용자는 한 눈에 메뉴항목을 쉽게 빨리 이해할 수 있게 된다. 예를들면 물체를 모델링하는 프로그램에서 사용자는 삼각뿔, 실린더, 구면체, 입방체 등을 만들기를 원할 것이다. 이 경우에 메뉴항목을 단순한 2 차원적인 버튼으로 나타내는 것보다는 실제로 만들고자하는 물체인 삼각뿔, 실린더, 구면체, 입방체 등을 직접 메뉴항목으로 나타냄으

로써 사용자의 이해와 사용의 편의성을 증진시킬 수 있다.

□ 3 차원 위젯(Widget)으로 구현된 메뉴 : 3 차원 위젯은 3 차원 기하와 행위를 가지는 일종의 변형된 메뉴로 생각할 수 있다. 사용자는 3 차원 공간상에서 모델링된 3 차원 위젯을 다룸으로써 원하는 명령을 직관적이고 직접적으로 줄 수 있다. 예를 들면 사용자는 시점을 바꾸기 위해 복잡한 메뉴항목으로 구성된 메뉴를 선택함으로써 명령을 줄 수도 있지만 가상공간상에 구현된 3 차원 위젯인 구면체를 회전시킴으로써 시점을 바꾸기 위한 명령을 직관적이고 직접적으로 줄 수 있다. 이러한 3 차원 위젯의 구체적인 예와 특징에 대해서는 5 절에 자세히 설명한다.

나. 가상메뉴의 선택, 이동, 회전

모델링된 가상메뉴를 이용하기 위해서는 가상메뉴의 항목을 선택할 수 있는 방법이 제공되어야 한다. 가상메뉴도 가상환경속에서 가상물체처럼 모델링되기 때문에 가상메뉴를 선택하는 것은 가상환경속에서의 가상물체를 선택하는 것과 유사하다. 본 연구에서는 가상메뉴의 항목을 선택하기 위해서 가상물체를 선택하는 방법처럼 OpenGL의 선택버퍼를 이용했다.

가상메뉴의 항목을 선택하였을 경우 사용자가 선택된 항목을 알아보기 쉽게 하기 위한 피드백(Feedback)이 제공되어야 한다. 피드백의 종류로는 선택된 항목만 크게 보여준다든지 선택된 항목의 색깔을 바꾸는 등의 여러가지가 있을 수 있다.

가상메뉴는 가상공간상에서 존재하는 것이므로 가상공간상에 위치와 방향을 가지게 된다. 따라서 사용자는 가상공간에서 가상물체를 다루듯이 원하는 위치와 방향에 가상메뉴를 놓고 상호작용을 할 수 있다. 이러한 가상메뉴의 이동과 회전은 사용자에게 여분의 자유를 주지만 많은 경우에 가상메뉴를 움직이는 것이 바람직하

지 않을 경우가 있다. 특히 2차원적인 메뉴를 가상공간에 3차원으로 그대로 나타내었을 경우 이러한 여분의 자유도는 사용자에게 혼란을 초래할 수 있다. 이러한 경우에는 사용자의 앞의 특정한 위치에 가상메뉴를 고정시킴으로써 사용자의 운항과 상관없이 가상메뉴가 항상 일정한 위치에 나타나게 할 수 있다.

3. 실시간 압연공정 시뮬레이터의 가시화를 위한 가상메뉴

실시간 압연공정 시뮬레이터의 가시화를 위한 가상메뉴는 실시간 시뮬레이션 동안 사용자가 취할 수 있는 모든 항목들을 포함해야 한다. 이러한 항목에는 실시간 시뮬레이션 동안 원하는 부분의 원하는 종류의 값을 가상공간상에 나타내기 위한 가상도구를 선택하는 부분, 가상도구를 적용하고 적용된 가상도구를 없애기 위한 부분 그리고 운항(Navigation)을 위한 3차원 위젯을 가상환경속에 나타내고 없애는 등의 운항과 관련된 부분으로 나타낼 수 있다. 또한 가상메뉴의 변형된 형태로써 운항을 하기 위한 3차원 위젯이 있다. 이러한 3차원 위젯에 대해서는 5 절에 자세히 기술한다.

시뮬레이션동안 사용자는 계속 가상메뉴와 상호작용을 하기 때문에 가상메뉴는 가상공간상에 위치와 방향을 바꿈으로써 생기는 혼란을 없애고 계속된 상호작용을 위해서 사용자의 시점에서 일정한 거리와 방향을 두고 나타나야 한다.

실시간 압연공정 시뮬레이터의 가상메뉴의 항목을 살펴보면 다음과 같다.

□ 가상도구 : 시뮬레이션 동안에 변하는 입측롤, 출측롤, 작업롤의 장력, 속도, 반경 등의 정확한 수치적 데이터를 보기 위한 메뉴로 Measure, Speed, Tension, Gap 등의 메뉴 항목이 있다. 이러한 가상메뉴는 각각의 항목을 해당되는 3 차원 물체로 모델링함으로써 사용자의 이해를 증진시킬 수 있다. 본 연구에서는 구현의 편의성을 위해서 입방체로 구현하였다. 향후 연구에서는 실제 세

계에 존재하는 가상도구에 대해서는 그대로 모델링하고 존재하지 않는 가상도구에 대해서는 가장 적절한 물체로 모델링할 계획이다.

□ 적용(Apply), 해체(Delete) : 선택된 메뉴 항목을 적용시켜서 가상공간상에 해당되는 가상도구가 나타나게 하고, 필요없는 가상도구를 없애기 위한 메뉴 항목이다.

□ 운항과 관련된 메뉴 항목 : 운항을 위한 3차원 위젯을 가상공간에 나타내고 동작할 수 있도록 해 주는 부분, 운항 후에 사용되지 않을 경우 화면에서 없애 주는 부분, 그리고 운항시에 Z 방향을 바꾸어 주는 부분으로 나눌 수 있다.

실시간 압연공정 시뮬레이터를 위한 가상메뉴를 그림 21에 나타내었다. 이러한 가상메뉴의 항목을 선택했을 경우 현재 선택된 항목을 나타내기 위한 피드백(Feedback)으로 선택된 항목을 붉은 색으로 강조시킨다. 메뉴판을 마우스로 선택하면 붉은 색으로 강조된 선택된 항목이 없어진다.

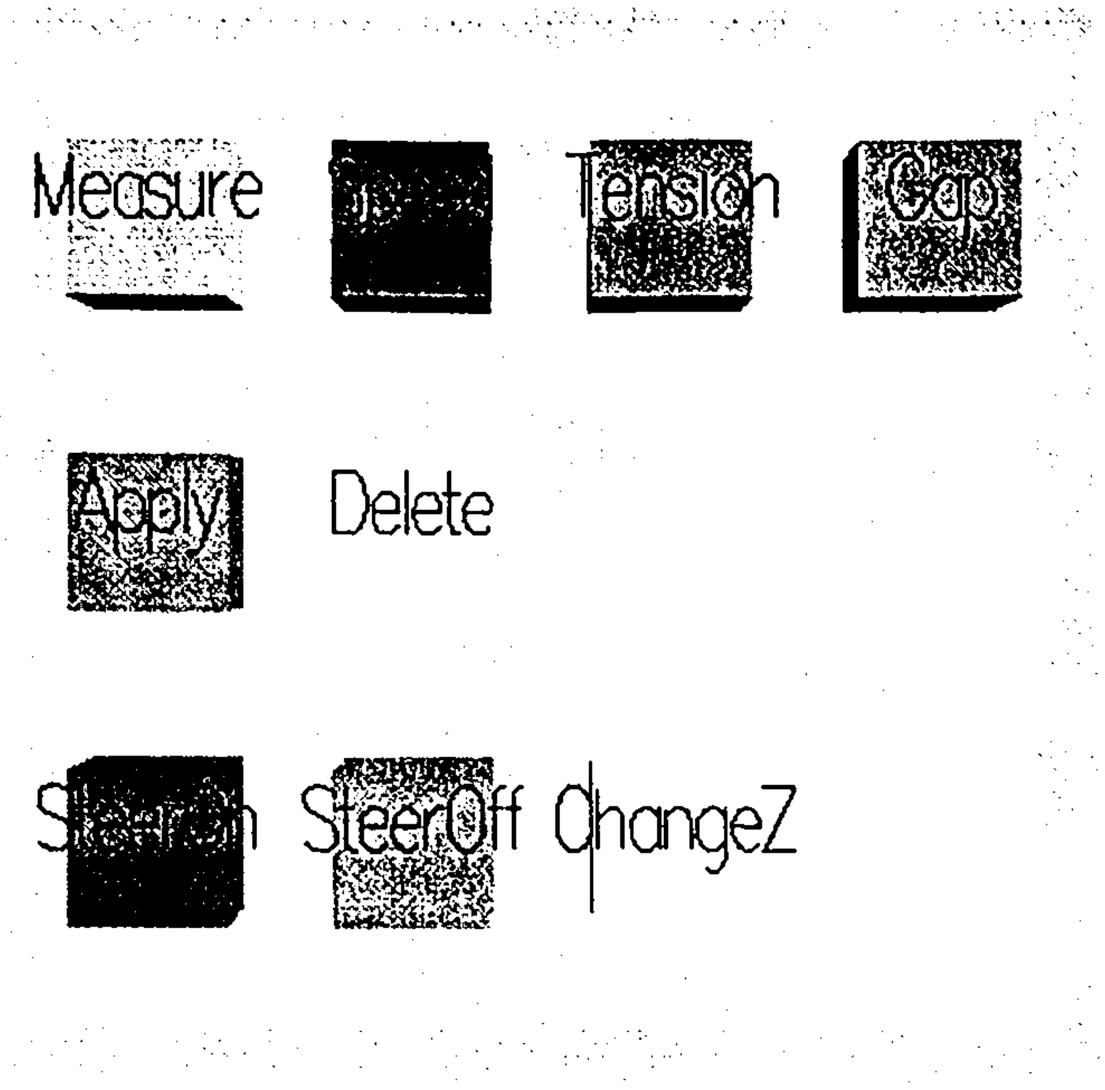


그림 21 실시간 압연공정 시뮬레이터를 위한 가상메뉴

4 절 가상도구(Virtual Instrument)

실시간 시뮬레이터를 통해서 나오는 모든 정보를 가상환경에 그대로 실시간에 보여준다면 사용자는 너무나 많은 양의 정보 때문에 혼란에 빠질 것이다. 히스토리 다이어그램(History Diagram)이나 칸트 차트(Kant Chart) 등을 이용하여 나타낼 수도 있지만 사용자가 원하는 정보만을 얻고자 할 경우 또는 원하는 정보의 정확한 값을 알고자 하는 경우에는 부적절하다. 원하는 정보만을 원하는 방법으로 사용자에게 제공해 줄 수 있는 방법으로 본 연구에서는 가상도구(Virtual Instrument)를 이용한다.

1. 가상도구의 특징

실시간 시뮬레이터를 통해서 나온 결과를 그대로 보기 보다는 사용자는 원하는 정보를 원하는 형태로 보기를 원할 것이다. 사용자는 원하는 정보를 얻기 위해서

가상환경속에서 정보를 얻고자 하는 가상물체와 그 정보를 어떤 형태로 보고자하는지를 나타내는 가상도구를 선택하여 해당되는 정보를 원하는 형태로 얻을 수 있다.

예를들면 사용자가 실시간 압연공정에서 입측롤의 속도를 숫자로 알고 싶다면 가상물체인 입측롤을 선택하고 숫자를 얻기 위한 가상도구를 사용한다. 또한 속도계기판의 형태로 보고 싶다면 가상물체인 입측롤을 선택한 후 속도 계기판의 형태로 보여 줄 수 있는 가상도구를 사용하면 된다. 이렇게 얻어진 정보가 더 이상 필요하지 않을 경우 사용자는 가상환경상에 있는 가상도구를 없앰으로써 사용자는 관심이 있는 정보만을 원하는 형태로 가상공간에 나타낼 수 있게 된다.

가상도구를 사용함으로써 사용자는 시뮬레이션의 결과를 보다 직관적이고 쉽게 이해할 수 있다. 사용자는 가상도구를 봄으로써 현재 가시화되고 있는 정보가 무엇인지를 알 수 있고, 원하는 가상도구를 사용자의 가까이에 놓든지 해당되는 가상물체의 주위에 놓음으로써 사용자의 이해를 증진시킬 수 있다.

이러한 가상도구의 특징을 나타내면 다음과 같다.

□ 가상도구의 기하(Geometry) : 가상도구는 가상환경에서 가상물체처럼 기하를 가진다. 가상도구의 기하는 실제세계에 있는 도구를 나타낼 경우에는 실제 도구를 가상환경에서 그대로 모델링 한 것이 될 수 있다. 가상도구가 실제세계에 존재하지 않는 것일 경우 사용자가 가상도구의 기능을 직관적으로 알 수 있도록 가상도구의 기하를 모델링하는 것이 좋다. 가상도구의 기하가 사용자에게 친밀한 것일수록 가상도구의 기능은 사용자에게 명백해지고 사용자의 이해를 증진시킬 수 있다.

□ 정보의 변형(Transformation) 및 여과(Filtering) : 가상도구는 시뮬레이션의 결과로 나오는 정보를 변형해서 사용자가 원하는 형태로 만들어 준다. 사용자는

가상도구를 사용하여 정보를 숫자, 그래프, 소리, 색, 3 차원 물체 등의 형태로 변형해서 볼 수 있다. 이렇게 정보를 변형해서 보여 줌으로써 사용자의 정보 습득은 훨씬 빨라진다. 또한 시뮬레이션의 결과로 나오는 정보를 여과함으써 필요한 정보만을 보여줄 수 있다. 이러한 여과 기능의 예로써 샘플링률 (Sampling Rate)을 바꾸는 것을 들 수 있다. 이때 샘플링률은 실시간 가시화 시스템의 샘플링률보다 커야 한다.

□ 정보의 시간성 : 가상도구는 정보를 실시간에 그대로 보여줄 뿐만 아니라 특정시간 간격동안의 정보를 보여 줄 수 있다. 예를 들면 사용자가 10 초 동안의 속도를 보고 싶을 경우 히스토리 다이어그램으로 볼 수 있다. 10 초 동안의 속도의 평균을 보고 싶을 경우 10 초 동안 평균을 낸 후 정보를 숫자나 그래프, 색등으로 변형한 후 볼 수도 있다. 사용자는 원하는 시간동안의 정보를 원하는 가상도구를 사용하여 변형한 후 볼 수 있다.

□ 정보의 저장 및 재생 : 사용자가 필요한 정보를 저장을 위한 가상도구를 사용하여 저장한 후 필요시에 재생해서 가상도구를 이용하여 변형해서 볼 수 있다. 이러한 저장과 재생기능은 정보를 분석하기 위해서는 꼭 필요한 기능이다.

□ 가상도구와의 상호작용(Interaction) : 가상도구도 가상환경속에서 존재하므로 가상도구와 상호작용하는 방법은 가상환경에서 가상물체와 상호작용하는 방법과 동일하다. 상호작용하는 방법에 대해서는 5 절에 기술한다.

□ 가상도구를 가상환경에서 사용하기 위해서는 다음과 같은 단계가 필요하다.

□ 가상물체의 선택 : 먼저 가상도구를 적용하기 위해서는 원하는 가상물체를 선택하여야 한다. 선택된 가상물체의 종류에 따라서 사용할 수 있는 가상도구가 달라진다. 선택된 가상물체에 따라서 사용할 수 있는 가상도구만을 나

타내 줌으로써 사용자는 혼란없이 쉽게 사용할 수 있다.

□ 가상도구의 선택 : 원하는 가상물체를 선택한 후 사용자는 사용가능한 가상도구 중에서 원하는 형태로 정보를 얻기 위한 가상도구를 선택하여야 한다. 하나의 정보를 여러형태로 보기 위해서 여러개의 가상도구를 선택할 수 있다. 가상도구를 선택할 시에 3 절에서 기술한 가상메뉴를 사용하면 용이하다.

□ 가상도구의 위치, 방향 선택 : 가상도구를 선택한 후 선택된 가상도구를 가상환경에서 원하는 위치와 방향에 놓을 수 있다. 가상도구를 해당되는 가상물체의 주위에 놓을 수도 있고 몇 개의 가상도구를 서로 비교, 분석하기 위해서 나란히 사용자의 시점주위에 놓을 수도 있다. 이 때 사용자가 가상도구를 가장 잘 볼 수 있게 가상도구의 방향이 항상 사용자의 시점을 보도록 하면 좋다. 이렇게 가상도구의 위치와 방향을 바꾸는 방법은 가상물체와 상호작용할때와 동일하다.

2. 실시간 압연공정 시뮬레이터의 가시화를 위한 가상도구

실시간 압연공정 시뮬레이터의 가상물체인 입측롤, 출측롤, 작업롤 등을 선택한 후 사용자는 원하는 가상도구를 가상메뉴에서 선택하게 된다. 가상도구를 선택하면 선택된 가상도구가 해당되는 가상물체의 정해진 위치에 나타나게 된다. 실시간 압연공정 시뮬레이터를 위한 가상도구로는 Measure, Speed, Tension, Gap, Magnify등이 있다.

가. 가상도구

실시간 압연 공정 시뮬레이터를 위한 가상도구를 나열하면 다음과 같다.

□ Measure : 롤의 반지름이나 판의 두께와 같은 길이 정보를 나타내기 위한 가상도구

- Speed : 롤의 속도를 나타내기 위한 가상도구
- Tension : 입측과 출측 판의 장력을 나타내기 위한 가상도구
- Gap : 작업롤사이의 간격을 나타내기 위한 가상도구
- Magnify : 특정부분을 확대해서 보기 위한 가상도구

이러한 가상도구를 실제의 물체에 맞게 구현해야 되지만 구현의 편의성을 위해서 막대기로 구현하였다. 향후 연구에서 가상도구에 따른 가장 적절한 형태로 구현하겠다.

가상물체에 따른 사용 가능한 가상도구를 나타내면 다음과 같다.

- 입측롤 : 입측롤의 반지름을 측정하기 위한 Measure, 입측롤의 속도를 측정하기 위한 Speed
- 입측릴 : 입측릴의 반지름을 측정하기 위한 Measure
- 입측판 : 입측판의 두께를 측정하기 위한 Measure, 입측판의 장력을 측정하기 위한 Tension
- 출측롤 : 출측롤의 반지름을 측정하기 위한 Measure, 출측롤의 속도를 측정하기 위한 Speed
- 출측릴 : 출측릴의 반지름을 측정하기 위한 Measure
- 출측판 : 출측판의 두께를 측정하기 위한 Measure, 출측판의 장력을 측정하기 위한 Tension
- 작업롤 : 작업롤의 반지름을 측정하기 위한 Measure, 작업롤의 속도를 나타내기 위한 Speed, 작업롤사이의 간격을 나타내기 위한 Gap, 작업롤을 확대해서

보기 위한 Magnify

실시간 압연공정 시뮬레이터를 위한 가상도구는 그림 22에 나타나 있다. 실시간 압연공정을 위한 시뮬레이터의 가상도구는 숫자로 보이게 하는 것만 구현되었다. 향후 연구에서 그래프, 색, 3차원 물체등으로 나타낼 수 있도록 하겠다.

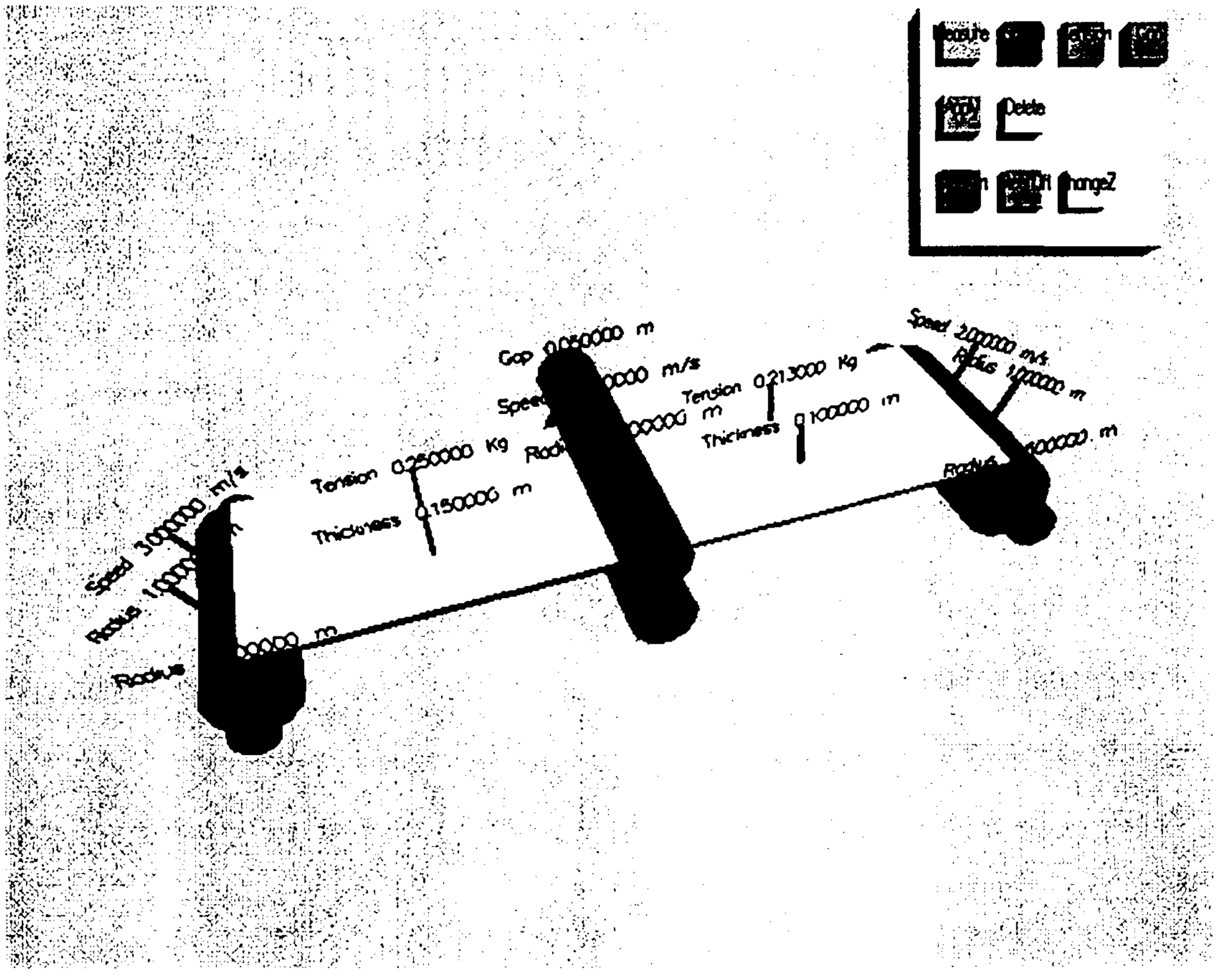


그림 22 실시간 압연공정 시뮬레이터를 위한 가상도구

나. 가상도구와의 상호작용

가상도구를 가상공간상의 특정한 위치에 놓기 위해서는 상호작용이 필요하다. 이러한 상호작용은 가상물체와의 상호작용과 유사하다. 그러나 실시간 압연공정을

위한 시뮬레이터와 같이 가상도구가 많지 않을 경우에는 가상도구의 위치를 해당되는 가상물체의 특정지역에 미리 정해 놓고 사용자가 가상도구를 사용했을 경우 그 위치에 나타내 주는 것이 좋다. 이렇게 함으로써 사용자는 가상환경에서 가상도구를 쉽게 인식할 수 있다.

5 절 사용자와의 상호작용(Interaction)

가상환경은 인간과 컴퓨터의 상호작용의 자연스러운 형태로써 많은 것을 할 수 있다는 것을 보여왔었다. 가상세계에서 사용자는 실제 세상에서 할 수 있는 것처럼 눈, 귀, 손을 사용할 수 있다. 즉, 사용자는 시점을 설정하기 위해서 머리를 움직이고 3차원 공간에서 나는 소리에 귀를 귀울이고, 가상물체를 잡기 위해서 손을 사용한다. 그러나 이러한 것을 구현하기 위해서 사용하는 HMD(Head Mounted Display)나 3차원 추적장치들은 일반적으로 널리 사용되고 있지 않다. 또한 일반적인 사용자는 HMD나 3차원 추적장치들을 사용한 상호작용에 친숙하지 못할 뿐만 아니라 이러한 장치들은 눈과 근육에 매우 피로를 주고 공간적, 시간적 해상도가 낮다.

따라서 본 연구에서는 일반적인 사용자에게 친숙하고 널리 사용되고 있는 마우스를 이용한 가상환경에서의 상호작용에 필요한 기반기술에 대해서 연구한다.

1. 3 차원 상호작용의 기본

가상물체를 다루고 가상환경을 통해 운항하기 위해서 오늘날 사용되고 있는 상호작용 기법은 특정한 작업에 맞게 구현된 일반적이지 못한 구현이다. 또한 대부분의 이러한 방법들은 특별한 장치를 요구하므로 다양한 환경에서 일반적으로 사용될 수 없다.

현재의 상호작용 기법을 살펴보면 다음과 같다.

- 물체 선택을 위한 광선 교차 선택(Ray Intersection Picking)

- 화면 정렬 이동(Screen Aligned Translation), 가상 구면체 회전(Virtual Sphere Rotation)등과 같은 가상물체의 직접 다룸(Direct Manipulation)

- 인지를 돕기 위한 Textures, Stereo Displays, 그림자(Shadows) 그리고 다른 Depth Cue 의 사용

- Pan, Zoom 그리고 카메라의 회전

- 사용자의 입력을 특정 자유도(Degree Of Freedom)에 제한하기 위한 물체 손잡이나 격자와 같은 3 차원 위젯

위의 상호작용의 예들은 몇가지 문제점들을 가지고 있다. 가상환경에 많은 물체들이 있다면 가상물체들이 서로 가리어질 수 있으므로 광선교차에 의한 물체의 선택은 문제가 있게 된다. 그리고 실제 세계에서의 2차원적인 손의 움직임을 가상환경에서의 가상물체의 움직임에 사상시키는 것이 어렵기 때문에 2차원 입력장치(Mouse, Tablet, Trackball, etc)를 사용해서 물체를 직접적으로 다루는 상호작용 기법을 구현하기는 어렵다. 따라서 마우스를 이용한 3차원 상호작용에서는 일반적으로 사용자의 손에서 마우스를 통해서, 그리고 3차원 위젯을 통해서, 마지막으로 물체에 사상되는 일련의 간접적 방법이 사용되고 있다.

가상환경에서 운항을 위한 기법도 시점을 통제하기가 어렵기 때문에 상당한 문제점을 드러낸다. 육표(Landmark)를 가상세계에 첨가함으로써 쉽게 길을 잃는 것을 막아 줄 수 있다. 그러나 사용자가 자신의 현재 위치와 가야할 방향을 알고 있다고 하더라도 여전히 운항을 통제할 수 있는 방법이 필요하다. 3차원 사용자 인터페이스에 있어서 은유(Metaphore)의 사용은 사용자가 실제 세계에서의 행동을 가상세계에서의 행동으로 바꾸기 위해서 중요하다. 여러 공간적, 물리적 은유가 사용자에게 운

영할 수 있는 공간의 느낌을 주기 위해서 제시되어져 왔다. 이러한 은유의 종류를 나열하면 다음과 같다.

- 보기 은유(Viewing Metaphore) : 투시투상(Perspective Projection), 평행투상(Orthographic Projection), 랜즈, 윈도우등이 있다.
- 활동 은유(Action Metaphore) : 가리키기(Pointing), 잡기(Grabbing), 날기(Flying), 공간이동하기(Teleportation)등이 있다.

특정한 은유가 사용자에게 효과적으로 해석되기 위해서는 사용자의 실제 세계의 지식을 가상세계에 그대로 적용할 수 있어야 한다.

본 연구에서는 3차원 위젯을 사용한 효과적인 3차원 상호작용에 대해서 개발했고 이러한 방법은 실시간 압연공정의 시뮬레이션의 가사화를 위해서 뿐만 아니라 일반적인 3차원 상호작용에서도 사용될 수 있다.

2. 인간의 가상환경 인지

인간이 가상환경을 인지하고 3차원 상호작용을 수행하는 방법과 많은 자유도를 제공하는 장치들의 문제점을 살펴본다.

가. 운항

가상세계에서 운항을 위한 인터페이스를 만들기 위해서는 실제 세계에서 인간이 운항하는 방법을 먼저 이해해야 된다. 운항은 실제 또는 가상공간을 통한 여행의 계획과 실행으로써 여행되어지는 공간의 내적, 외적 표기를 참조하면서 수행되어진다. 이러한 운항은 위치나 방향을 결정하기 위해서 위치(Position), 속도(Velocity), 가속도(Acceleration)의 세가지 운동학적 단위에 근거를 두고 있다.

□ 위치 근거 운항(Position-based Navigation) : 고정된 육표로부터 나오는 신호의 형태인 위치, 방향을 나타내는 외부 신호에 의존한다.

□ 속도 근거 운항(Velocity-based Navigation) : 여행의 속도와 방향을 나타내는 외부 신호에 의존한다. 이러한 신호를 적분함으로써 선형, 회전 이동량을 구할 수 있다.

□ 가속도 근거 운항(Acceleration-based Navigation) : 초기 위치에 대한 선형, 회전 이동량을 구하기 위해서 여행의 선형, 회전 가속도를 두 번 적분한다.

효과적인 운항을 위해서는 이러한 것들 중에서 위치와 속도가 일반적으로 더 중요시되고 있으며 육표나 수평선, 그림자 같은 것들도 사용자의 위치와 방향에 대한 정보를 제공한다.

나. 3 차원 상호작용

인간은 원래부터 모든 활동을 수행하는데 있어서 활동적이고 의도적이기 때문에 인간이 수행하는 모든 상호작용에 있어서 인지-행위의 피드백 루프가 있다. 현재 인간과 컴퓨터의 상호작용은 거의 감각을 제공해주는 장치들을 포함하고 있지 않기 때문에 주로 사용자의 행위의 시각적, 청각적 피드백만 제공하고 있다.

다. 많은 자유도를 제공하는 장치

시점의 설정이나 물체를 특정위치에 놓는 것과 같은 많은 자유도를 요구하는 일을 위해서 많은 자유도를 제공해 줄 수 있는 장치들을 제공해 주면 좋다. 입력장치와 그 장치의 자유도, 그리고 수행할 수 있는 일들을 표로 나타내면 다음과 같다.

Device	DOF	Task
Slider/dial	1	volume control

Mouse	2	picking, drawing, 2D location
6D Mouse, Head tracker	6	docking, view control
Glove, Face	16이상	hand/face animation
Body suit	100이상	whole body animation

표 1-1 장치, 자유도, 수행하는 일과의 관계

위의 표에서 열거된 장치들과 일의 관계는 명확하다. 그러나 동일한 수의 자유도를 가지지 않는 일을 위해 장치를 사용할 때 문제가 발생한다. 일이 요구하는 자유도가 입력장치의 자유도보다 더 많을 경우에는 사용자 인터페이스는 복합적인 상호작용의 복잡한 간접적인 형태를 취해야 한다. 역으로, 수행하고자하는 일의 자유도가 입력장치의 자유도보더 적을 경우에는 입력장치의 자유도를 수행하고자하는 일의 자유도에 맞추어야 한다. 입력장치의 자유도가 수행하고자하는 일의 자유도에 제한 되었다고 할지라도 사용자의 입력장치를 여분의 자유도 방향으로 움직일 수 있으므로 여전히 문제가 된다. 이러한 문제의 해로써 물리적인 보조장치를 사용하기도 한다.

3.3 차원 상호작용에 있어서의 해결해야 할 문제점

현재 3차원 인터페이스분야에서는 새로운 3차원 상호작용을 만들기 위한 통합된 Toolkit의 필요성을 느끼고 있으며 해결해야 될 문제점으로는 다음과 같은 것이 있다.

- 상호작용 기본단위의 표준화
- LCD, Haptic, 소리 등의 나타내는 기술의 발전
- 입력장치 기술의 발전(음성 및 제스쳐 인식, 시선 추적등)

- 네트웨킹, 여러형식의 입출력의 동시 지원
- 입출력 장치의 장기간의 사용에 의한 증상을 완화
- 확장성과 LOD(Level Of Detail)

4.3 차원 상호작용을 위한 기반기술

사용자가 보고자 하는 정보는 3차원인 가상환경에 존재한다. 그러나 일반적으로 널리 사용되는 컴퓨터 스크린이나 마우스는 2차원적인 성질을 가지고 있다. 이러한 경우에 가상환경에 존재하는 3차원 정보와 사용자와의 상호작용을 위해서 사용되는 장치는 2차원 성질을 가진 장치이므로 사용자는 가상공간과 상호작용을 하기 위해서 3차원 메타포(Metaphore)를 이용한 3차원 위젯(Widget)을 이용하거나, 3차원 가상 메뉴를 이용할 수 있다. 이중에서도 특히 3차원 위젯은 사용자와 상호작용을 하는데 편리하고 효과적이다.

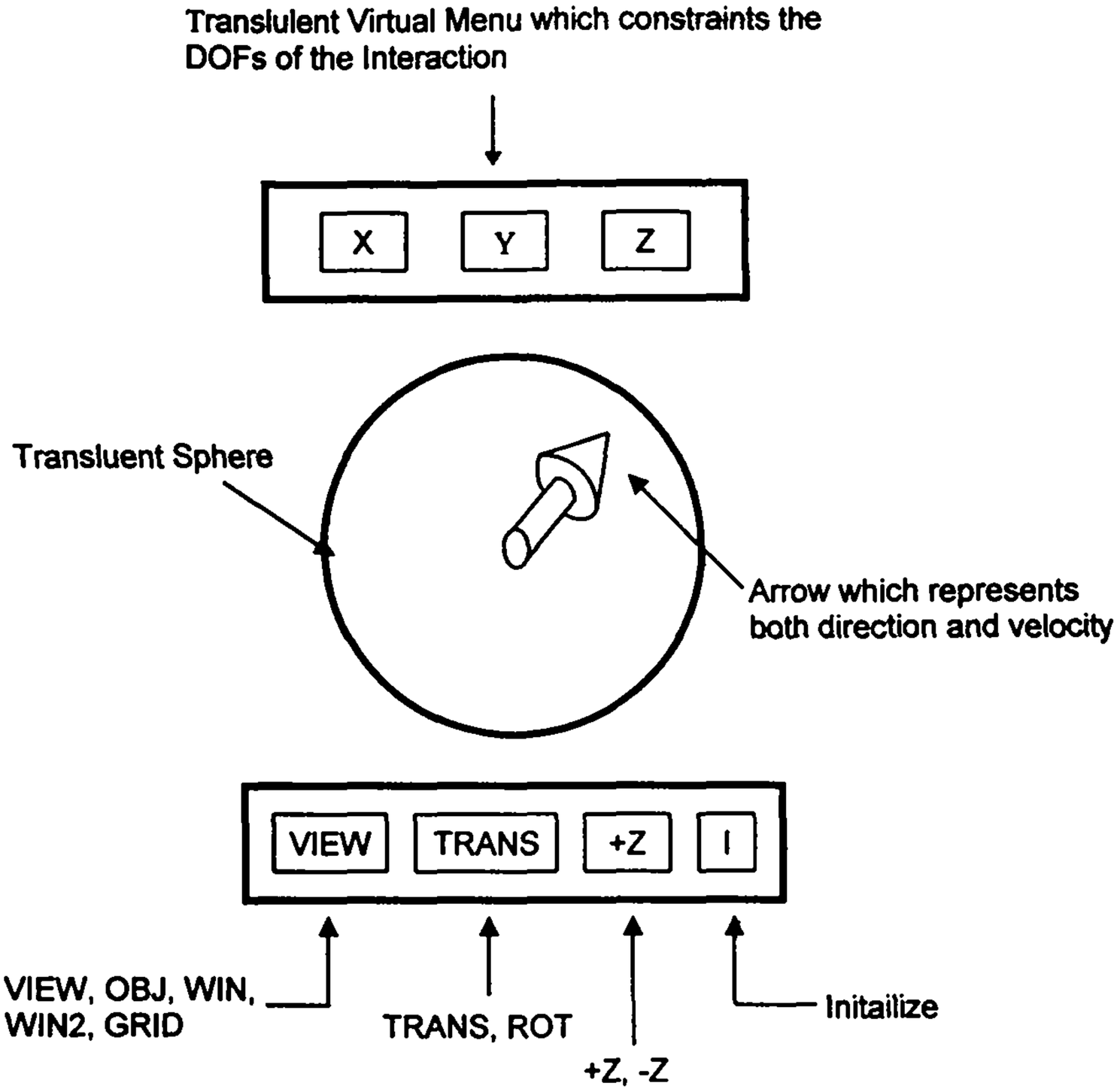


그림 2 3 상호작용을 위한 3 차원 위젯

따라서, 지금부터 가상공간상에서 물체를 다루고 시점을 바꾸는등의 3차원 상호작용을 하는데 편리하고 효과적인 3차원 위젯의 구현과 이를 응용한 상호작용에 대해서 기술한다.

가. 3 차원 위젯의 구현

가상환경에 존재하는 정보를 사용자가 원하는 위치와 방향에서 보기 위해서는 가상환경에 나타나 있는 가상물체나 사용자의 시점(Viewpoint)을 사용자가 원하는

위치로 이동할 수 있어야 한다. 물체의 위치와 방향이나 사용자의 시점을 조정하기 위해서 가장 일반적으로 사용되는 입력장치는 마우스이다. 그러나, 마우스는 2차원적인 정보만을 나타내며, 사용자는 3차원 공간인 가상공간에서 작업을 하여야 하므로 문제가 발생한다. 이러한 문제점은 컴퓨터와 사용자와의 상호작용이 요구되는 거의 모든 응용 분야에서 나타난다.

본 연구에서는 이러한 문제점을 해결하기 위해서 3차원 상호작용에 가장 편리하고 효율적인 3차원 위젯을 디자인했다. 상호작용을 위한 3차원 위젯의 부분별 기능은 그림 2 3에 나타나 있다. 본 연구에서 고안한 3차원 위젯의 특징을 기술하면 다음과 같다.

(1) 3 차원 위젯의 기하(Geometry)

본 연구에서는 상호작용을 위한 3차원 위젯을 구면체(Sphere)로 모델링했다. 구면체는 다른 형태의 물체와는 달리 3차원의 모든 방향을 가장 자연스럽게 나타낼 수 있다. 구면체가 컴퓨터 화면(Screen)에 투사되었을 경우 사용자는 구면체의 반쪽만을 볼 수 있기 때문에 사용자가 볼 수 없는 다른 하나의 반구(Hemisphere)면체를 선택하기 위해서 두 반구면체를 서로 토글(Toggle)시켜줄 수 있는 방법이 필요하다.

(2) 반투명체(Transluent Object)

3차원 위젯을 반투명체로 구현함으로써 얻을 수 있는 잇점은 다음과 같다.

- 사용자는 3 차원 위젯과 상호작용동안에 반투명한 3 차원 위젯뒤에 있는 가상환경을 그대로 볼 수 있다. 사용자는 3 차원 위젯과 가상환경을 동시에 볼 수 있기때문에 더욱 효과적으로 상호작용을 할 수 있다.

- 사용자는 두 개의 반구면체를 동시에 볼 수 있다. 두 개의 반구면체를 동시

에 봄으로 해서 사용자는 두 개의 반구간에 물리적인 전환을 할 필요가 없으며 더욱 더 자연스러운 상호작용을 할 수 있다.

□ 사용자는 방향과 속도를 나타내는 화살표를 볼 수 있다. 3차원 위젯을 반투명체로 함으로써 사용자는 방향과 속도를 바꾸었을 경우 이에 대한 피드백(Feedback)을 동시에 얻을 수 있다. 이러한 피드백을 이용함으로써 사용자는 원하는 방향과 속도로 쉽게 갈 수 있다.

(3) 방향벡터와 속도를 나타내는 화살표

3차원 위젯의 중심에 현재의 방향벡터와 속도를 나타내는 화살표 막대를 가지고 있다. 이 화살표의 한쪽 끝은 구면체의 중심에 고정되어 있으며, 반대쪽 끝은 사용자가 가리키는 위치에 따라 움직이며, 화살표의 길이는 구면체의 반지름과 항상 일치한다. 이 화살표가 가리키는 위치를 계산해서 움직이고자 하는 물체의 방향을 정한다. 즉, 사용자가 마우스를 이용하여 움직이고자 하는 위치를 찍으면 구면체 표면상의 위치 (x, y, z)를 계산한다. x와 y 좌표는 마우스의 화면상에서의 위치를 사용하고 z좌표는 반구의 반지름 r과 x, y좌표를 이용해서 구한다. 즉, 원하는 z는

$$z = \pm\sqrt{r^2 - x^2 - y^2}$$

z값이 양수, 음수 두 가지가 나오게 되고 이 값은 각각 사용자의 시점을 보고 있는 반구면체와 반대쪽을 보고 있는 반구면체의 값이 된다. 그러므로 사용자는 이 두 가지 값 중에서 하나를 선택할 수 있어야 한다. 이러한 선택은 후술한 가상메뉴를 통해서 구현하였다.

이렇게 계산된 (x, y, z)값은 사용자가 움직이고자 하는 방향벡터가 된다. 이러한 방법을 사용하면 2차원 정보만을 나타내는 마우스를 사용하여 3차원인 가상공간의 모든 방향을 나타낼 수 있고 사용자는 원하는 방향으로 움직일 수 있다.

이 3차원 위젯의 단점은 Z축 방향을 한번에 모두 연속적으로 나타낼 수 없다. 사용자가 앞으로 가다가 뒤로 올 경우에 반드시 가상메뉴와의 상호작용이 필요하게 된다. 그러나 인간은 앞으로 가는 행위와 뒤로 가는 행위를 연속적으로 수행하는 것이 아니라 일반적으로 따로 수행하므로 큰 단점이 될 수는 없다.

(4) 3 차원 위젯에 부착된 반투명한 가상메뉴

반투명한 구면체의 위, 아래에는 반투명한 가상메뉴가 부착되어 있어 쉽게 필요한 명령을 가상환경에 줄 수가 있다. 메뉴의 항목을 나타내면 다음과 같다.

□ 3 차원 위젯의 적용대상을 결정하는 메뉴버튼 : 이 메뉴버튼을 선택하면 사용자의 시점을 바꾸기 위한 항목(VIEW), 가상물체를 다루기 위한 항목(OBJ), 현재 보고 있는 위치에서 보이지 않는 가상 물체를 선택하기 위한 사용자의 또 다른 시점을 나타내어 주는 항목(WIN1), 선택된 가상물체 주위를 살펴보기 위한 항목(WIN2), 화면에 격자(Grid)와 관계되는 상호작용을 선택하기 위한 항목(GRID)으로 차례대로 바뀐다.

□ 이동과 회전을 선택하기 위한 메뉴버튼 : 이 메뉴버튼을 선택하면 이동을 위한 항목(TRANS)과 회전을 위한 항목(ROT)이 차례대로 바뀐다. 3 차원 위젯은 이 메뉴버튼에 따라 적절한 작용을 하게 된다.

□ z 방향을 선택하기 위한 메뉴버튼 : 이 메뉴버튼을 선택하면 마우스로 선택된 구면체상의 점이 사용자의 시점을 보고 있는 반구면체에 사상될 것인지 다른 반구면체에 사상될 것인지를 결정할 수 있다.

□ 초기화를 위한 메뉴버튼 : 이 메뉴버튼을 선택하면 현재 선택된 윈도우나 물체가 초기의 위치로 가게 된다. 즉, VIEW 가 선택되어 있다면 사용자의 시점은 최초의 위치로 초기화된다.

□ 자유도(DOF)를 제한하기 위한 메뉴버튼 : 이 메뉴버튼은 가상 구면체 상단에 존재하는 것으로 현재의 자유도를 제한하기 위해서 사용된다. 즉 X를 선택한 후 TRAN인 경우에는 X방향으로만 이동하고 ROT인 경우에는 X축에 대해서만 회전하게 된다. 이 버튼을 이용함으로써 사용자는 여분의 자유도를 제한할 수 있다.

지금까지 설명한 반구 모양의 3차원 위젯을 이용하면 2차원 정보만을 나타내는 마우스를 이용하여 3차원 공간에서 물체의 위치와 방향, 사용자의 시점을 원하는 위치로 움직일 수 있다. 이 3차원 위젯의 구현된 형태가 그림 24에 나타나 있다.

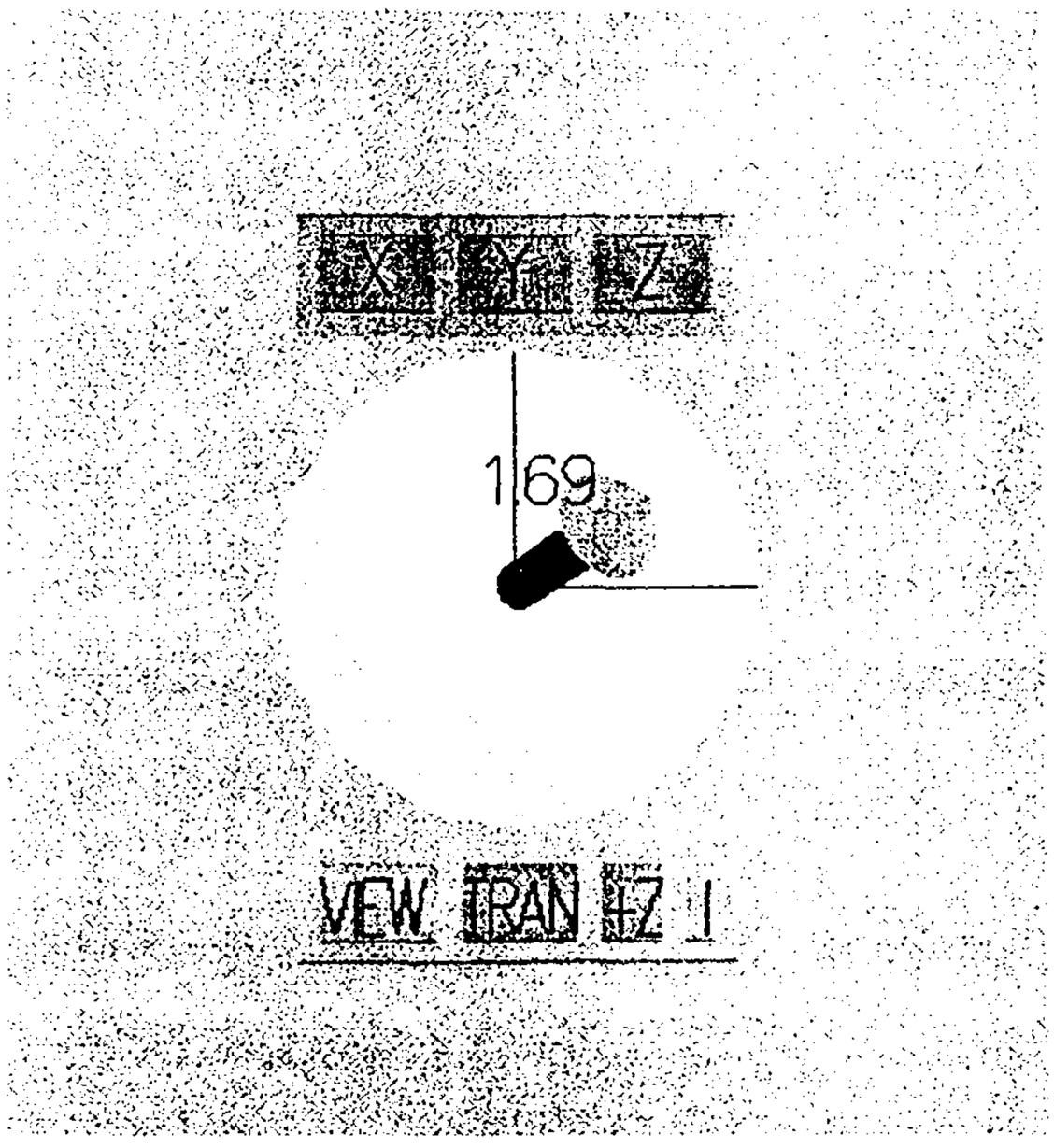


그림 24 상호작용을 위한 3차원 위젯의 구현

나. 이동 속도 조절

이 단락에서는 사용자가 이동하고자 하는 방향을 위에서 설명한 방법과 같이 정

한 다음, 이동하는 속도를 원하는 크기로 정하기 위해서 사용되는 상호작용 방법과 이의 구현에 대해서 설명한다.

대부분 사용자는 물체를 움직일때, 물체가 목적지에서 멀리 떨어져 있을 경우에는 물체는 빠른 속도로 움직이게 하고 물체가 목적지에 가까워지면 정확히 조정하기 위해서 보다 느린 속도로 물체를 움직이게 한다. 또한, 사용자의 시점을 움직일 때도 이와 마찬가지로 먼 거리를 이동할때는 빠른 속도로, 짧은 거리를 이동할때는 느린 속도로 이동한다.

이와 같은 이유 때문에 물체나 사용자의 시점을 이동할때 사용자가 원하는 속도로 이동할 수 있도록 이동 속도를 조절할 수 있어야 한다. 일반적으로 가장 쉬운 방법은 슬라이더 바(slider bar)를 이용해서 원하는 속도를 정한 다음 다시 물체를 이동하거나, 명령어 모드에서 원하는 속도를 직접 입력하여 이동 속도를 정할 수 있다. 그러나, 이러한 방법은 사용자가 물체나 사용자의 시점을 움직이고 있는 도중이라면, 현재 수행중이던 이동 동작을 멈추고 속도를 조절한 후 수행 중이던 이동을 다시 해야 하는 불편함이 있다. 특히, 명령어 모드를 사용할 경우에는 사용중이던 마우스를 계속 사용하지 않고 키보드를 사용해야 하므로 사용자에게 불편함을 따르게 한다.

이와 같은 문제점을 해결하기 위해서 여기서는 위에서 설명한 구면체 모양의 3 차원 위젯을 사용하면서 동시에 속도도 조절할 수 있는 방법을 기술한다. 원하는 방향을 정한 후, 하나의 마우스 버튼(왼쪽 마우스 버튼)을 계속 누르고 있으면 속도가 계속 증가하고, 다른 마우스 버튼(가운데 마우스 버튼)을 계속 누르고 있으면 속도가 계속 감소하게 된다. 그리고, 누르고 있던 마우스 버튼을 놓으면 속도의 변화가 없이 현재 이동되고 있는 속도와 방향으로 계속 이동되며, 지금까지 사용되지 않은 나머지 마우스 버튼(오른쪽 버튼)을 누르면 현재 수행되고 있는 이동을 멈추

게 한다.

이때 사용자가 현재의 속도를 쉽게 알 수 있도록 하기 위해서 3차원 위젯 가운데 부분에 속도를 숫자로 나타내고, 속도가 증가함에 따라 화살표 막대의 색을 변하한다. 또는 화살표 막대의 굵기를 변하게 해서 속도의 크기를 쉽게 알 수 있도록 할 수도 있다. 이렇게 함으로써 사용자는 현재 이동 중인 물체를 멈추지 않고 이동 방향을 변경하면서 자유롭게 속도 조절을 할 수 있으며 3차원 위젯위에 나타난 숫자 정보와 화살표 막대의 색이나 굵기등을 봄으로써 속도의 변화를 쉽게 알 수 있다.

다. 물체의 이동

위의 3차원 위젯을 이용하여 물체를 이동시키기 위해서 사용자와의 상호작용이 어떻게 이루어지는지 알아본다.

물체를 이동하기 위해서는 우선 이동하고자 하는 물체를 오른쪽 마우스 버튼을 이용하여 선택하고, 위에서 설명한 대로 구면체 모양의 3차원 위젯을 이용해서 이동시키고자 하는 방향 벡터를 정하고 마우스 버튼을 이용해서 알맞은 속도로 원하는 위치로 이동시킨다.

사용자는 많은 경우에 물체를 X축만을 따라서 이동한다든지, XY 평면에서만 이동하는 것과 같이 물체의 이동 방향에 제한을 두기를 원한다. 즉 물체의 자유도를 제한할 수 있어야 한다. 이와 같이 물체의 자유도를 제한할 때에는 어느 축에 제한을 둘 것인지를 결정해야 되는데 이것은 간단한 토글 버튼을 사용하거나, 3차원 위젯에 부착된 가상메뉴를 사용할 수 있다. 예를 들어, 각 축에 대한 토글 버튼(ToggleButton)을 두어서 그 축에 제한을 둘 것인지를 결정한다. 축에 대해 제한하는 방법을 기술하면 다음과 같다.

□ 한 축에만 제한하는 경우 : 한 축에만 제한을 두면, 그 축으로만 물체를 이동하고자 하는 것이므로 마우스가 어디를 가리키든지간에 화살표 막대를 선택된 축만을 가리키게 하고 그 축을 향해 현재 지정된 속도로 물체를 이동시킨다. 화살표 방향이 선택된 축만을 가리키게 하기 위해서는 3 차원 위젯상에서 화살표의 좌표를 예전과 달리 구해야 한다. 제한된 축의 좌표는 3 차원 위젯인 구면체의 반지름으로 하고, 나머지 축의 좌표는 0 로 하면 된다. 즉, X 축을 제한했을 경우에는 화살표는 (r, 0, 0)방향을 가리키게 된다.

□ 두 개의 축에 제한하는 경우 : 두 개의 축에 제한을 두면, 평면상에서만 물체를 이동하고자 하는 것이므로 3 차원 위젯의 화살표 막대를 제한된 평면상에서만 움직이도록 한다. 이렇게 하기 위해서는 마우스가 가리키는 방향에 따라 3 차원 위젯상의 화살표 좌표도 다르게 계산된다. 즉, 정해진 3 차원 위젯상의 좌표를 제한을 두지 않은 경우처럼 구한 다음, 제한된 축의 좌표 두 개와 나머지 축의 좌표를 0 로 해서 화살표 막대의 길이를 반구의 반지름으로 확장해서 구한다. 이런 방법으로 구해진 화살표의 좌표를 이용해서 물체를 이동시키면 제한된 평면에서만 물체가 이동된다. 세개의 축에 모두 제한을 두는 경우와 어떠한 축에도 제한을 두지 않는 경우에는 아무 제한도 없는 것으로 하고, 반구 모양의 3 차원 위젯상의 좌표를 그대로 이용해서 물체를 이동시킨다.

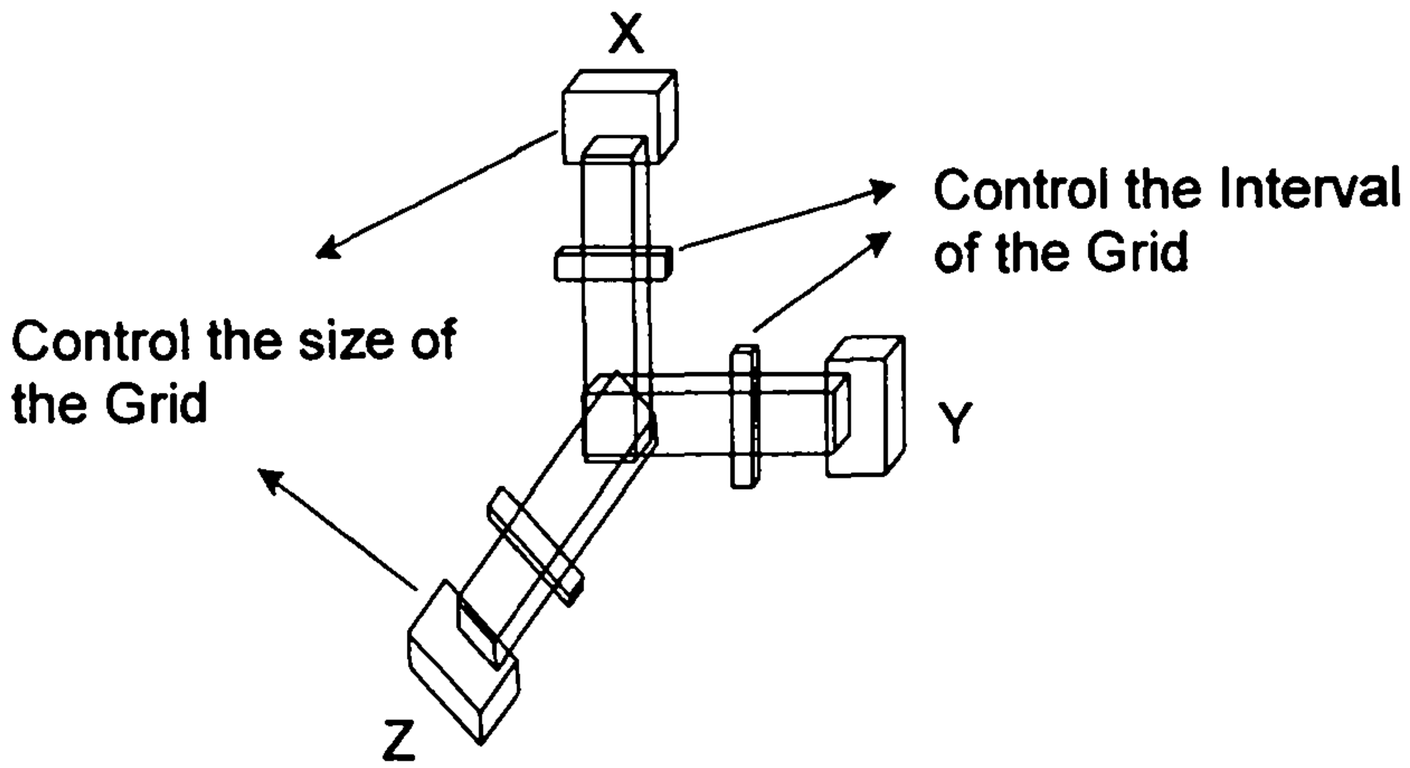


그림 25 격자를 위한 3차원 위젯

라. 물체 이동을 위한 격자(Grid)

사용자가 물체를 이동시킬때 물체를 선택해서 임의의 양만큼 이동시킬 수도 있지만, 정확한 양의 이동을 위해서 사용자에게 격자를 제공할 수 있다. 이 방법은 한 축이나 한 평면에 제한을 두고 이동할 때도 가능하며, 임의의 방향으로 이동할 때도 가능하다.

이와 같은 물체 이동을 위한 격자는 물체를 선택하고 격자를 나타내고자 하면 그 물체 중심을 중심으로 한 격자가 나타난다. 이때 격자의 크기와 간격을 조절하기 위해서 메뉴나 명령어 모드, 슬라이더 바등을 사용할 수 있지만, 사용자와의 효과적인 상호작용을 위해서 또 다른 3차원 위젯으로 구현한다. 이 격자 조절 위젯은 3개의 축으로 이루어져 있으며 각 축 끝에는 전체 격자의 크기를 조절하는 스케일(Scale) 위젯이 있으며, 각각의 축에는 격자의 간격을 조절하기 위해서 각 축을 따라 자유롭게 움직이는 스케일 위젯이 있다. 사용자는 선택된 물체를 정확히 이동시키기 위해서 격자를 선택된 물체에 나타나게 하고, 격자 조절 위젯을 사용해서

원하는 격자의 크기와 간격을 정하면 물체는 이 격자점만을 따라서 이동하게 된다. 이러한 격자의 크기와 간격을 조절하기 위한 3차원 위젯의 기능과 구현된 형태는 그림 25, 그림 26에 나타나 있다.

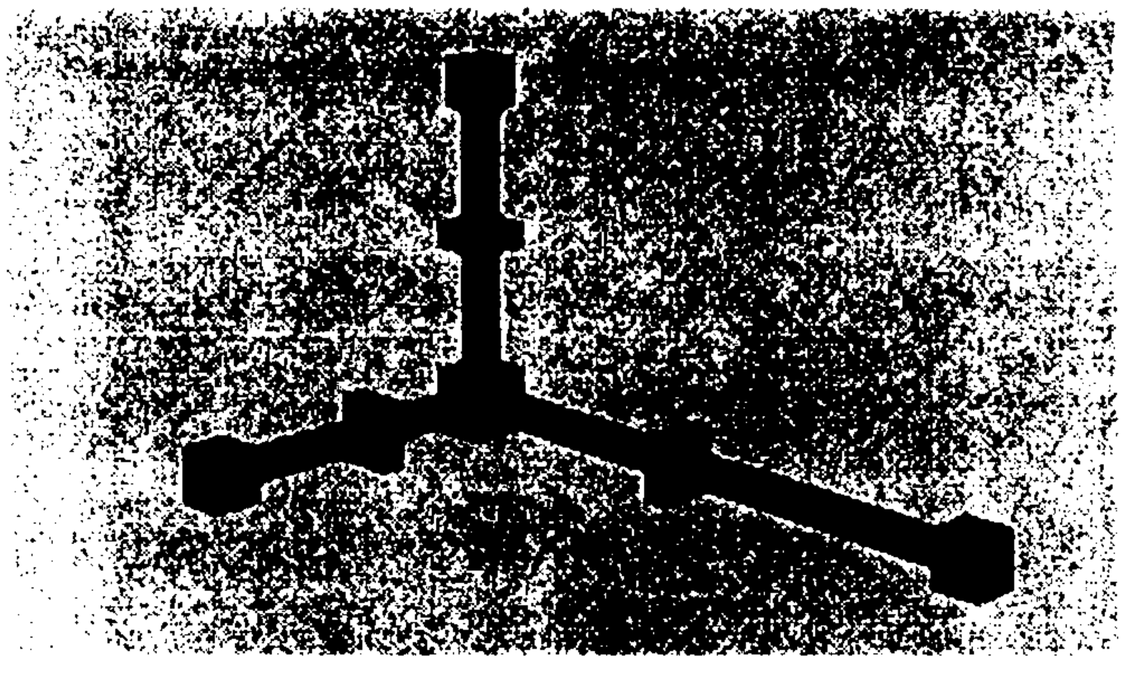


그림 26 격자를 위한 3차원 위젯의 구현

마. 물체의 회전

2차원 마우스를 통해 3차원 회전에 대한 인터페이스를 구현하는데 트랙볼의 개념이 도입된다. 이 단락에서는 3차원 상의 좌표에 대한 회전 계산과 구현 방법을 기술한다.

한 윈도상에 임의의 구를 가정하고, 마우스의 입력 좌표는 구상의 한 점으로 해석된다. 그림 27은 윈도우상에 주어진 구를 보여주며, 두 점 (x1, y1, z1)는 마우스가 클릭된 처음의 점이며, (x2, y2, z2)는 선택된 마지막 점이 된다. 마우스 클릭시 윈도상의 임의의 점 (x1, y1), (x2, y2) 좌표값은 다음과 같이 구상의 z1, z2 좌표값을 구한다.

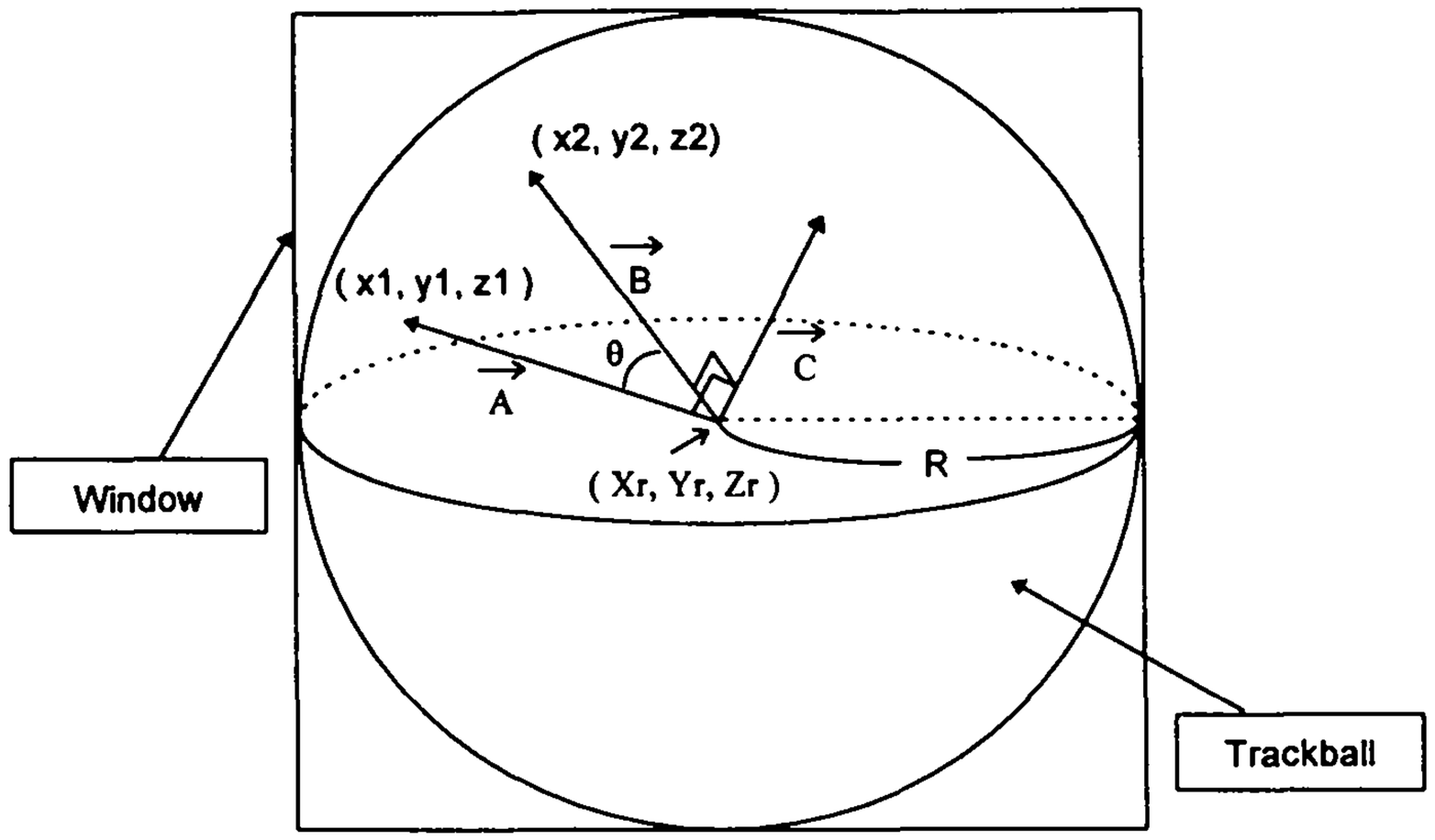


그림 27 트랙볼

구의 방정식은 다음과 같이 주어지며

$$(x-Xr)^2+(y-Yr)^2+(z-Zr)^2=R^2$$

$$z=\pm\sqrt{R^2-(x-Xr)^2+(y-Yr)^2} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots(1)$$

z1, z2 값은 (1)식의 x, y 에 대입하여 계산한다. + 부호는 구의 위쪽 반구의 점을 의미하며 -부호는 구의 아래쪽 반구상의 점으로 해석한다. 구상의 두점을 구상의 좌표에 대한 벡터로 다음과 같이 표기한다.

$$\vec{A}=(A_x,A_y,A_z)=(x1-Xr,y1-Yr,z1-Zr)$$
$$\vec{B}=(B_x,B_y,B_z)=(x2-Xr,y2-Yr,z3-Zr)$$

두 벡터의 내적으로 이루는 각θ를 구하고, 외적을 통해 회전된 각의 회전축을 계산한다.

$$\vec{A}\cdot\vec{B}=|A|\cdot|B|\cdot\cos\theta$$

$$\therefore \theta = \cos^{-1}\left(\frac{\vec{A}\cdot\vec{B}}{|A|\cdot|B|}\right)$$

$$\vec{C} = \vec{A}\times\vec{B}$$

$$\therefore \vec{C} = ((y1\cdot z2 - y2\cdot z1),(x2\cdot z1 - x1\cdot z2),(x1\cdot y2 - x2\cdot y1))$$

위의 트랙볼 인터페이스 통해 계산된 회전축과 각도를 알고 있으면, 3 차원 상의 객체에 대한 회전연산의 결과는 쿼터니언(Quaternion)을 이용하여 얻을 수 있다. 쿼터니언은 다음과 같이 수학적으로 정의된다. 그림 2 8 를 참조한다.

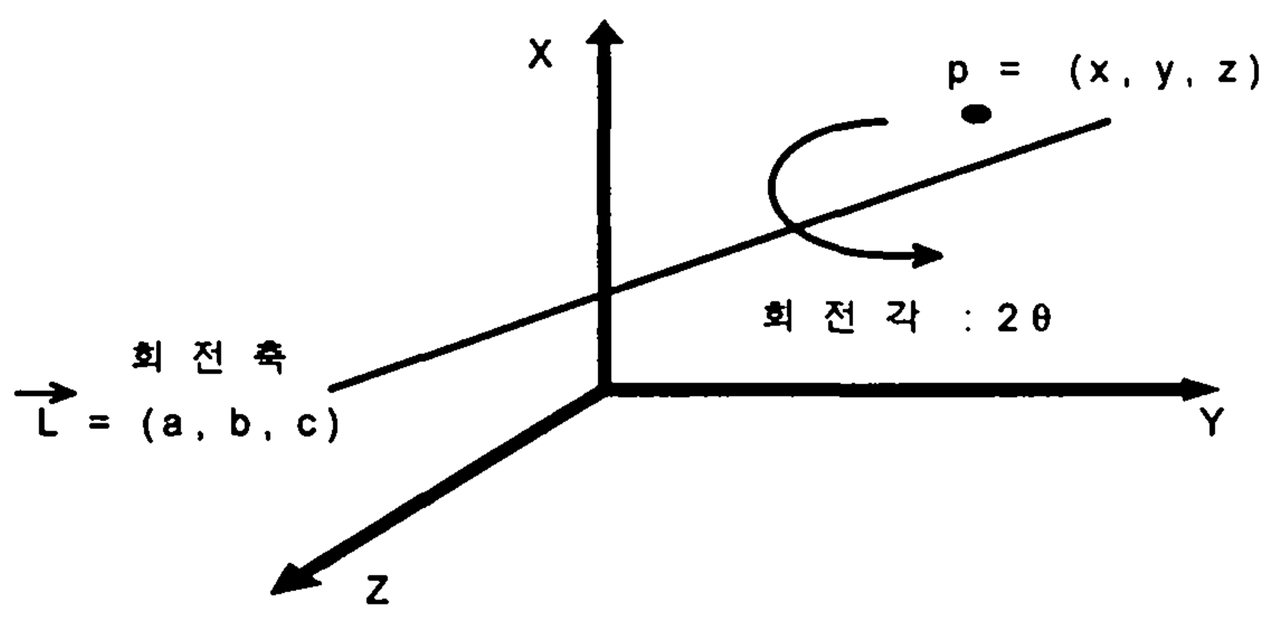


그림 2 8 임의의 축에 대한 회전

$$q = (\cos\theta, \sin\theta\cdot(a,b,c)) \in S^3$$

회전 쿼터니언 q 에 대해 임의의 점 p 를 회전시킨 결과 $R_q(p)$는 다음과 같다.

$$R_q(p) = q\cdot p\cdot \bar{q}$$

(단, $p = (0,(x,y,z))$, $\bar{q} = (\cos\theta, -\sin\theta(a,b,c))$)

결과적으로 나온 좌표는 회전각 2θ에 대한 회전된 좌표이다. 두번의 쿼터니언

곱으로 회전연산을 할 수 있으므로 기존의 행렬연산과 비교해 볼 때, 상호작용적인 시스템에서 이용하면 매우 효율적이다.

바. 사용자 시점(Viewpoint)의 이동과 회전

사용자는 화면에 나타난 물체 자체를 이동시키거나 회전시킬 수도 있지만 물체는 움직이지 않고 사용자가 보고자 하는 물체를 좀더 가까이 보거나 그 물체의 뒷면이나 윗면, 또는 다른 각도에서 보고자 할 수도 있다. 이러한 경우에는 사용자의 시점을 움직이면 된다. 즉, 물체를 가까이 보고나, 멀리 보고 싶을때는 사용자의 시점을 이동(Translate)시키고, 물체를 다른 각도에서 보고 싶을때는 사용자의 시점을 회전(Rotate)시키면 된다.

이때, 사용자의 시점을 이동시키거나 방향을 바꾸는 방법과 구현 방법은 물체를 이동시키거나 물체의 방향을 바꾸는 경우와 같다. 그러나, 이동 방향과 회전 방향은 물체의 경우와 반대로 되어야 한다. 예를 들어 물체를 오른쪽으로 이동시키고 싶으면 물체가 오른쪽으로 이동되도록 물체의 좌표를 모두 이동시키면 되지만, 사용자의 관점을 오른쪽으로 이동시키고 싶으면, 실제로 물체는 왼쪽으로 이동된 것처럼 보이므로 물체를 왼쪽으로 이동시켜 나타나게 해야된다. 사용자의 관점을 회전시키는 경우도 이와 마찬가지로 구현 방법과 사용자와의 상호작용 방법은 같지만 회전 방향만 반대로 해 준다.

사. 사용자의 다른 시점을 가진 창(Window)

물체를 이동할때 한 화면에 보이는 다른 장소로 움직일 수도 있지만 화면에 나타나지 않은 다른 곳으로 물체를 이동할 수도 있다. 이러한 경우에는 물체를 선택해서 원하는 장소 부근으로 이동시킨 후에 사용자의 관점을 다시 원하는 장소로 이동시키는 일을 계속 반복해야 한다. 또한, 물체의 이동이 다 끝난 후에 원래의 위치

에서 다시 일을 해야 하는 경우엔 또 다시 사용자의 관점을 원래의 위치로 이동시켜야 하는 어려움이 있다.

따라서, 이러한 어려움을 해결하기 위해서 현재 위치하고 있는 사용자의 관점과 다른 관점을 가진 창(Window)을 둔다. 이 창에서 사용되는 사용자의 관점도 위에서 설명한 방법과 같이 반구 모양의 3차원 위젯을 같이 사용함으로써 이동할 수 있다.

이 창은 현재 화면에 나타난 부분외에 다른 부분에 대한 정보를 보여줌으로써 사용자가 보다 많은 정보를 이용할 수 있게 해준다. 즉, 물체를 화면에 나타난 부분외에 다른 부분으로 이동시킬 경우 이 창에서 사용되는 관점을 물체를 이동시키고자 하는 위치로 이동시켜 놓고 물체를 이동시킨다. 그러면, 처음에는 물체가 이동하는 모습이 주 화면에 나타나다가 화면밖의 다른 부분, 즉, 창의 관점안에 있는 부분으로 이동되면 그 창에 물체가 이동하는 모습이 보인다. 이렇게 함으로써 물체와 사용자 관점의 이동을 반복해야 하는 번거로움을 없앨 수 있으며, 사용자에게 보다 넓은 부분의 정보를 제공할 수 있다. 또한 이 창에서도 물체를 선택할 수 있게 함으로써 화면에 보이지 않고 다른 부분에 있는 물체도 선택할 수 있게 할 수도 있다. 이와 같은 또 다른 사용자의 관점을 가진 창을 화면의 일정 부분에 나타나게 하기 위해서는 Viewport를 다르게 설정해서 나타나게 하면 된다.

아. 선택된 물체를 가까이 보여주는 창

물체를 선택해서 원하는 위치에 원하는 방향으로 놓고 싶을때 선택된 물체를 좀 더 자세히 보면 더 편리하다. 이런 경우에 사용자의 관점을 이동시킬 수도 있지만, 이와 같은 상황은 거의 항상 일어나는 일이므로 선택된 물체를 가까이 보여주는 창을 따로 두는 것이 더 편리하다. 이와 같이 선택된 물체를 가까이 보여주는 창을 따로 보여줌으로써 사용자가 물체를 다루는데 더욱 편리하게 되고 물체와의 상호작

용이 좀 더 정밀해진다. 이 창은 다른 사용자의 시점을 가진 창과 같이 Viewport 를 다르게 설정해서 나타낼 수 있다.

선택된 물체를 가까이에서 보여주는 창의 이동과 회전은 사용자의 시점의 이동, 회전과 유사하다. 그러나 회전의 경우 사용자의 시점 주위로 회전하는 것이 아니라 물체의 중심점의 주위로 회전한다. 이렇게 함으로써 물체의 관심있는 부분을 쉽게 볼 수 있다.

지금까지 설명한 3차원 위젯과 창을 사용함으로써 사용자는 마우스와 같은 2차원 입력장치를 사용하여 3차원 공간상에서 정확한 상호작용을 할 수 있다. 그림 2 9에 상호작용을 위한 기반기술을 응용한 예에 대해서 나와 있다.

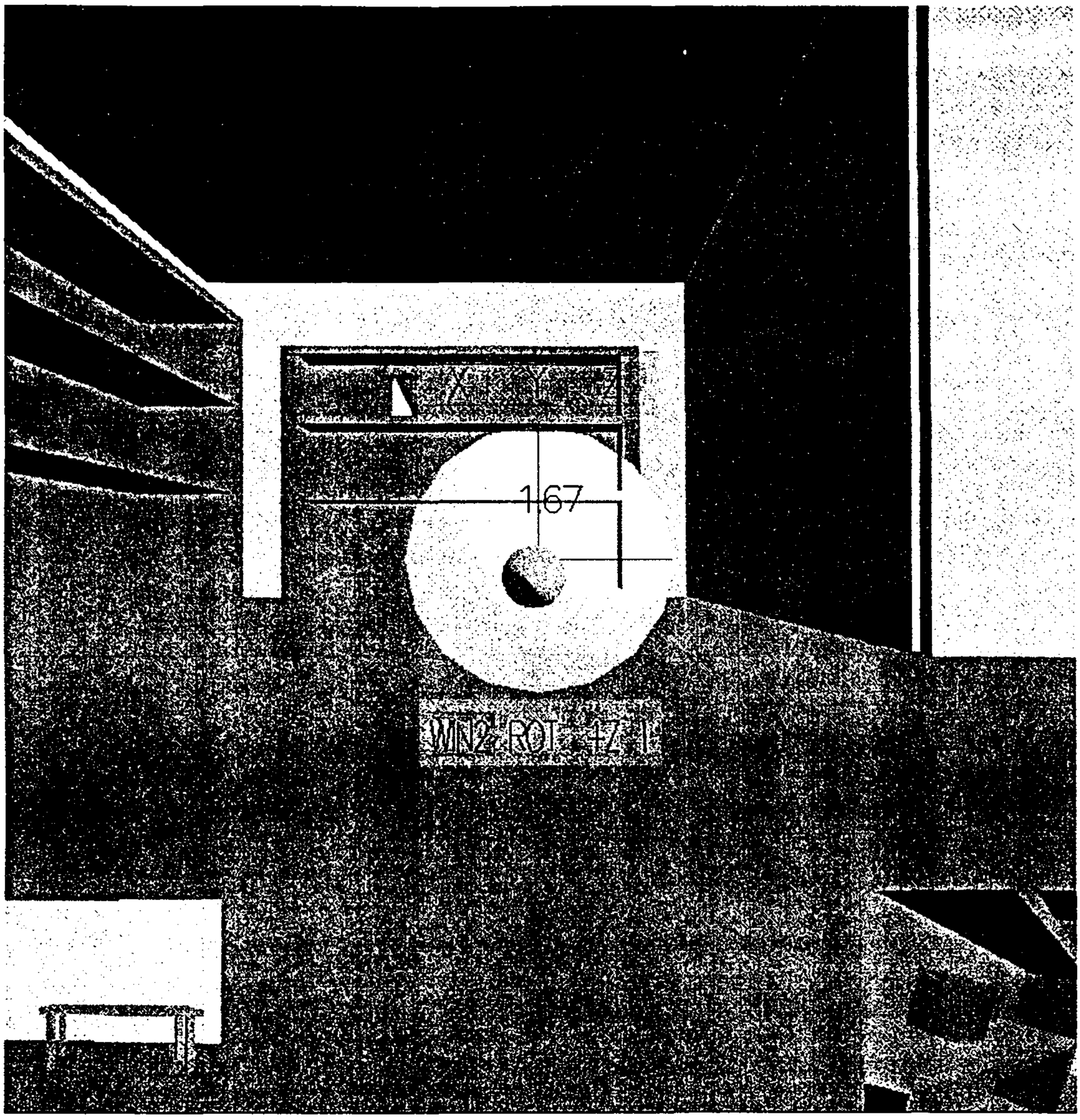


그림 29 상호작용을 위한 기반기술을 응용한 예

5. 실시간 압연공정 시뮬레이터의 가시화를 위한 3 차원 상호작용

실시간 압연공정 시뮬레이터의 가시화를 위해서 앞에서 기술한 마우스를 이용한 3차원 상호작용을 위한 기반기술중에서 일부만을 이용한다. 실시간 압연공정 시뮬레이터의 가시화의 경우에 가상환경이 비교적 간단하기 때문에 전술한 기반 기술중

일부만을 사용함으로써 사용자의 편의성을 오히려 증대시킬 수 있다.

가. 회전

전체의 화면을 가상 구면체상에 사상시켜서 회전시키는 가상 trackball의 원리를 이용한다. 이러한 원리는 전술한 기반기술에서 사용한 원리와 동일하다. 실시간 압연공정 시뮬레이터의 가시화에서는 가상환경이 비교적 단순하고 공간적으로 밀집되어 있으므로 사용자의 시점을 이동시키는 것보다는 전체 가상환경을 회전시켜서 사용자가 살펴보도록 하는 것이 분석하기에 용이하다. 그림 30에 가상환경을 회전시킨 예가 나타나 있다.

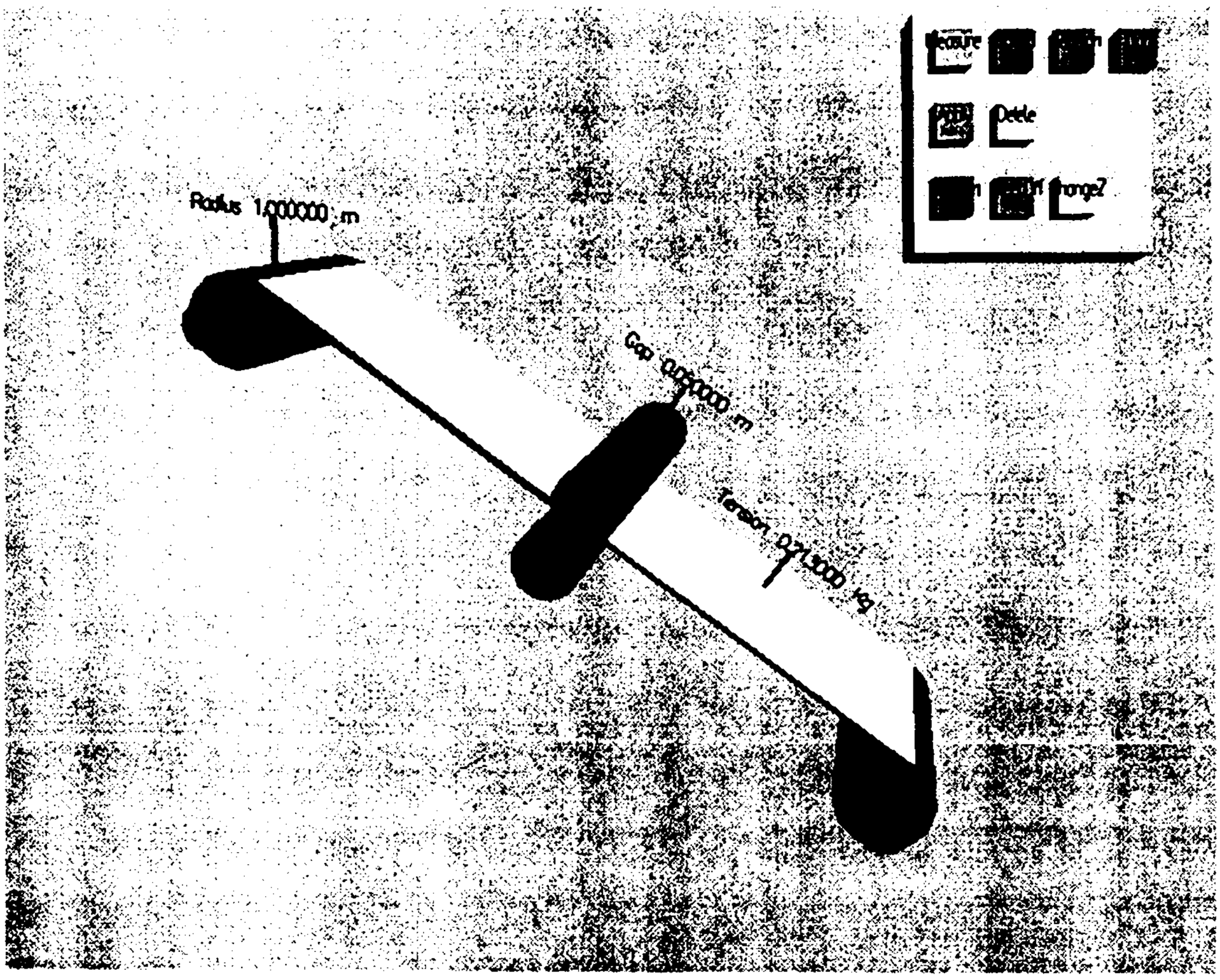


그림 30 가상 환경의 회전

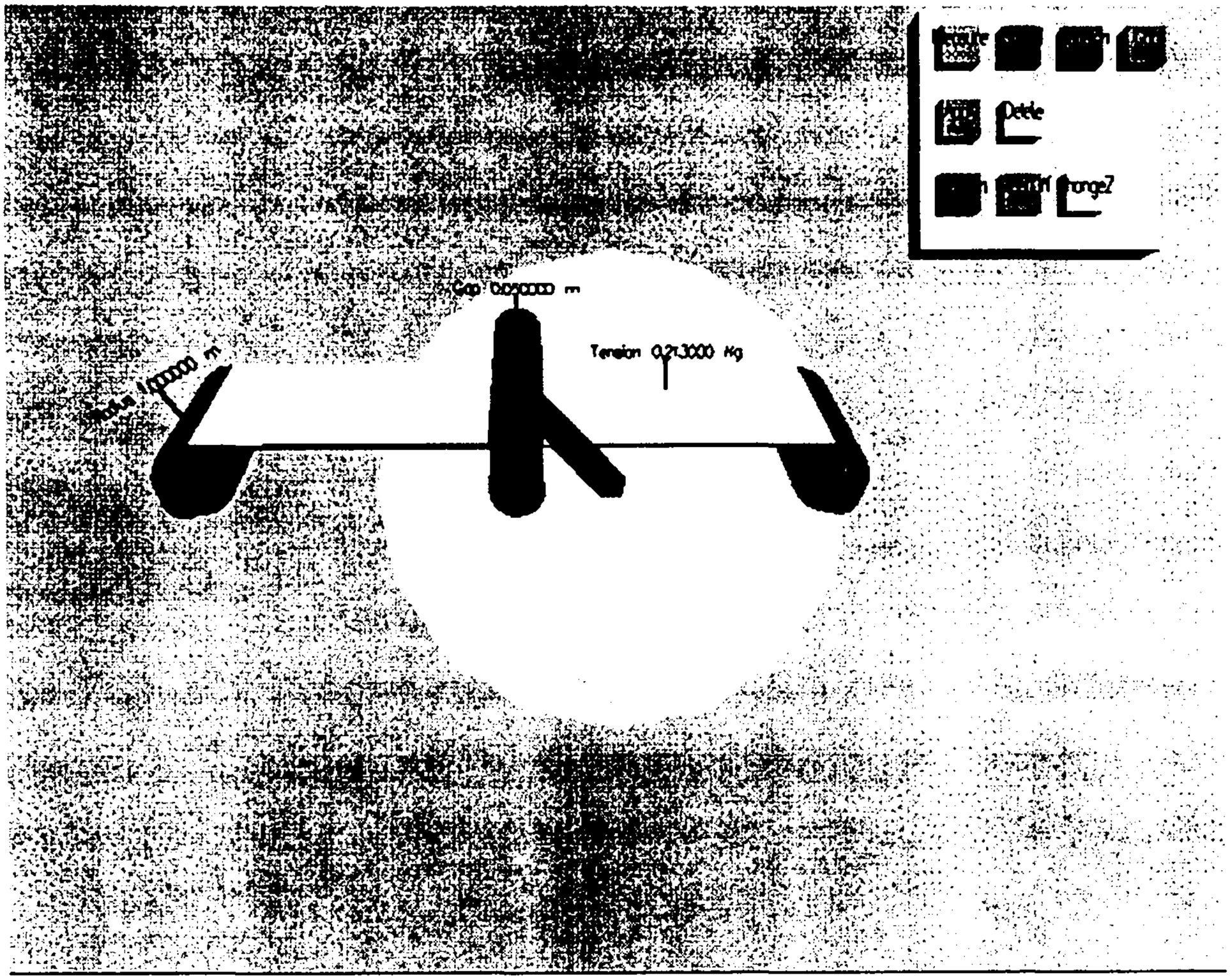


그림 31 가상환경의 이동

나. 이동

가상환경에서 이동하기 위한 방법으로는 전술한 3차원 위젯을 사용한다. 가상공간상에 반투명한 구면체를 사용하며 이동하는 방향을 나타내는 막대기를 구면체상에 나타낸다. 이렇게 함으로써 X, Y, Z방향을 동시에 제어할 수 있고 사용자의 이해도를 증진시킬 수 있다. 즉, 가상 구면체상의 특정위치를 오른쪽 마우스 버튼을 눌러서 선택하므로써 2차원상의 좌표(x, y)가 3차원상의 구면체상의 좌표(x, y, z)를 사상되고 시점이 (x, y, z)방향으로 단위거리만큼 움직인다.

가상환경상에서 이동을 위한 3차원 위젯을 나타내고 없애기 위해서 가상메뉴상의 SteerOn, SteerOff 버튼을 사용한다. 위의 가상구면체는 한 번에 반구밖에 사용할 수 없으므로 앞으로 가다가 뒤로 갈 경우에는 Z방향을 바꾸어 주어야 한다. 이때 가상메뉴상의 ChangeZ 버튼을 사용한다. 그림 3 1에 이동에 대한 예가 나타나 있다.

6 절 결론 및 향후 연구계획

실시간 그래픽 사용자 인터페이스는 전체 시스템의 최상단에 위치하는 것으로써 실시간 압연공정 시뮬레이션을 통해서 얻어진 데이터를 실시간에 3차원으로 사용자와 상호작용을 통해서 가시화한다. 압연공정 시뮬레이션을 통해서 연속적으로 발생하는 데이터를 TCP/IP를 이용해서 받은 후 이 데이터를 일정한 시간 주기로 샘플링하고 샘플링된 데이터를 가상도구를 이용하여 실시간에 가상환경에 가시화하고 에니메이션한다. 데이터를 3차원으로 가시화할 수 있을뿐만 아니라 가상환경에 문자나 숫자와 같은 2차원으로도 가시화할 수 있다. 이 때 사용자는 실시간에 컴퓨터와의 상호작용을 통해서 시점을 바꾼다든지 물체의 크기를 조정한다. 사용자의 상호작용을 편리하게 하기 위해서 가상메뉴나 3차원 위젯을 제공한다. 본 연구에서는 상호작용시에 일반적으로 가장 널리 이용되는 마우스를 이용한 상호작용 방법을 개발했다.

향후 과제에서 사용자가 시뮬레이션의 조건을 가상현실 기술을 이용하여 가상환경속에서 직접 입력하고 입력된 결과를 받아서 시뮬레이터가 시뮬레이션한 후 이를 다시 가상현실 기술을 이용하여 가상환경에서 사용자가 이해, 분석할 수 있는 방법에 대해서 연구하겠다.

여 백

제 3 장 실시간 마이크로 커널

1 절 서론

자동화된 작업 공정에서는 각 작업 모듈이 사물의 형태, 유체의 속도와 같은 외부 상황을 센서를 통해 파악하고 각 상황에 필요한 계산을 수행한 다음 기계 손이나 밸브 등을 조작한다. 일반적으로 이러한 동작들은 주어진 시간 내에 이루어져야 하며 그렇지 않을 경우 작업이 순조롭게 진행되지 못하거나 심한 경우 전체 시스템에 치명적인 문제를 일으킬 수 있다. 따라서 이러한 시스템들은 실시간 시스템으로 구성된다. 작업 모듈들 중에 제품 표면의 흠을 찾아내는 모듈은 많은 연산 시간을 필요로 한다. 하지만 일반적인 프로세서로는 주어진 시간 내에 실행 결과를 얻을 수가 없으므로 벡터 컴퓨터나 병렬 컴퓨터 등 빠른 연산을 할 수 있는 시스템이 요구된다. 벡터 컴퓨터와 같은 시스템은 가격이 상당히 비싸므로 본 연구에서는 보다 적은 가격으로 유사한 성능을 낼 수 있는 병렬 컴퓨터를 실시간 시스템에 사용한다

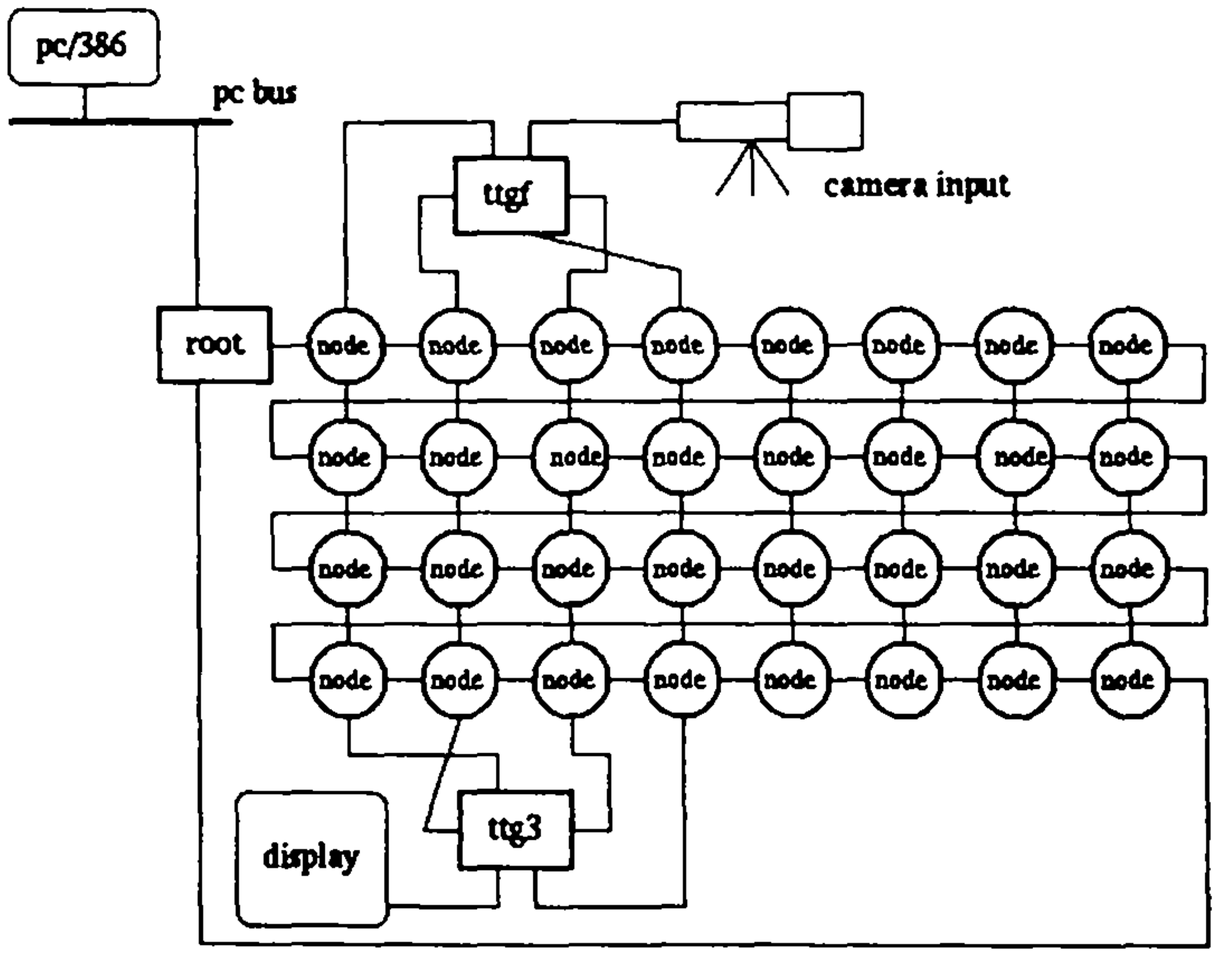


그림 32 TiME 의 구조

병렬 컴퓨터 TiME은 포항 공과 대학교에서 개발한 시스템으로 32개의 작업 노드(프로세서)와 4개의 디스크 노드, 1개의 카메라 입력 노드, 1개의 모니터 출력 노드, 호스트 컴퓨터와의 인터페이스를 위한 루트 노드 1개 등 총 39개의 프로세서로 구성되어 있다 (그림 32). 작업 노드는 기본적으로 4×8 메쉬 (mesh) 구조로 되어 있으며 외부의 인터페이스는 C004라는 스위치를 통해 서로 간의 연결이 가능해 토러스 구조로도 변형이 가능하다. 각 노드는 기본적으로 INMOS사의 병렬 프로세서인 transputer T800이 기본 프로세서로 동작한다. 카메라 노드에는 카메라 제어를 위한 카드가 있고 모니터 노드에는 그래픽 카드가 있다. 또한 각 디스크 노드에는 SCSI 디스크를 제어하는 프로세서가 내장되어 있다. 일반적으로 TiME은 이미지 처리를 위해 사용되며 공장 자동화에 적용할 경우 제품 표면의 흠을 검사하는 모듈에 적당하다. 이 때, 카메라 노드가 제품 표면을 읽고 각 작업 노드가 입력된 이미지를 처리하며 모니터 노드가 그 결과를 화면에 출력하도록 한다. 디스크 노드는 이미지 처리 중에 필요한 정보를 읽어 오거나 결과를 임시 저장하는 데 사용한다. 루트 노드는 호스트 컴퓨터에서 작업을 제어하기 위해 사용한다. 이 때 전체 시스템이 실시간을 지원하기 위해서는 각 노드의 커널이 실시간을 지원해야 한다.

병렬 컴퓨터 TiME의 사용으로 실시간 시스템이 요구하는 빠른 실행 시간의 조건을 만족시킬 수 있더라도 실시간 시스템은 기본적으로 예측 가능성을 만족해야 한다. 즉 새로운 작업이 수행되기 전에 이 작업이 시스템에 추가되더라도 다른 프로세스의 마감 시간이 지켜질 수 있는지 알 수 있어야 한다. 주어진 프로세스들이 마감 시간을 지킬 수 있도록 관리하는 것이 스케줄러인데, 대표적인 스케줄링 알고리즘으로 Rate Monotonic, Earliest Deadline First, Stack-Based Scheduling 등이 있다. 이들 스케줄링 알고리즘들은 다중 우선순위 환경에서 수행되기 때문에 기본적으로 시스템이 다중 우선순위를 지원해야 한다. 하지만 transputer는 2-단계의 우선순위만을 지원하기 때문에 소프트웨어적으로 다중 우선순위를 지원해야 한다.

TiME은 각 노드별로 설치되어 있는 하드웨어가 다르며 따라서 작업의 종류도 다르게 된다. 각 노드가 일반적으로 사용하는 기능을 커널로써 모두 포함시키게 되면 커널의 용량이 매우 커지게 되어 각 노드 당 필요한 메모리의 양이 증가한다. 이러한 문제를 해결하기 위해 본 연구에서는 마이크로 커널을 각 노드의 커널로 한다. 마이크로 커널이 가지는 기능은 프로세스 스케줄링, 프로세스 관리, 메모리 관리, 통신 관리 등이다. 이외에 카메라 노드나 작업 노드에서 필요한 기능은 별도의 라이브러리로써 지원한다.

본 장의 구성은 다음과 같다. 2 절에서는 마이크로 커널의 기본 기능인 메모리 관리, 통신 관리 그리고 프로세스 관리에 대해서 다룬다. 3 절에서는 마이크로 커널의 기능의 하나인 프로세스 스케줄링을 다룬다. 마지막으로 4 절에서는 결론과 추후 연구 방향에 대해서 논한다.

2 절 마이크로 커널의 기능

1. 메모리 관리

동적 메모리를 사용하기 위한 ANSI C 에서 기본적으로 제공하는 malloc(), free()등의 메모리 운영 함수들을 일반적으로 빠른 속도를 제공하지 않는 것으로 알려져 있다. 메시지 전송에 기반을 둔 시스템에서는 메시지 전송이 발생할 때마다 메모리 할당과 복구(free)가 이루어져야 하므로 잦은 malloc(), free()등의 사용은 시스템에 큰 부하를 발생시킨다. 따라서 빠른 메모리 운영이 필요하게 된다. 이를 위해 테이블 기반의 자료 구조를 구성한다. 각 테이블 내의 자료는 같은 크기의 사용 가능 메모리의 위치를 가리키고 각 사용 가능 메모리는 연결 리스트(linked list) 구조를 이룬다. 본 연구에서 설계된 메모리 함수에서는 free()에 의해서 메모리를 되돌리지 않고 메모리 자료 구조의 리스트 내에 저장된다. 새로운 메모리 할당 함수는 우선 이 자료 구조를 살핀 다음 필요한 크기의 메모리가 있거나 더 큰 크기의 메모리가 있으

면 그 메모리를 할당해주고 그렇지 않으면 기존의 malloc()을 호출해서 메모리를 얻는다. 실제로 메모리 할당과 복구가 빈번한 시스템에서는 같은 크기 메모리의 할당과 복구가 자주 일어나므로 이 자료 구조는 메모리 관리 시간을 단축시킨다.

2. 통신 관리

가. 포트 제어

생성된 프로세스 마다 포트(Port)라는 전달된 메시지를 저장하는 자료 구조를 할당함으로써 프로세스간 통신이 가능하게 한다. 하나의 프로세스는 여러 개의 포트를 가질 수 있으며 프로세스가 생성될 때 정해진 크기의 배열로써 포트를 생성한다. 각 배열의 원소는 포트 자료 구조를 가리킨다.

나. 메시지 관리

분산 컴퓨팅 환경에서는 기본적으로 임의의 프로세스 간의 메시지 전달이 제공되어야 한다. 이를 위해서 각 프로세서의 커널마다 하드웨어 링크로부터 메시지를 받고 이 메시지가 자신의 프로세서에서 실행되는 프로세스를 향한 메시지인지 다른 프로세서의 프로세스인지를 확인한다. 그 메시지가 자신의 프로세스를 향한 메시지이면 그 프로세스의 포트에 옮긴다. 그렇지않은 경우에는 라우팅 테이블(routing table)을 사용해서 메시지가 이동해야 할 하드웨어 링크로 메시지를 보낸다. 외부에서 들어오는 메시지는 CommServerCode 와 같은 프로세스가 처리하게 된다. 노드간의 통신은 송신 측에서 헤더 정보를 보낸 후 수신 측에서 헤더 정보를 읽으면 송신 측에서 데이터 메시지를 보낸다.

다. 이벤트 제어

프로세스간의 동기화를 위해서 이벤트라는 자료 구조를 프로세스에 할당한다. 즉, 다른 프로세스들로부터 지정된 개수 만큼의 이벤트가 도착했을 때 프로세스가

작동되도록 할 때 사용된다. 이벤트 구조는 배열로 할당된다.

3. 프로세스 관리

프로세스가 생성될 때 프로세스가 사용할 스택(stack)이나 힙(heap)의 크기를 지정하는 기본적인 기능 외에 통신과 사건 제어를 위한 자료 구조를 구성해야 한다. 이들 자료 구조는 커널이 프로세스를 운용하고 IPC 와 같은 다양한 통신 기능을 제공하기 위해서 필요하다.

3 절 다중 우선순위 스케줄러

실시간 시스템의 가장 기본적인 요구 조건은 예측 가능성으로 프로세서들의 제어에 비결정적인 요소가 배제되어야 한다는 것이다 [bodhisattwa93]. 즉, 긴급한 작업을 요하는 event 에 대한 인터럽트 처리는 높은 우선순위를 가지고 매우 신속하게 일어나야 하며 이 인터럽트 처리 루틴의 기능과 실행 시간이 잘 정의되어 있어야 한다. 또한 이런 인터럽트 사이에 발생하는 문맥 전환에는 최소한의 overhead 가 소모되어야 한다. 인터럽트 외에도 중요 도가 높고 마감 시간이 짧은 프로세스도 일반 프로세스보다 높은 우선순위를 주어 먼저 수행되어야 한다. 이와 같이 실시간 시스템에서는 프로세스의 중요 도에 따라서 우선순위를 주게 되며, 지금까지 제안된 모든 실시간 스케줄링 기법은 다중 우선순위 환경을 기본 전제로 하고 있다. 따라서 이들 우선순위에 따라 실행시킬 프로세스를 결정하는 다중 우선순위 스케줄러가 하드웨어나 소프트웨어적으로 요구된다.

1. transputer 의 스케줄러 구조

병렬 시스템의 CPU 로 사용되는 transputer 은 자체에 하드웨어 스케줄러가 내장되어 있다. 그림 33 은 transputer 의 하드웨어 스케줄러의 구조를 보여준다. 하드웨

어 스케줄러는 2 단계의 우선순위를 지원하며 높은 우선순위 큐(HIGHQ)와 낮은 우선순위 큐(LOWQ)를 관리한다. 이들 큐는 그림 34과 같이 메모리상에서 list 의 형태로 유지되며 각 큐의 구성요소들은 준비(ready)상태 프로세스의 Workspace 이다. 이 list 의 front 와 back 을 가리키는 레지스터인 front-pointer(FPtrReg0, FPtrReg1)와 back-pointer(BPtrReg0, BPtrReg1)들을 하드웨어 스케줄러가 사용한다. 이들 한 쌍의 pointer 는 우선순위 큐마다 존재한다.

하드웨어 스케줄러는 각 우선순위마다 우선순위 큐 이외에도 시간대기 큐(TimeQ), 통신대기 큐 (CommQ)를 관리한다. 하드웨어 스케줄러는 우선순위 큐마다 스케줄 정책이 다른데 높은 우선순위 큐에 대해서는 CPU 를 점유한 프로세스의 실행이 끝나거나 시간대기, 통신대기에 의해서 대기상태에 들어가기 전까지는 인터럽트가 발생하지 않는다. 반면 낮은 우선순위 큐에서는 time slice 가 존재한다. 이 time slice 는 대개 2048μsec 인데, 하드웨어 스케줄러는 time slice 만큼 프로세스에게 CPU 를 할당하고 time slice 가 지나면 강제적으로 프로세스에게서 제어 권을 빼앗아 그 프로세스를 낮은 우선순위 큐의 끝에 붙이고 낮은 우선순위 큐의 앞에 있는 프로세스를 실행시키는 round robin 방식을 사용한다. 이때 context switching 은 한 명령문의 실행이 끝나면 바로 CPU 를 넘겨주는 것이 아니고 jump 나 function call 과 같이 저장해야 할 context 가 아주 적은 시점에 발생한다. 낮은 우선순위의 프로세스가 CPU 를 점유하고 있을 때 높은 우선순위의 프로세스가 준비 상태가 되면 실행중이던 프로세스는 즉시 실행이 중단이 되고 프로세스 상태는 특정한 기억 장소(ISL: Interrupt Save Location)에 보관된다. 실행 상태나 준비 상태의 높은 우선순위 프로세스가 있는 경우에는 낮은 우선순위 프로세스는 실행되지 않는다.

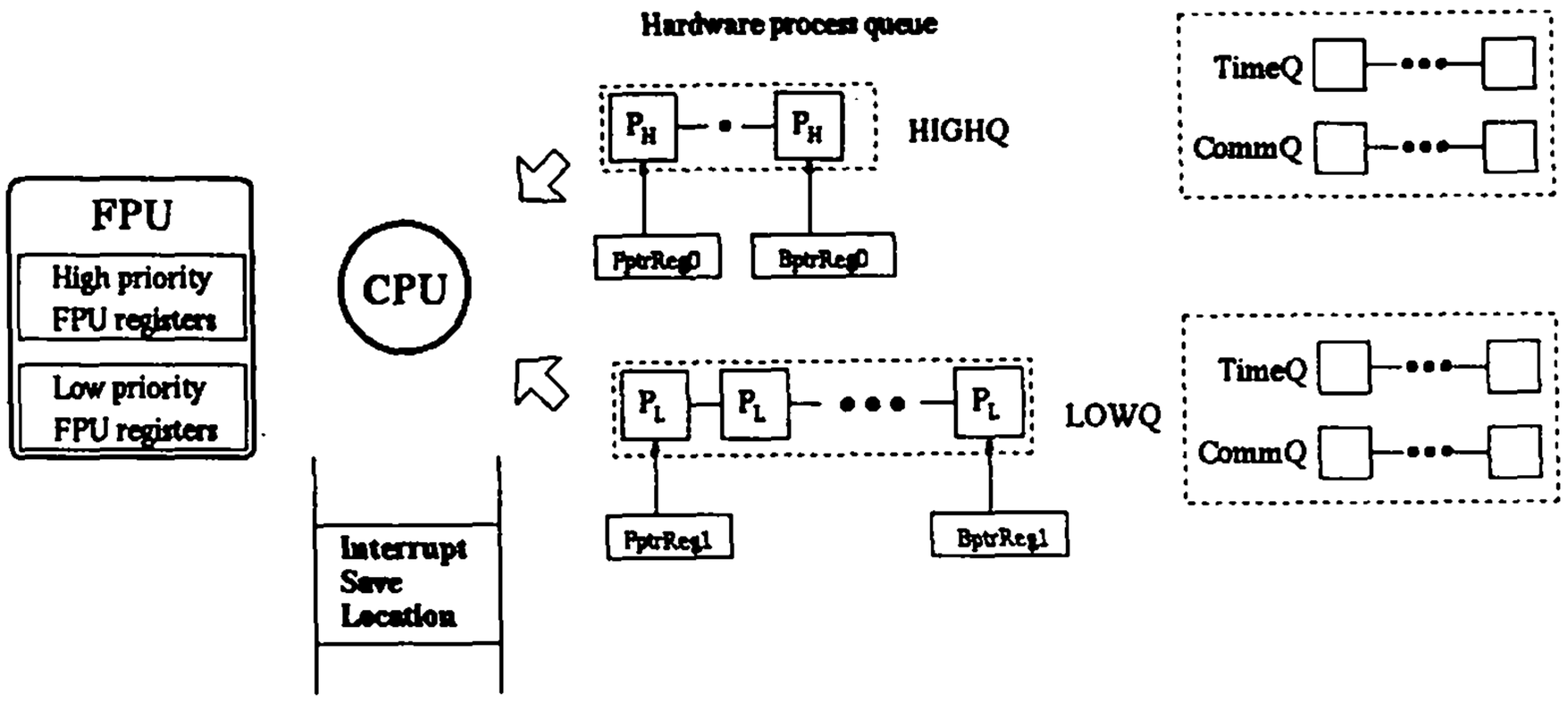


그림 33 transputer 의 하드웨어 큐 구조

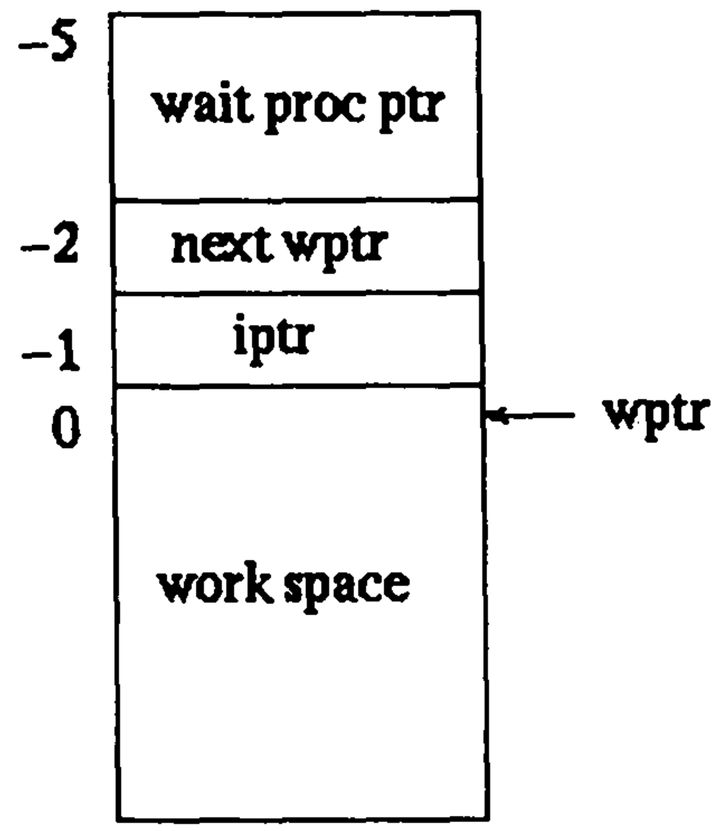


그림 34 프로세스의 Workspace

하드웨어 프로세스 큐는 기억 장치 내에서 list 의 형태로 구현된다. 그림 35는 낮은 우선순위 하드웨어 프로세스 큐의 FPtrReg1 이 큐의 첫번째 프로세스의 workspace(wptr)를 가리키며 첫번째 프로세스 workspace 의 -2 위치(wptr-2)에 두 번째 프로세스의 wptr 를 가리키는 pointer 값이 존재한다. 큐의 마지막 프로세스의 wptr 은 BPtrReg1 가 가리키며 마지막 프로세스의 (wptr-2)값은 정의되지 않는다.

생성되거나 대기상태에서 벗어나 준비 상태가 된 프로세스는 항상 프로세스 큐의 끝에 붙게 된다. 이런 작동 방식 때문에 높은 우선순위 프로세스라 할지라도 낮은 우선순위 back-pointer register 의 값을 마음대로 바꾸지를 못한다. 높은 우선순위 프로세스가 back-pointer register 값을 바꾸는 도중에 낮은 우선순위 프로세스가 준비 상태가 되면 하드웨어 스케줄러가 즉시 현재의 back-pointer register 가 가리키는 기억 장소에 준비 상태의 프로세스를 붙여버리므로 프로세스 큐가 불완전한 상태에 놓이게 되어 시스템이 다운된다.

FPU 는 각 우선순위마다 작업 레지스터들을 두고 있어서 문맥 전환시간을 줄인다. 반면 프로세스는 우선순위가 다른 FPU 레지스터에 접근할 수 없다.

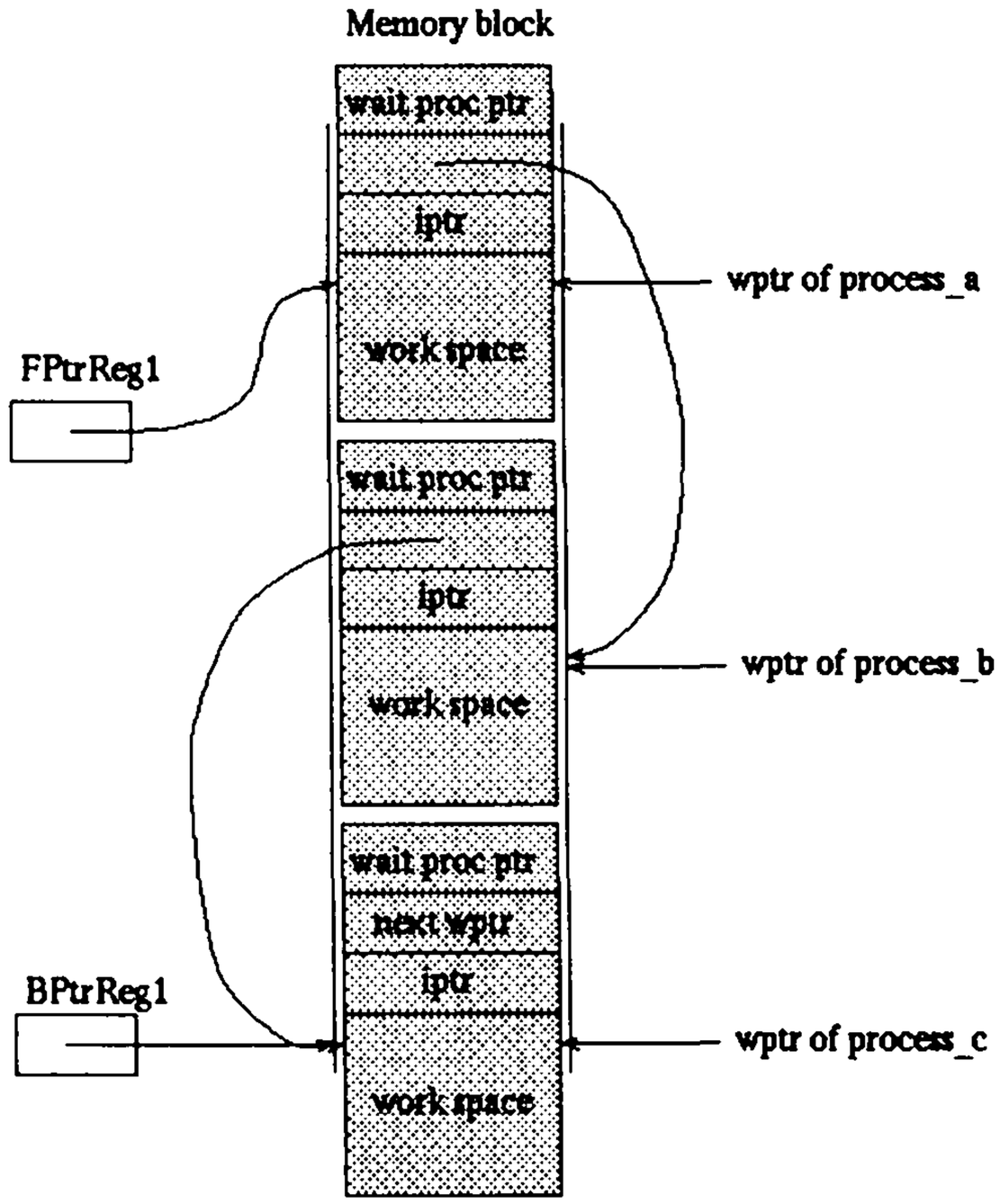


그림 35 하드웨어 큐 구조

2. 이전 연구 및 본 연구의 특징

transputer 자체는 2 단계의 우선순위만을 지원하기 때문에 실시간 응용에 적합하지 않다. 따라서 transputer 상에서 실시간 응용 프로그램을 실행시키기 위해서는 다중 우선순위 스케줄러를 소프트웨어로 구현하고 그 위에서 실시간 응용 프로그램을 시켜야 한다. 이미 transputer 상에서 다중 우선순위 스케줄러를 구현하는 연구가 있어왔다 [shea92,ckmp90,swb90,we90,ploeg94]. 그 중 [ckmp90,swb90,we90]의 스케줄러들은 고 수준 언어인 OCCAM [occam89]으로 구현되어 있으며 이로 인해 새로 준비 상태가 된 프로세스는 우선순위나 긴급함에 관계없이 프로세스 큐의 끝에 붙게 되어 이미 준비 상태인 다른 프로세스들이 CPU 를 양보하기 전까지는 실행할 수가 없었다. 이 때문에 발생하는 시간의 불 예측성을 피하기 위해 응용 프로세스들에게 CPU 를 양보하는 코드를 삽입시켰다. 이 방법은 복잡한 프로그램을 생성할 뿐만 아니라 고 수준 언어로 구현되었기 때문에 효율이 떨어지는 문제가 있었다.

[shea92]의 스케줄러는 낮은 우선순위 큐를 직접 다룸으로써 스케줄러의 효율을 높인다. 하지만 스케줄러가 거의 주기적으로 호출이 되므로 같은 우선순위의 프로세스들만 실행되는 경우와 같이 큐를 다룰 필요가 없는 경우에도 실행 중인 프로세스를 자료 구조에 넣고 자료 구조에서 다시 빼내는 오버 헤드가 존재한다. [ploeg94]의 경우는 control-system 이라는 특수한 환경에서 실행되는 소프트웨어를 제어하므로 일반적인 다중 우선순위 스케줄러와는 거리가 멀다.

[choe95]에서는 프로세스 생성, 소멸, 대기가 발생한 경우, 세마포 대기, 세마포 대기 종료, 통신 대기, 통신 대기 종료 등을 사건이라고 정의하며 사건이 발생하는 경우에만 소프트웨어 스케줄러가 동작하도록 한다. 따라서 스케줄러가 호출되는 횟수를 줄였으며 사건별로 처리하는 코드를 둠으로써 각 사건에서의 오버헤드를 최소화 한다. 하지만 스케줄러가 CPU 제어를 얻는 시점이 round-robin 으로 문맥 전환이 일어나는 시점이므로 선점 지연 시간이 round-robin time-slice 보다 길다는 문제점이

있다.

[cheung95]에서는 선점 지연 시간을 줄이기 위해 round-robin time-slice 가 지나기 전에 프로세스를 선점하는 알고리즘을 사용했다. 이 알고리즘은 현재 우선순위보다 높은 우선순위의 프로세스가 준비 상태가 되면 스케줄러가 현재 실행 중인 프로세스의 코드에 현재상태를 저장하는 코드를 삽입시킴으로써 선점이 가능하도록 한다. 따라서 선점 오버헤드가 증가하는 문제가 있다.

본 연구에는 프로세스가 선점되었을 때 정보를 저장하는 기억 장소인 Interrupt Save Location(ISL)을 이용한 선점 알고리즘을 구현함으로써 [choe95] 스케줄러의 선점 지연 시간을 42 μsec 이내로 줄였다. 선점을 위해서는 ISL 뿐만 아니라 프로세스 큐와 FPU 의 레지스터도 저장을 해야 하며 이를 위해 FPU 의 레지스터를 저장하는 프로세스를 별도로 이용한다.

MPS 는 다음과 같은 세가지 성질에 의해서 스케줄러의 부하와 선점 지연 시간을 줄이게 된다. 첫번째로 MPS 는 하드웨어 프로세스 큐의 크기가 바뀌는 사건이 발생했을 때만 호출되므로 같은 우선순위의 프로세스들이 실행되는 경우와 같이 하드웨어 프로세스 큐의 크기가 바뀌지 않는 경우에는 호출되지 않기 때문에 스케줄러 자체의 호출 횟수가 줄어든다. 두 번째로 스케줄러는 호출되더라도 항상 하드웨어 프로세스 큐를 재조정하지는 않는다. 이 상황은 사건의 종류를 관찰함으로써 쉽게 알 수 있다. 예를 들자면 중단된 프로세스가 대기상태로 바뀌었을 때 실행 중인 프로세스와 같은 우선순위를 가지고 있으면 이 프로세스는 중단 큐에서 바로 하드웨어 프로세스 큐로 이동한다. 이 작업은 하드웨어 프로세스 큐를 재조정하고 자료 구조에서 프로세스를 꺼내어 하드웨어 프로세스 큐에 넣는 것보다 적은 실행 시간을 요구한다. 세 번째로 실행 중인 프로세스의 정보를 저장하는 구조를 사용함으로써 실행 중인 프로세스를 바로 선점하여 선점 지연 시간을 줄였다.

본 연구에서는 2 절에 MPS 의 전체 구조, 알고리즘, 선점 알고리즘을 다루고, 3 절에서는 MPS 의 성능 실험 결과와 다중 우선순위를 지원하는 Run Time Library 의 성능을 다루며 마지막으로 5 절에 결론을 맺고 있다.

3. 사건의 정의 및 종류

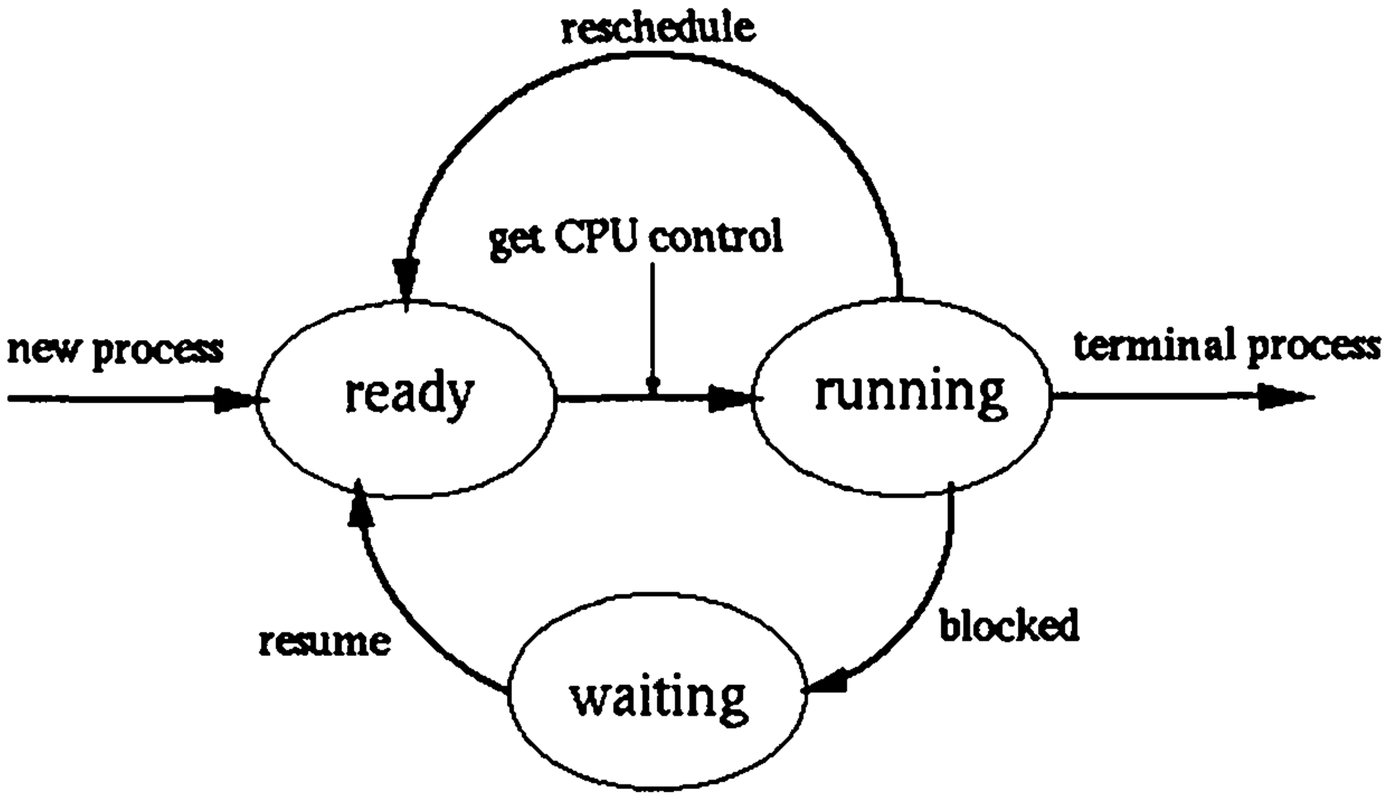


그림 36 Process state diagram

프로세스는 그림 36과 같이 기본적으로 3 개의 state 중 하나의 상태에 있다. 그림 36의 화살표를 따라서 프로세스의 상태가 바뀌게 되는데 이를 `사건 (event)'이라고 정의한다. 사건의 종류는 그림 36과 같이 6 가지로 분류되며 각 사건을 발생시키는 명령문이나 상황은 INMOS-C 의 경우 대략 다음과 같다.

- new process: ProcRun()
- get CPU control: LOWQ 에 대기중인 프로세스가 round-robin 방식에 의해 CPU control 을 얻음
- reschedule: 실행 중인 프로세스가 2048 μsec 동안 CPU 를 사용한 후 다음 준비 프로세스에게 CPU control 을 넘길 때 발생

- blocked: SemWait(), ProcWait(), CommIn/Out()
- resume: SemSignal(), CommIn/Out()시 통신 요청에 의해 block 되어 있던 상대 프로세스가 깨어나는 경우, ProcWait()에 의해 block 되어 있던 프로세스가 대기시간의 경과로 다시 준비 상태가 될 때
- terminal process: Exit()

4. 다중 우선순위 스케줄러의 구조

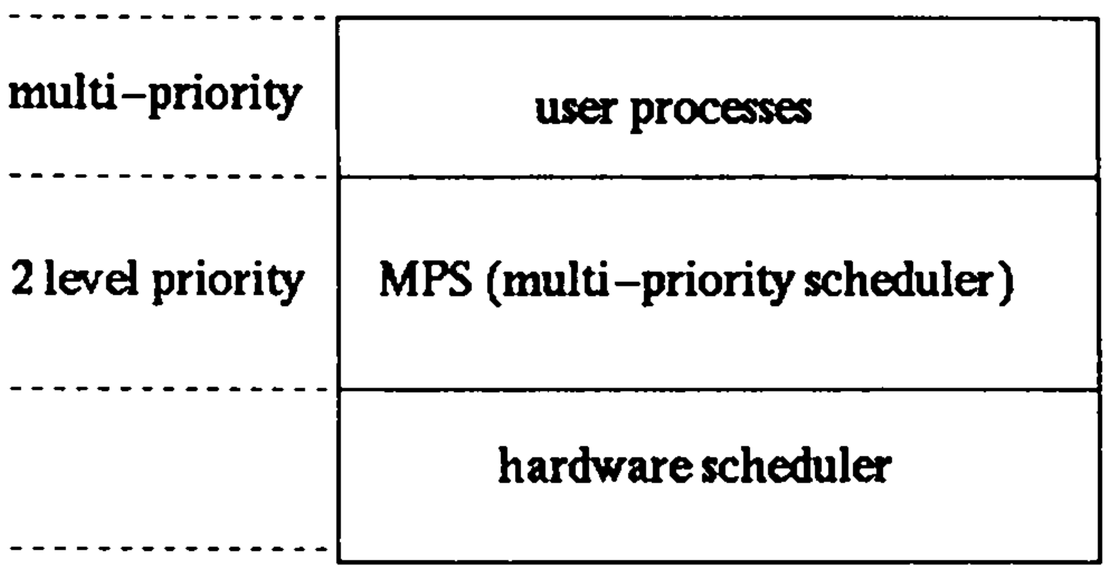


그림 37 시스템 구조

그림 37은 응용 프로세스와 스케줄러, 사건 처리기, 하드웨어 스케줄러들 상호간의 관계를 보여준다. 하드웨어 스케줄러에 의해 실행되는 모든 프로세스들은 2단계 우선순위 환경에서 실행되나 MPS 위에서 실행되는 프로세스들은 MPS 상의 다중 우선순위에 의해서 실행 순서가 결정된다. 그림 38은 하드웨어 스케줄러의 입장에서 본 프로세스들이다. 응용 프로세스들은 실제로 낮은 우선순위 하드웨어 스케줄러에 의해 실행되며 우선순위가 같은 응용 프로세스들은 여러 개가 동시에 실행된다. 이때는 하드웨어 스케줄러의 제어를 받게 되어 round robin 방식으로 스케줄링 된다. MPS 역시 프로세스로 인식된다. MPS 는 낮은 우선순위 프로세스 큐 상의 응용 프로세스들을 제어하기 위해 그림 38과 같이 높은 우선순위 프로세스로서 실행된다.

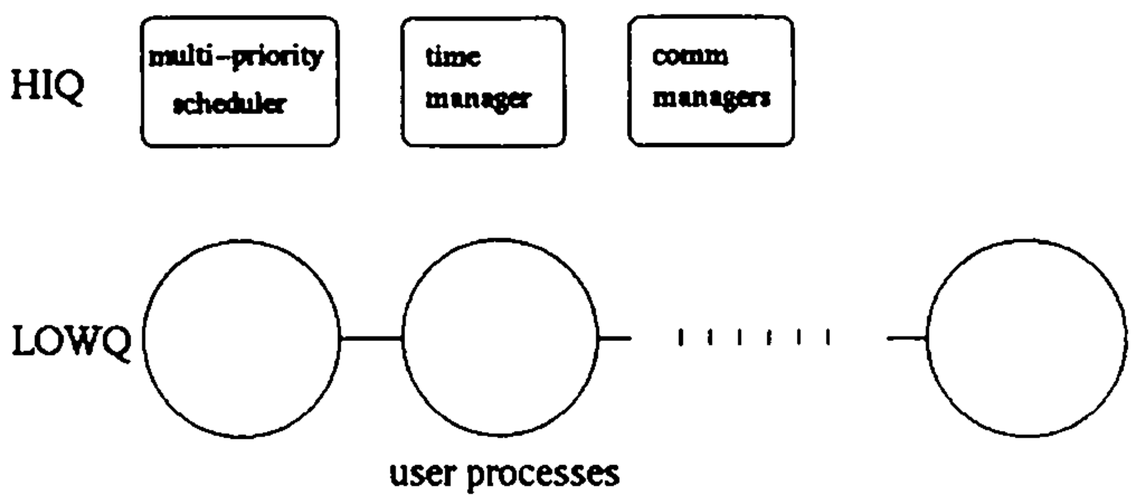


그림 38 하드웨어 스케줄러의 입장에서 본 다중 우선순위 스케줄러와 응용 프로세스들(여기서 HIQ 는 높은 우선순위 프로세스를 LOWQ 는 낮은 우선순위 스케줄러를 나타낸다.)

MPS 는 다중 우선순위 스케줄러로 multi-level queue scheduling 을 하며 각 우선순위마다 프로세스 큐를 둔다. 가장 높은 우선순위를 가진 프로세스들이 round-robin 방식으로 스케줄링 된다. MPS 는 가장 우선순위가 높은 프로세스들을 LOWQ 에 올린다. LOWQ 로 올라간 프로세스들은 2048 μsec time-slice 인 round-robin 방식으로 실행된다. transputer 의 하드웨어 스케줄러는 1 μsec 이하의 문맥 전환 오버헤드를 가진다.

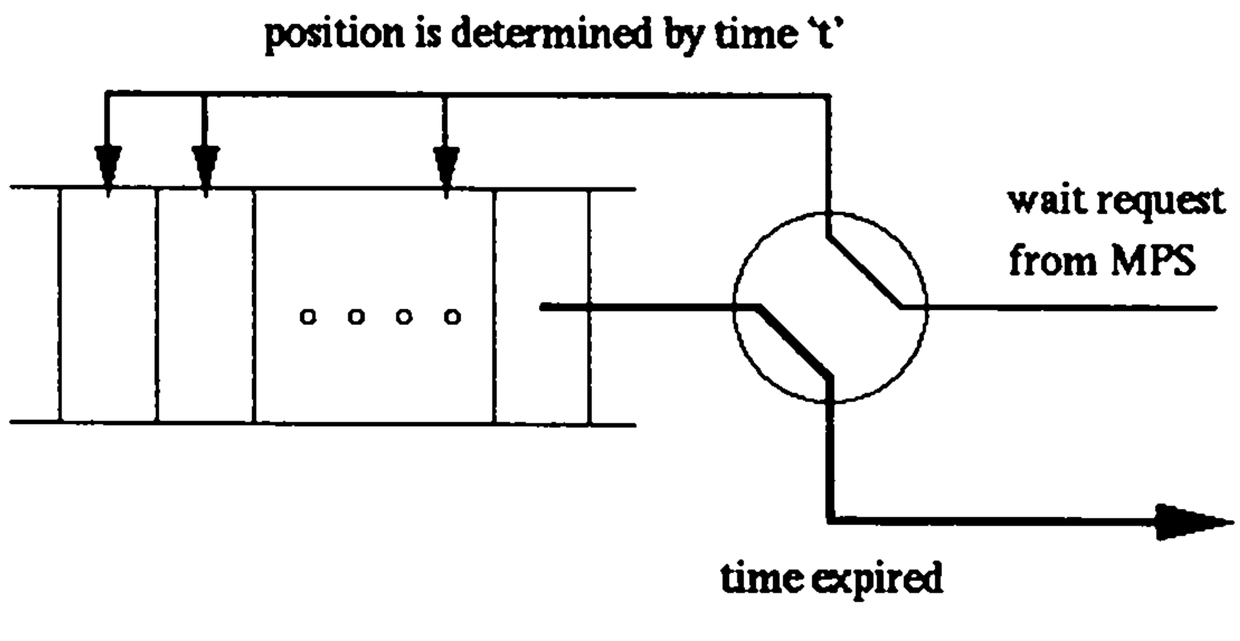


그림 39 시간관리 프로세스의 구조

따라서 하드웨어 스케줄러의 문맥 전환을 이용한다면 상당히 적은 오버헤드로

문맥 전환이 가능하다. 우선순위가 더 낮은 프로세스들은 MPS 의 자료 구조 내에 보관된다. LOWQ 에서 실행되던 프로세스들이 모두 블록 되거나 실행이 끝나면 MPS 의 자료 구조 내에서 우선순위가 가장 높은 프로세스들이 다시 LOWQ 에 올려진다. 현재 LOWQ 에서 실행되는 프로세스들의 우선순위보다 더 높은 프로세스 p 가 준비 상태가 되면 현재 LOWQ 에서 실행되는 프로세스들은 MPS 의 자료 구조에 저장되고 p 가 실행된다. 여기서 get CPU control 과 reschedule 은 transputer 하드웨어 스케줄러가 하게 되므로 MPS 는 new process, blocked, resume, terminal process 등 4 개의 사건만 처리하면 된다.

이들 사건 중 대부분은 함수 호출에 의해서 발생하고 대기시간의 끝을 알리는 사건은 시간 관리 프로세스에 의해 발생한다. 시간 관리 프로세스의 구조는

에 있다. 시간 함수 ProcWait(t)는 이를 호출한 프로세스가 주어진 시간 t 만큼 블록 된 후 준비 상태가 되게 한다. ProcWait()가 호출되면 MPS 는 시간 관리 프로세스에게 $t + t_{current}$ 시각에 자신을 부르도록 요구한다. 시간 관리 프로세스는 MPS 로부터 시각 $t + t_{current}$ 에 대한 요구가 오면 시각 $t + t_{current}$ 와 이 시각까지 대기하는 프로세스 workspace w 를 $t + t_{current}$ 의 오름차순으로 정렬된 큐에 삽입한다. 시간 관리 프로세스는 큐의 가장 앞에 있는 시각까지 기다린 후 그 시각을 큐에서 없애고 MPS 를 호출한다. 다시 시간 관리 프로세스는 큐의 가장 앞에 있는 시각까지 기다린다. MPS 는 블록 된 프로세스를 다시 reschedule 한다. 시간 관리 프로세스는 MPS 와 같은 중요 도를 가지므로 하드웨어적으로 높은 우선순위를 가진다. MPS 의 시간 관리 알고리즘은 그림 40 에 나타나 있다.

```
wait-time <- next_int;
   while forever
      wait until message_come or wait_time out
      if message_come then
         insert_message_to_time_queue
```

```
        recalculate_wait_time
    else if wait_time_out
if time_queue is not empty
    find time_out message from queue and extract message
    send_message to scheduler
    recalculate_wait_time
else
    wait_time <- max_int;
```

그림 40 시간 관리 프로세스의 알고리즘

세마포는 SemWait()와 SemSignal() 함수에 의해 사용되며 실행 중이던 프로세스가 SemWait()를 불렀을 때 세마포 값이 0 이하일 경우 이 프로세스는 블록 되며 그 세마포의 큐에 저장된다. 세마포마다 그림 41과 같은 큐 구조를 가지며 이 큐 구조는 우선순위에 의해 정렬되어 있다. 프로세스가 SemSignal()을 불렀을 때, 세마포 값이 0 미만이면 세마포 큐의 앞에 있는 프로세스가 준비 상태로 된다. MPS 의 세마포 관리 알고리즘은 그림 42 에 나타나 있다.

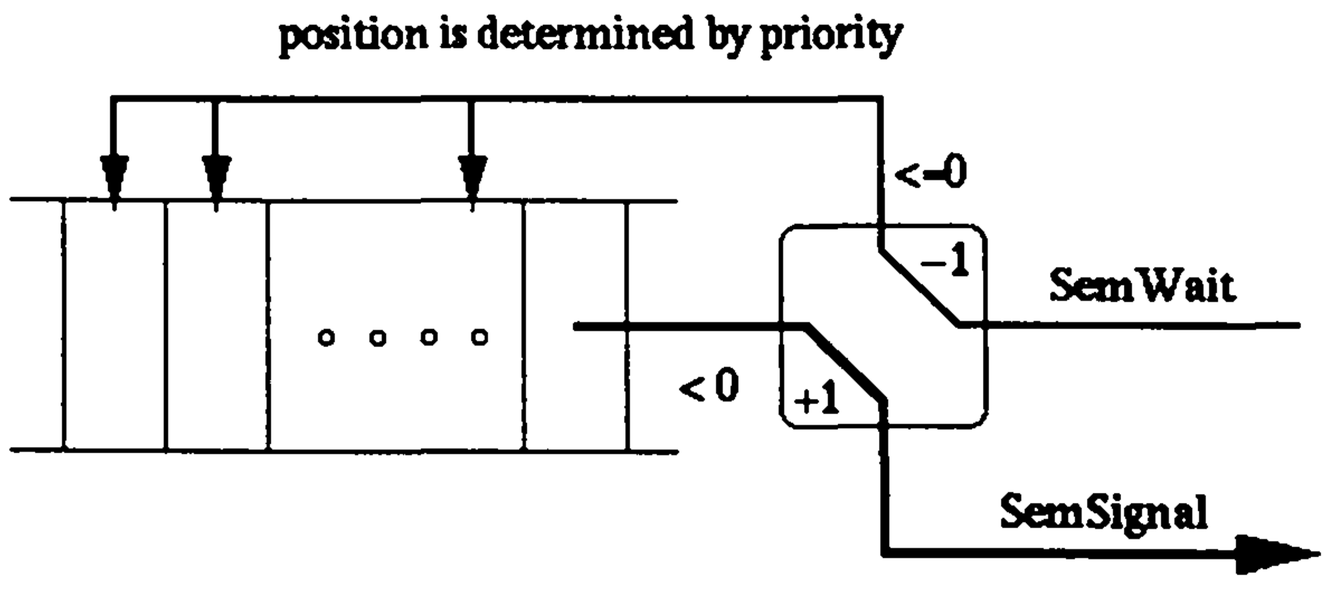


그림 41 각 세마포의 자료 구조

```
while forever
```

```
    wait until message_come
    if content of message is `semaphor_wait.resource' then
resource.size <- resource.size - 1
if resource.size < 0
     insert_message_to_semaphor_queue
     tell_scheduler that a process block
    if content of message is `semaphor_signal.resource' then
resource.size <- resource.size + 1
if resource.size <= 0
     extract_a_message_from_semaphor_queue
     send a process in message to schedule
```

그림 42 세마포 알고리즘

통신을 위해서는 8 개의 높은 우선순위 프로세스들을 사용한다. transputer 에는 4 개의 외부 통신 link 가 있으며 각 link 는 Input link 와 Output link 가 있다. 따라서 한 transputer 에 그림 43과 같이 8 개의 link 가 있으며 각 link 에 하나의 프로세스가 입력이나 출력을 담당한다. 출력의 경우에는 상대편 link 가 받을 준비가 될 때까지 통신을 보내는 프로세스가 블록이 된다. 응용 프로세스가 직접 통신을 기다리며 블록이 되면 MPS 의 제어 권에서 벗어나므로 스케줄러가 통신을 처리해야 한다. 하지만 스케줄러 자신이 통신을 기다리면 그 동안에 다른 처리를 할 수 없으므로 통신을 대신 기다리는 프로세스가 필요하다. 입력의 경우는 응용 프로세스가 입력을 요구할 때 뿐만 아니라 현재 프로세서를 경유하는 packet 이 있을 경우도 있으므로 항상 입력을 기다려야 한다. 입력된 packet 의 목적지가 현재 프로세서 내의 프로세스인 경우에는 이 프로세스가 실제로 packet 의 입력을 요구할 때까지 기다린 후 이 프로세스에게 packet 을 넘겨준다. 단순히 경유를 위한 packet 인 경우에도 응용 프로세스 중에 라우팅을 위한 프로세스가 있을 것이므로 그 프로세스에게 packet 을 넘겨준다.

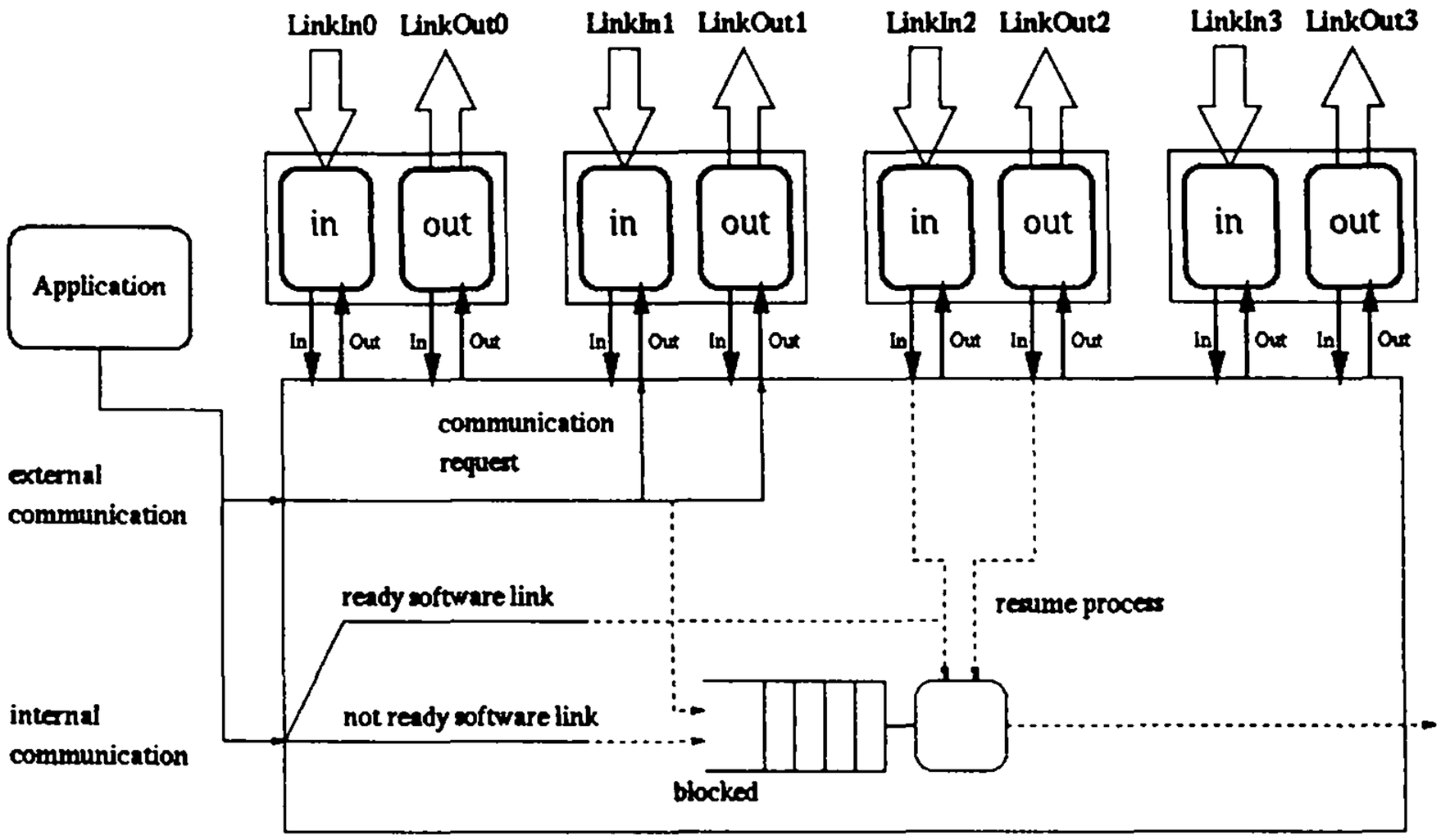


그림 43 MPS 내의 통신 관리를 위한 모듈

그림 44는 MPS 가 동작하는 개요를 보여주고 있다. 준비 상태에 있는 프로세스들은 다중 프로세스 큐에 있으며 가장 우선순위가 높은 우선순위 큐를 하드웨어 스케줄러가 직접 다루며 이는 LOWQ 와 일치한다. 이보다 우선순위가 낮은 프로세스들은 MPS 의 자료 구조 내에 저장된다. 이 자료 구조는 그림 44에서 보는 바와 같이 같은 우선순위를 가진 프로세스들이 하나의 리스트를 형성한다. 여기서 보관되는 프로세스들은 준비 상태에 있으며 우선순위가 낮아서 하드웨어 스케줄러에 의해 실행되지 않는다. 하드웨어 스케줄러는 LOWQ 를 라운드 로빈 방식으로 스케줄하며 그동안 소프트웨어 스케줄러는 호출되지 않으므로 오버헤드가 발생하지 않는다. 실행 중이던 프로세스가 ProcWait(), SemWait(), 통신 등으로 블록 될 시점이면 MPS 는 그 프로세스를 블록 되는 프로세스가 저장되는 자료 구조에 넣는다. 프로세스의 블록이 끝나면 resume 되어 LOWQ 나 MPS 의 자료 구조에 삽입된다. 또한 MPS 는 LOWQ 와 CPU 에 프로세스가 없으면 자료 구조로부터 LOWQ 와 CPU 로 프로세스들을 이동시키고 더 높은 우선순위의 프로세스에 의한 선점이 필요할 때는 현재

LOWQ 와 CPU 의 프로세스들을 자료 구조로 옮기는 context switch 도 담당한다. 이 때 CPU 의 정보는 ISL(Interrupt Save Location)에 저장되어 있으므로 ISL 의 정보를 이동시킴으로써 CPU 의 상태를 유지하고 FPU 의 레지스터의 값들도 저장하거나 복구하게 된다.

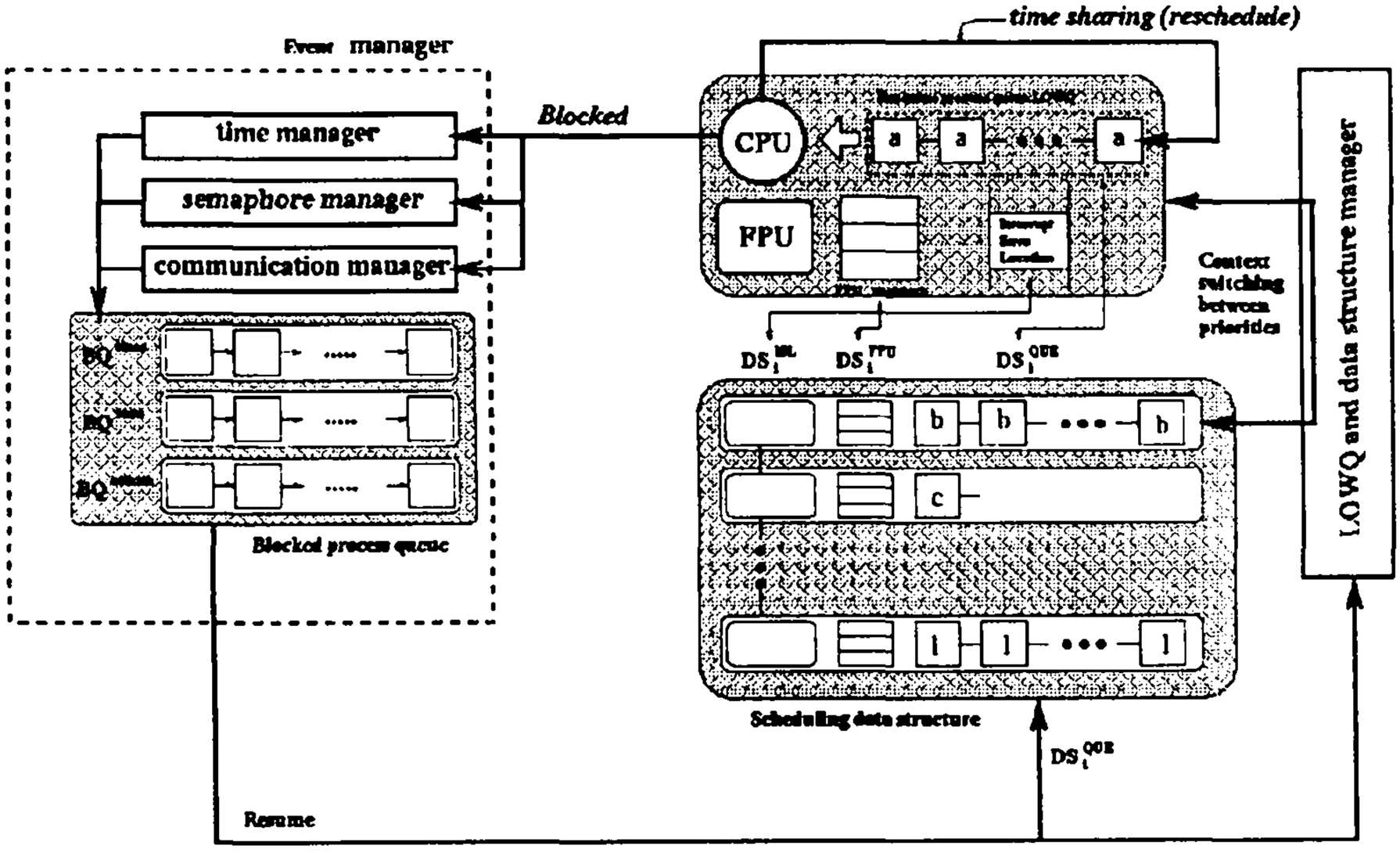


그림 44 스케줄러 동작의 개관: n 단계의 우선순위가 있는 시스템

5. 스케줄링 알고리즘

1. if a process is to be blocked

    a. if blocking reason == wait-time

        (1) insert the process into wait-time blocked process queue.

    b. if blocking reason == semaphore

        (1) insert the process into semaphore blocked process queue.

    c. if blocking reason == communication

        (1) insert the process into communication blocked process queue.

    d. if the number of processes in LOWQ is zero

(1) inserts all processes of the software process queue of

the highest priority level into LOWQ

(2) if there is a preempted process for the same priority level

.restore FPU registers and status

.restore Interrupt Save Location

2. if a process is to be ready-to-run from blocked or a new process is to be created

a. remove the process from the corresponding blocked process queue

if the process is blocked.

/* the Pnew: priority of the process */

/* the Pcur: priority of currently running process */

b. if Pnew < Pcur

(1) insert the process into the software process queue of Pnew.

c. else if Pnew == Pcur

(1) attach the process to the end of LOWQ

d. else if Pnew > Pcur

(1) insert all processes in LOWQ into the software process queue

of Pcur and clear LOWQ.

(2) save the content of Interrupt Save Location to the corresponding

save area for interrupt process context information.

(3) clear Interrupt Save Location memory area.

(4) save FPU information such as FPU registers and status.

(5) insert the process to the end of LOWQ.

3. if a process is to be terminated

a. do the same execution steps in 1-d.

그림 45 제안된 스케쥴러의 전체적인 알고리즘

다중 우선순위 스케줄러는 **오류! 참조 원본을 찾을 수 없습니다.**과 같이 동작한다. 스케줄러는 각 사건의 종류에 따라 그에 적절한 행동을 취하게 된다. MPS 가 다루는 사건은 크게 프로세스가 우선 순위 큐에서 빠져나가는 경우 (blocked, terminal process)와, 우선순위 큐로 들어오는 경우 (resume, new process) 두 가지로 나뉜다. 한 프로세스가 우선순위 큐로 들어오는 경우는 그 프로세스의 우선순위와 우선순위 큐 내의 가장 높은 우선순위에 의해 3 가지 경우로 다시 나누어진다. 프로세스가 우선순위 큐에서 나가는 경우는 CPU 에서 실행되던 프로세스가 실행이 중단되는 경우로 우선순위가 가장 높은 프로세스 큐, 즉 하드웨어 프로세스 큐의 길이가 0 인 경우와 0 보다 높은 경우 두 가지로 나누어진다.

□ 준비 상태가 된 프로세스 우선순위가 실행 중인 프로세스의 우선순위보다 낮은 경우

새로 들어온 프로세스는 실행 중인 프로세스 큐보다 우선순위가 낮은 큐에 삽입된다. 이 큐는 스케줄러의 자료 구조에 존재하므로 프로세스는 하드웨어 큐에 삽입되지 않고 바로 자료 구조로 들어간다. 따라서 이때는 스케줄러가 하드웨어 큐를 다룰 필요가 없다.

□ 준비 상태가 된 프로세스의 우선순위가 실행 중인 프로세스의 우선순위와 같은 경우

이 경우는 준비 상태가 된 프로세스가 가장 높은 프로세스이므로 하드웨어 프로세스 큐의 끝에 붙게 된다. 이때도 스케줄러는 큐를 다룰 필요가 없다. 또한 우선순위 큐 전체가 비게 되어 하드웨어 프로세스 큐에 아무런 프로세스가 없게 될 경우에도 새로운 프로세스는 하드웨어 프로세스 큐의 끝에 붙게 된다.

□ 준비 상태가 된 프로세스의 우선순위가 실행 중인 우선순위보다 높은 경우

이때는 실행 중인 프로세스를 인터럽트하고 하드웨어 프로세스 큐에 있는 프로세스들과 실행이 중단된 프로세스를 소프트웨어 스케줄러의 자료 구조에 넣는다. 준비 상태가 된 프로세스는 하드웨어 프로세스 큐의 끝에 붙게 되고 다른 프로세스가 하드웨어 프로세스 큐에 없으므로 곧바로 실행된다.

□ 실행 중이던 프로세스가 종료하거나 대기상태가 되고 하드웨어 프로세스 큐가 비어있는 경우

하드웨어 프로세스 큐에는 실행될 프로세스가 없으므로 스케줄러의 자료 구조가 프로세스가 1 개 이상 있다면 자료 구조 내의 프로세스 큐 중 우선순위가 가장 높은 우선순위 큐가 하드웨어 프로세스 큐로 옮겨진다. 이 때는 각 큐의 프로세스를 앞부분부터 하나씩 꺼내어 하드웨어 프로세스 큐의 끝에 옮겨지며 이 동작은 transputer 어셈블리 명령인 runp 명령으로 구현한다.

6. 선점 알고리즘

그림 46은 준비 상태가 된 프로세스의 우선순위가 실행 중인 프로세스의 우선순위보다 높은 경우의 예를 보여준다. 시간은 (a),(b),(c), ... ,(h)의 순으로 경과하며 가장 왼편의 사각형은 Interrupt Save Location 의 상태를, 그 오른편의 사각형은 CPU 를, 그리고 오른편의 원들은 LOWQ 에 있는 프로세스들을 나타낸다. 현재 n 개의 프로세스가 있고 프로세스 lp_1 이 실행 중이다 (a). 이들의 우선순위보다 높은 우선순위를 가진 hp 가 준비 상태가 되면 스케줄러가 lp_1 을 선점하고 실행 중이던 lp_1 의 context 는 Interrupt Save Location 에 저장된다 (b). 스케줄러는 Interrupt Save Location 과 LOWQ 의 프로세스들을 모두 스케줄러 자료 구조에 저장하고 (c), FPU register 를 저장하기 위한 프로세스인 save_fpr()를 Interrupt Save Location 로 옮긴다 (d). 스케줄러는 스스로 중지되고 save_fpr()이 마치 interrupt 되었던 것처럼 실행을 계속하며 FPU register 의 값들을 저장하고 스케줄러에 자신의 작업이 끝났음을 알린다 (f). 스

케줄러는 save_fpr()를 Interrupt Save Location 에서 제거하고 hp 를 LOWQ 에 붙인다 (g). 스케줄러가 중지되면 hp 는 실행을 시작한다 (h).

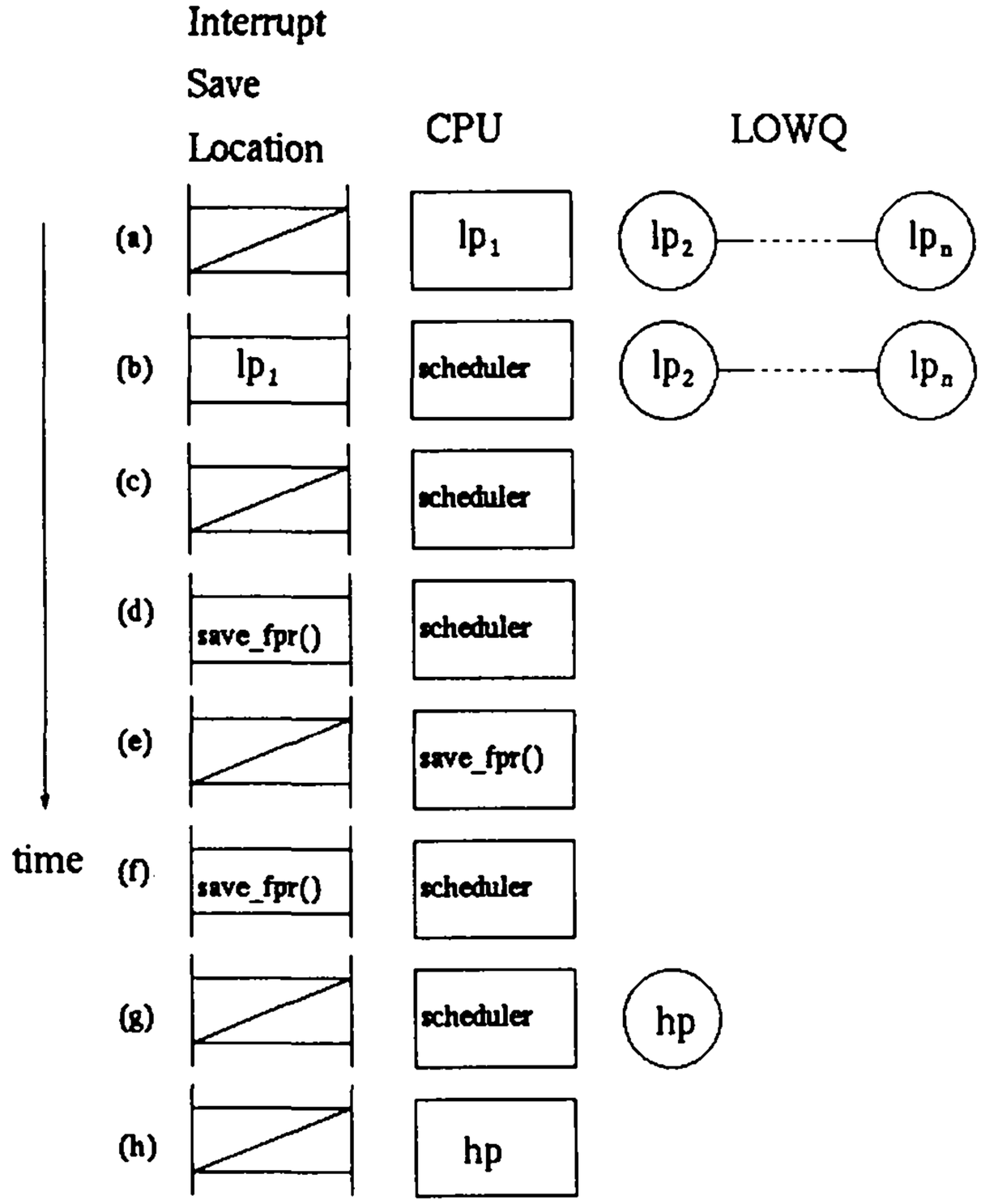


그림 46 전체 알고리즘의 스텝 1-c 에서 낮은 우선순위 프로

세스들을 자료 구조로 옮기는 알고리즘

여기서 transputer 의 행동에 비일관성이 있음을 주의해야 한다. 일반적으로 스케줄러는 응용 프로세스가 보낸 메시지를 받음으로써 그 응용 프로세스를 선점하게 되는데 선점된 프로세스는 ISL 에는 정보가 항상 있지만 간혹 LOWQ 의 끝에 workspace 가 붙어 버리는 경우가 발생한다. 정확한 원인은 알 수 없지만 실행 중인 프로세스가 reschedule 되려는 시점에 선점이 발생해서 이런 문제가 생겼다고 본다

이런 행동은 두 가지 경우에 문제를 일으키는데 하나는 실행 중인 프로세스가 ProcWait()나 CommIn/Out 등으로 블록이 되려고 할 때이다. MPS 가 ISL 에 있는 프로세스만 블록된 프로세스들의 자료 구조에 보관하게 되면 LOWQ 의 끝에 붙은 동일한 프로세스가 실행을 계속해서 동일한 프로세스 2 개가 존재하는 현상이 생길 것이다. 다른 경우는 우선순위가 더 높은 프로세스에 의해 선점이 일어날 때이다. 이 때 MPS 는 실재로 자료 구조로 이동하는 프로세스의 수보다 하나 많은 프로세스가 자료 구조로 이동한 것으로 잘못 판단한다. 또 중복된 프로세스 뒤에 다른 프로세스가 붙은 뒤 ISL 에 있던 프로세스가 LOWQ 의 끝에 붙게 되면 사이에 있던 프로세스는 사라지게 된다. 따라서 이 두 가지의 경우에 중복된 프로세스가 있는지 확인한 뒤 그 프로세스를 제거한다.

기본적으로 선점에 의해 문맥 전환이 발생할 때 이동하는 정보는 CPU 레지스터, Iptr, Wptr, 실행 중인 명령문의 정보, LOWQ, FPU 레지스터들이다. LOWQ 와 FPU 레지스터를 제외한 정보는 모두 ISL 에 있으므로 ISL 을 그대로 저장하면 되고 LOWQ 는 프로세스 list 를 저장하면 된다. 남은 것은 FPU 레지스터인데 응용 프로세스는 모두 LOWQ 에서 실행되므로 낮은 우선순위 FPU 레지스터를 저장해야 한다. 하지만 MPS 는 높은 우선순위 프로세스이므로 낮은 우선순위 FPU 레지스터에 접근할 수가 없다. 이 문제를 해결하기 위해 FPU 레지스터의 저장과 복구만을 담당하는 낮은 우선순위 프로세스를 생성해서 선점의 마지막 단계에 이 임시 프로세스를 실행시킨다. 실행시키는 과정은 그림 46의 (d)에서 (f)까지이다.

7. 선점 방식의 개선

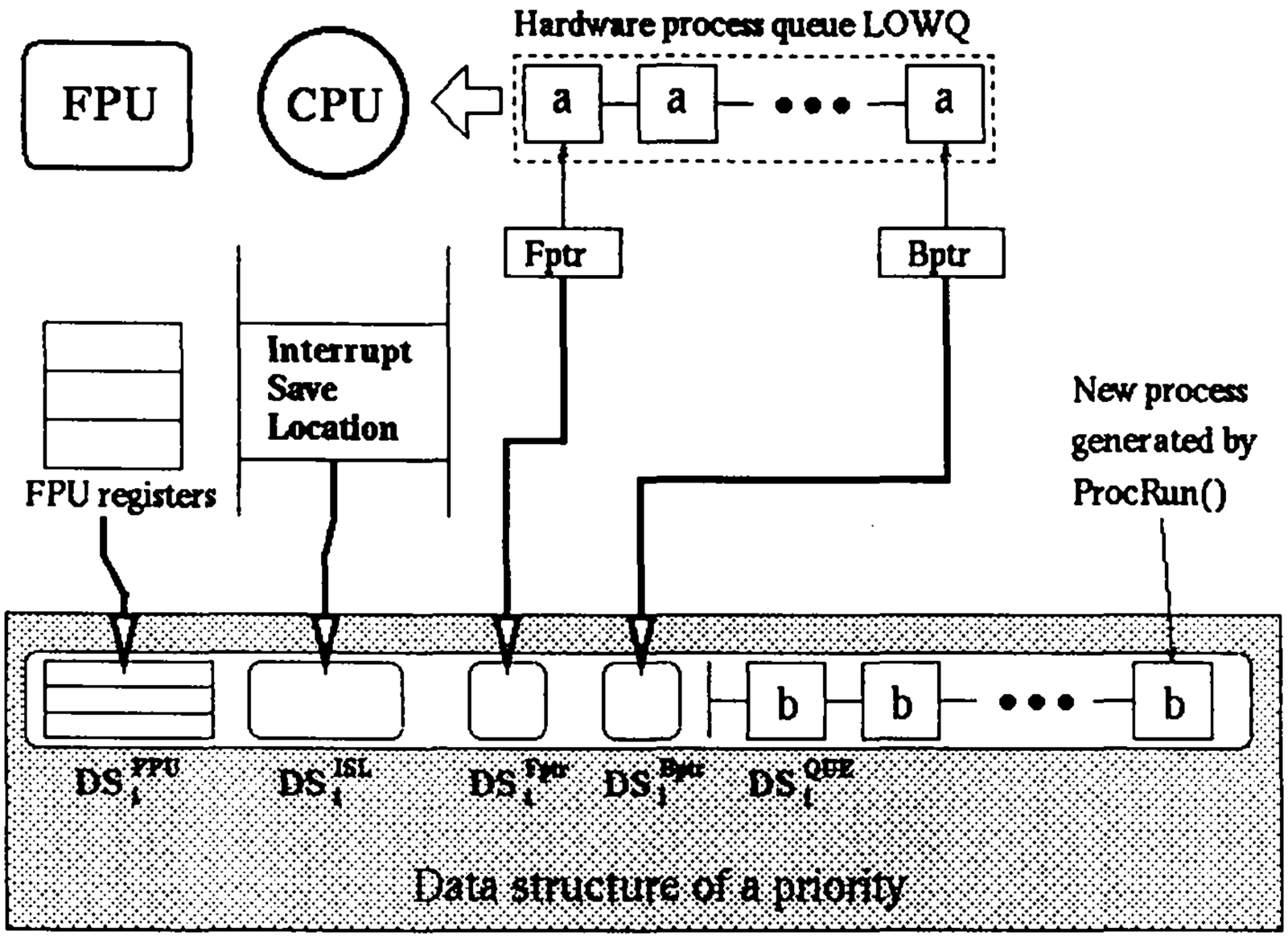


그림 47 한 우선순위의 프로세스들을 보관하는 자료 구조

기존의 방식은 higher priority 프로세스에 의해 선점된 프로세스들과 실행이 연기되고 있는 프로세스들이 모두 스케줄러 자료 구조의 list 구조 안에 저장된다. 이때, 각 process 의 Workspace pointer 인 Wptr 이나 process pointer 들은 MPS 가 할당한 메모리에 저장된다. 하지만 실제로 transputer 에서 선점된 프로세스들은 메모리에 자신의 Workspace 를 보유하고 있다. 즉, MPS scheduler 와 hardware scheduler 가 중복되는 자료 구조를 갖는다. 하지만 이 Workspace 자체가 linked list 로 연결할 수 있는 구조이므로 선점된 프로세스의 Workspace 를 그대로 이용하는 것이 메모리 낭비를 줄일 수 있다. LOWQ 에 있는 프로세스들은 Fptr 과 Bptr 로 Workspace list 의 시작과 끝을 알리고 Workspace 내부에 자신의 다음에 오는 프로세스의 Workspace 를 가리키는 pointer 가 존재한다. 이러한 특징을 이용하면 선점과 복구가 발생했을 때 수행해

야 할 오퍼레이션의 수가 줄어든다.

그림 47는 preemption 이 발생했을 때, 스케줄러가 저장해야 할 정보들을 나타낸다. FPU register 와 Interrupt Save Location 은 각각 DS_i^{FPU}, DS_i^{ISL} 에 저장된다. LOWQ 의 경우는 process 들의 Workspace 가 저장되는 대신 Fptr 과 Bptr 이 DS_i^{Fptr}, DS_i^{Bptr} 에 저장된다. Block 되어 있던 프로세스 pi 가 다시 ready 상태가 되었을 때, 현재 priority 가 자신의 priority i 보다 높으면 data structure 내의 자신과 같은 priority i 의 구조 내에 저장된다. 이때는 DS_i^{Bptr} 가 가리키는 Workspace 다음에 list 로 연결하고 DS_i^{Bptr} 은 Pi 의 Workspace 를 가리켜서 Workspace list 의 끝에 pi 가 붙게 한다.

Priority i 의 process 들이 복구될 때는 preemption 과 반대 과정을 거친다. 여기서 Priority i 의 자료 구조 끝부분에 있는 list 는 ProcRun()등에 의해 새로 생성된 프로세스이다. 이들 프로세스는 아직 실행되지 않았기 때문에 Workspace 를 가지고 있지않다. 따라서 실행에 필요한 정보들이 이 list 에 저장된다. Priority i 가 복구될 때 이들 list 에 있는 process 들은 LOWQ 의 끝에 붙게 된다. 스케줄러의 자료 구조를 이와 같이 변경함으로써 priority 가 같은 process 가 많은 경우 preemption, restore overhead 가 줄게 된다.

8. 실험 결과

실험 환경

스케줄러는 INMOS ANSI C [ansi_user,ansi_ref]과 transputer 어셈블러 [trans_inst]으로 구현했으며 성능 비교를 위해 [shea92]의 스케줄러를 INMOS C 로 바꾸어 실행했다. 실험에 사용된 transputer 은 T800-20MHz 로 2Mbyte 의 메모리를 가진다. 실험 결과는 스케줄러를 10 번 실행한 값들의 평균을 사용했다. 실행 횟수는 적지만 결과 값들의 분산 값은 1 μsec 이내를 나타낸다. 실험에 사용된 응용 프로세스들은 주기

적인 프로세스들로 rate monotonic algorithm[liu73]에 의해 우선순위가 할당되었다. 공정한 비교를 위해 같은 실험 환경상에서 [shea92]을 실행시켰다. 전체 시간에서 프로세스가 CPU를 점유한 시간의 비율을 50\%와 90\%로 두었으며 우선순위의 분포를 3가지 경우로 나누었다. 즉 각 프로세스가 다른 우선순위를 가지는 경우, 5개의 우선순위 단계가 있는 경우, 모든 프로세스가 같은 우선순위를 가지는 경우 등이다.

가. MPS의 성능 측정

그림 48과 표 1에 실험 결과가 나타나 있으며 [shea92]의 알고리즘과 비교해 볼 때 모든 경우에 더 나은 성능을 보여주고 있다. [shea92]의 알고리즘은 모든 경우에 거의 일정한 오버헤드를 나타내고 있는데 이는 호출되는 상황에 관계없이 하드웨어 프로세스 큐의 재설정과 프로세서의 이동이라는 거의 동일한 작업을 수행하기 때문이다.

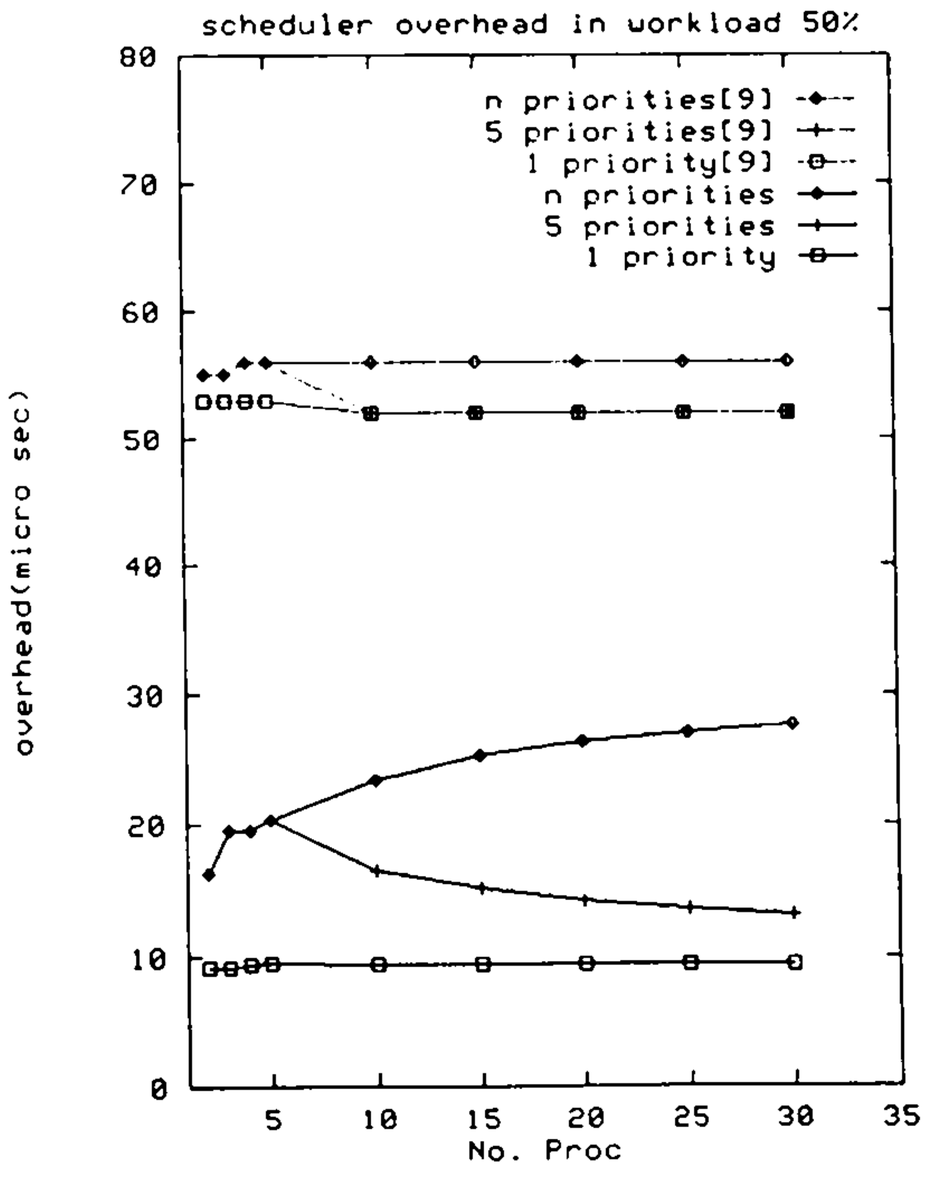


그림 48 프로세스의 부하가 50%일 때의 스케줄러 오버헤드

그림 48과 표 1은 스케줄러는 모든 프로세스들이 같은 우선순위를 가지는 환경에서보다 모든 프로세스들이 다른 우선순위를 가지는 환경에서 더 많은 오버헤드를 나타내고 있음을 보여준다. 특히 30개의 프로세스가 실행되고 프로세스의 CPU 점유율이 50\%일 때는 우선순위가 서로 다를 때가 우선순위가 서로 같을 때에 비해서 2배 이상의 스케줄러 오버헤드를 나타낸다. 반면에 Shea et. al의 스케줄러는 프로세스들의 우선순위의 차이가 실행 시간에 어느 정도 영향을 미치기는 하지만 제안된 스케줄러 만큼의 차이는 보이지 않고 있다. Shea et. al의 스케줄러는 실행 시간의 차이의 대부분은 스케줄러 자료 구조의 리스트 검색 시간에서 비롯되었다고 볼 수 있다. 하지만 본 보고서에서 제안된 스케줄러는 리스트 검색 시간 이외에 다른 요인이 작용한다고 보아야 한다. 실제로 제안된 알고리즘은 표1과 같이 스케줄러가 호출되는 상황에 따라 실행되는 명령어들이 다르게 나타난다. 우선순위의 차이에 의해서 스케줄러가 호출되는 상황에도 변화가 생기게 된다.

스케줄러 호출 상황		1-우선순위		n-우선순위	
사건 형태	우선순위 /큐 길이	횟수	평균시간	횟수	평균시간
준비	Pnew < Pcur	30	26.43	620	33.05
	Pnew = Pcur	734	9.15	55	8.95
	Pnew > Pcur	0	0.00	51	28.55
블록	LOWQ가 비어있지 않음	752	6.14	54	6.11
	LOWQ가 비어있음	11	34.55	670	21.39

표 1 프로세스의 CPU 부하가 90%이고 프로세스 수가 30 개 일 때, 1-우선순위 단계와 n-우선순위 단계에서 각 사건에 따른 스케줄러 오버헤드의 비교. Pnew 는 준비 상태가 된 프로세스의 우선순위이고 Pcur 는 현재 실행되고 있는 프로세스의 우선순위를 나타낸다

표 1은 각 사건 형태에 따른 스케줄러 오버헤드를 나타낸다. 본 실험은 한 프로세스가 준비 상태가 됐을 때와 블록 됐을 때의 스케줄러의 오버헤드를 나타내고 있다. 프로세스 준비의 경우에 대기상태에서 막 준비 상태가 된 프로세스가 현재 실행 중인 프로세스보다 낮은 우선순위를 가진 경우에는 프로세스가 스케줄러의 자료 구조에 삽입된다. 이때의 오버헤드는 약 33.05 μsec 이다. 반면 실행 중인 프로세스와 같은 우선순위를 가진 경우에는 준비 상태가 된 프로세스는 LOWQ 의 끝에 붙게 된다. 이때의 오버헤드는 약 8.95 μsec 이 된다. 이 프로세스가 실행 중인 프로세스보다 높은 우선순위를 가진 경우에는 LOWQ 에 있는 프로세스들을 모두 자료 구조로 옮기고 새 프로세스를 실행시킨다. 이때의 오버헤드는 약 28.55 μsec 이 된다. 모든 프로세스들이 같은 우선순위를 가지면 준비 상태가 된 프로세스는 모두 LOWQ 에 붙게 된다. 반면 모든 프로세스들이 다른 우선순위를 가지면 다른 두 가지 경우가 많이 발생하게 되므로 같은 우선순위의 경우보다 많은 오버헤드를 가지는 작업을 수행한다.

프로세스 대기가 시작되는 경우에는 다른 프로세스가 LOWQ 에 남아 있는 경우와 그렇지 않은 경우가 있는데 전자의 경우에는 별도의 작업이 필요 없지만 후자의 경우에는 스케줄러의 자료 구조로부터 실행될 프로세스들을 LOWQ 로 이동시켜야 하기 때문에 21.39 μsec 정도의 시간을 요하는 작업이 필요하다. 모든 프로세스들의 우선순위가 같은 경우에는 대부분의 프로세스들이 LOWQ 에 있기 때문에 별도의 작업이 필요하지가 않지만 모든 프로세스들의 우선순위가 다른 경우에는 항상 LOWQ 가 비게 된다. 즉, 같은 경우에 스케줄러가 호출되더라도 우선순위의 차이에

의해서 오버헤드가 달라지게 된다. 특히 우선순위가 다른 경우에는 비교적 많은 실행 시간을 요구하는 작업이 대부분이므로 우선순위가 같은 경우보다 많은 오버헤드를 나타낸다.

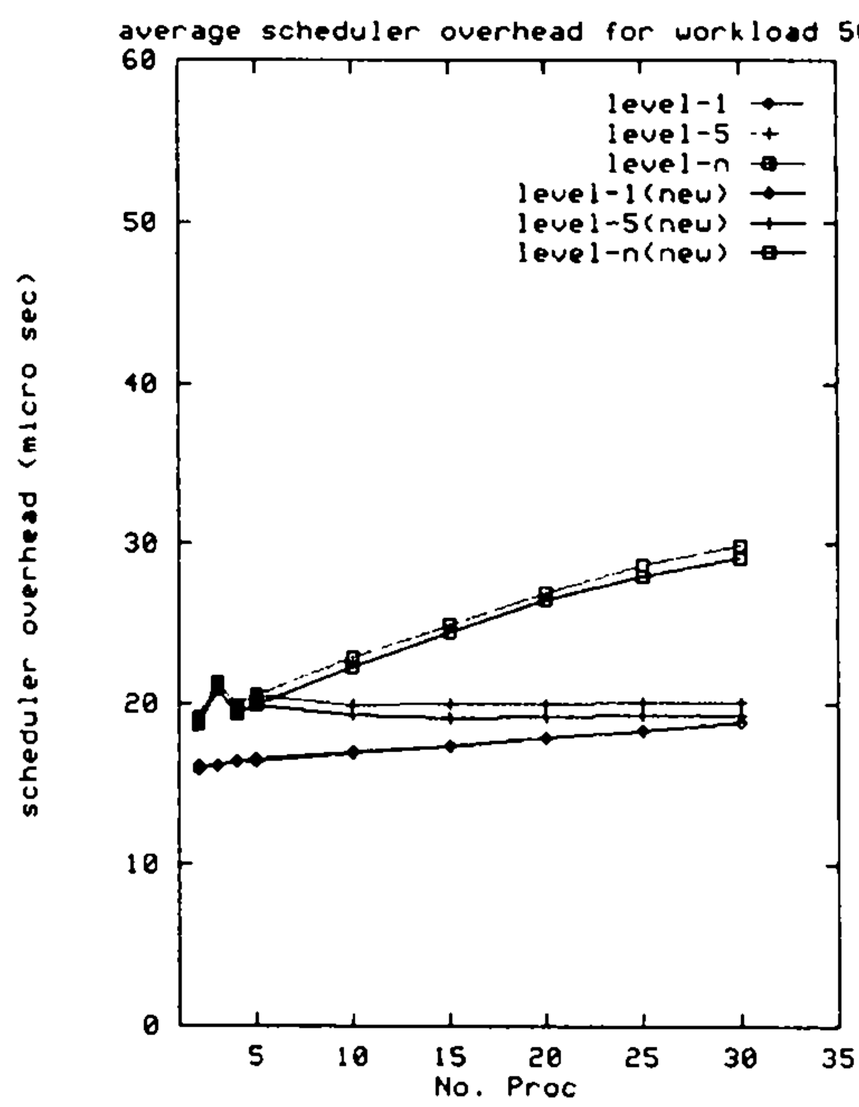


그림 49 리스트 자료 구조와 Fptr, Bptr 을 사용했을 때의 각각의 스케줄러 오버헤드

나. 개선된 선점 방식의 성능 측정

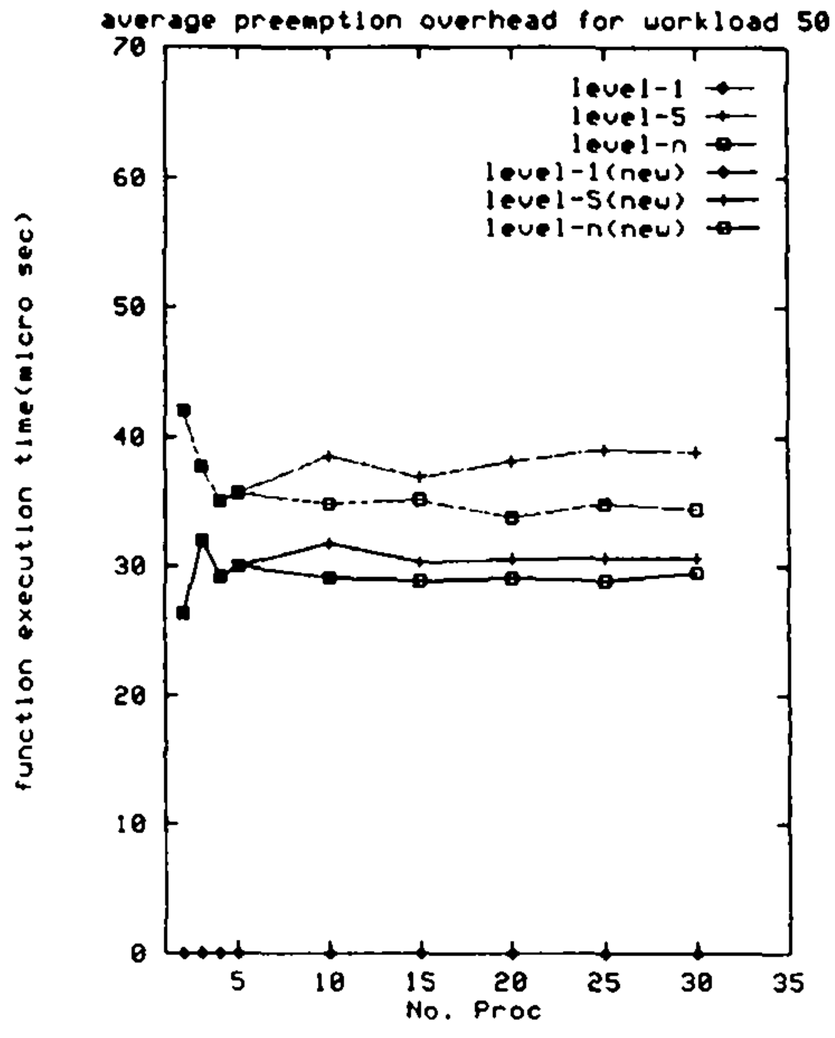


그림 50 리스트 자료 구조와 Fptr,Bptr 을 사용했을 때의 각각의 선점 지연 시간

그림 49과 그림 50은 list 를 저장한 기존의 자료 구조와 Fptr/Bptr pointer 를 저장한 자료 구조간의 성능차를 나타낸다. 1-priority level 에서는 preemption 이 발생하지 않기 때문에 성능차가 없고, n-priority level 에서도 LOWQ 에 process 가 없으나 기존의 알고리즘은 중복되는 Workspace 를 없애기 위해 LOWQ 를 검색하기 때문에 pointer 만을 넘기는 개선된 알고리즘이 더 나은 성능을 보인다. 하지만 5-단계 우선순위의 환경에서 프로세스의 수가 증가하더라도 성능이 더 나아지는 것을 그림 그림 49에서는 볼 수가 없다. 그 이유는 resume 된 프로세스가 MPS 의 자료 구조에 들어갈 때 자료 구조의 끝에 붙이는 작업이 개선되지 않았을 때의 알고리즘보다 복잡하기 때문이다. 결국 전체적으로 거의 일정한 성능 향상만을 보인다. 개선된 알고리즘이 더 나은 효과를 보이는 부분은 선점 지연 시간 (그림 50)으로 우선순위의 환경에 거의 무관하게 일정한 성능을 보여주고 있어서 예측 가능성에 많은 도움을 준다.

다. 다중 우선순위 환경을 지원하는 변경된 Library function 의 성능

Function	time(micro second)
ProcRun	7.5
ProcWait	11.5
SemWait	15.5
SemSignal	13.5
ChanIn/OutInt	9.0

표 2 INMOS-C compiler 의 성능: 실행 프로세스의 갯수는 2 개이고 모두 HIGH priority 로 수행시켰을 때

function	time(micro second)
ProcRun	51.00
ProcWait	61.18
SemWait	37.87
SemSignal	43.73
ChanIn/OutInt	72.35

표 3 MPS 의 성능: 실행 프로세스 수는 2 개이고 모두 같은 우선순위로 수행시켰을 때

응용 프로세스에 다중 우선순위를 지원하기 위해 MPS 는 library function 을 제공한다. 이들 function 은 그림 36에서 프로세스의 상태 변화를 일으키는 function 들이다. 이들 함수는 하드웨어에서 제공하는 자료 구조를 사용하지 않고 자체의 자료 구조를 사용한다.

procedure	time(micro second)
notify event	9
case statement	4
function call	5
s_preempt	19
deschedule	22
insertSQueue	15
total	74

표 4 ChanIn()을 구성하는 프로시저의 각각의 수행 시간

procedure	time(micro second)
s_preempt	19
queue operation	6.5
deschedule	22
total	46

표 5 SemWait()를 구성하는 프로시저의 각각의 수행 시간

표 2와 표 3은 INMOS-C Library function 의 성능과 다중 우선순위를 지원하는 MPS Library function 의 성능을 각각 나타내고 있다. 전체적으로 MPS 가 INMOS-C Library function 보다 5 배 이상의 실행 시간을 가진다. SemWait(), SemSignal()은 하드웨어적으로 지원되지않는 함수인데도 MPS 의 함수보다 3 배에 가까운 성능을 보인다. 그 원인은 MPS 가 scheduler 를 부르기 위해 통신을 사용하고 LOWQ 를 조작하는 등 부가적인 overhead 가 존재하기 때문이다. 표 4와 표 5는 Multi-Prioirty Scheduler 를 지원하는 Library function 의 전체 실행 시간을 부분적으로 나눈 결과를

보여주고 있다.

실제로 MPS 에서 semaphore 자료 구조를 다루는 부분의 실행 시간은 8 μsec 으로 INMOS-C semaphore function 의 전체 실행 시간인 15.5μsec 보다 훨씬 적지만 프로세스가 블록 되어 저장되는 시간과 MPS 의 자료 구조 내의 프로세스가 LOWQ 로 이동하는 시간 등이 부과되어 결과적으로 3 배에 가까운 실행 시간을 가지게 된다. 하드웨어 스케줄러의 문맥 전환 시간이 1μsec 이내이고 MPS 의 문맥 전환 시간이 평균 15.4μsec 이므로 실행 시간의 차를 줄이기는 힘들 것으로 보인다. 따라서 INMOS-C Library function 과 Multi-Priority Scheduler 를 지원하는 Library function 의 성능차는 MPS 의 overhead 가 주 원인으로 작용한다.

4 절 결론 및 향후 연구 방향

본 연구에서는 공장 자동화에 필요한 실시간 시스템을 구성하기 위해 병렬 컴퓨터 TiME 을 사용했다. TiME 은 각 프로세서가 담당하는 작업이 다르므로 모든 프로세서가 요구하는 함수를 커널 내부에 모두 넣게 된다면 커널의 크기가 증가해서 각 프로세서 당 많은 메모리를 요구한다. 본 연구에서는 마이크로 커널을 TiME 상에서 구현함으로써 커널의 크기를 축소하고 각 프로세서에서 필요로 하는 함수를 라이브러리로써 제공한다. 마이크로 커널에는 메모리 관리, 통신 관리, 프로세스 관리, 프로세스 스케줄링 등의 기능이 포함된다.

다중 우선순위 스케줄러는 실시간 시스템의 구축을 위해 기본적으로 요구되는 조건이나 TiME 의 프로세서인 transputer 에서는 제공되지 않으므로 소프트웨어적으로 그 기능을 구현하였다. 본 연구에서 제안한 다중 우선순위 스케줄러는 우선순위 큐의 상태가 변하는 사건이 발생했을 때만 호출되므로 주기적으로 호출되는 프로세스보다 호출 횟수가 적기 때문에 전체 실행 시간에서 스케줄러가 차지 하는 비율이 줄어든다. 또한 스케줄러가 호출된 경우라도 하드웨어 프로세스 큐를 다루는 경우

가 적기 때문에 호출 횟수 당 오버헤드도 줄어든다. 더욱이 모든 프로세스가 같은 우선순위를 가진 경우에는 스케줄러의 오버헤드는 무시할 수 있을 정도로 작다. 실험 결과 제안된 스케줄러의 오버헤드는 약 15.4 μsec 로 [shea92]의 알고리즘의 오버헤드인 53 μsec 보다 적게 나타난다.

스케줄러의 오버헤드가 15.4 μsec 로 나타났다고 해도 라이브러리 함수의 성능 저하에 많은 영향을 미친다. 특히 선점이 발생하는 경우에 많은 오버헤드가 발생하는데 이 오버헤드들을 줄이기 위해 많은 연구가 필요할 것으로 본다.

참고문헌

[ckmp90] O.~Caprani, J.E. Kristensen, C.~Mork, and H.B. Pedersen. Implementation of real-time scheduling algorithms in a transputer environment. In *Proceedings of the 13th Occam User Group Technical Meeting*, pages 186–197, September 1990.

[cheung95] M.H. Cheung, K.M. Shea, and Francis~C.M. Lau. A technique for process preemption in the transputer. *Microprocessors and Microsystems*, 19(1), February 1995.

[choe95] Tae-Young Choe, Chan-Ik Park, Chan~Mo Park, and Byung-Seop Kim. A multi-priority scheduler for transputer-based real-time systems. In *Proceedings of World Transputer Congress'95, Harrogate, UK*, September 1990.

[trans_inst] INMOS. *Transputer Instruction Set*, 1988.

[occam89] INMOS. *OCCAM2 toolset user manual*, 1989.

[ansi_ref] INMOS. *ANSI C toolset reference manual*, 1990.

[ansi_user] INMOS. *ANSI C toolset user manual*, 1990.

[liu73] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 1(1), January 1973.

[bodhisattwa93] Bodhisattwa Mukherjee, Karsten Schwan, and Kwashik Ghosh. A survey of real-time operating systems - preliminary draft. Technical report, College of Computing, Georgia Institute of Technology, 1993.

[ploeg94] E.~Ploeg, J.P.E. Sunter, A.W.P. Bakkers, and H.W. Roebbers. Dedicated multi-priority scheduling. In *Proceedings of the 17th World OCCAM and Transputer User Group Technical Meeting, Bristol,U.K.*, April 1994.

[shea92] K.M. Shea, M.H. Cheung, and F.C.M. Lau. *An Efficient Multi-Priority Scheduler*

for the Transputer, pages 139--153. IOS Press, 1992.

[swb90] J.P.E. Sunter, K.C.J. Wijbrans, and A.W.P Bakkers. Cooperative priority scheduling in occam. In *Proceedings of the 13th Occam User Group Technical Meeting*, September 1990.

[we90] P.H. Welch. Multi-priority scheduling for transputer-based real-time control. In *Proceedings of the 13th Occam User Group Technical Meeting*, September 1990.

제 2 차년도
연차 보고서

RTS 기술개발

RTS Technology Development

미시적 시뮬레이터 구축 기술 개발

연구기관
시스템공학연구소

과 학 기 술 처

여 백

제 4 장 미시적 실시간 시뮬레이터 구축 기술 개발

1 절 서론

미시적 실시간 시뮬레이션은 실시간 시스템을 구성하는 프로세서들의 기능적 행위(Functional Behaviors)와 시간적 행위(Temporal Behaviors)를 정밀하게 모형화하고 이를 고성능의 컴퓨터를 이용하여 실시간으로 실행하여 봄으로써 실시간 시스템의 구성 요소간의 유기적인 관련성과 영향을 분석 평가하는 기술이다. 실시간 시스템의 정밀한 모델링을 위하여 실시간 시스템을 구성하는 구성 요소를 모델링할 때 모델의 기능적 행위와 시간적 행위를 상세히 기술할 수 있는 기법이 필요하다.

본 연구에서는 1 차년도에 개발한 실시간 객체지향 모델링 방법을 이용하여 압연공정 AGC 의 시뮬레이션 모델을 구축하고 이를 구현하였다. 개발된 AGC 시뮬레이터는 AGC 개발과정의 설계단계의 각종 데이터를 시험할 수 있는 설계지원 도구로서의 역할을 할 수 있으며 Controller 설계의 상위 수준의 모델을 압연공정 시뮬레이션 환경에서 수행하여 모델의 정확성 및 설계변수의 시험을 위한 Validation tool 로서도 활용할 수 있다. 또한 압연공정 및 AGC 운영 관계자에게는 교육 훈련 도구로서도 활용할 수 있을 것이다.

시뮬레이션 모델의 설계 방법은 RTO.k 실시간 모델링 방법을 이용하였고 시뮬레이션 코드는 C++DL 을 구현 언어로 이용하였다. 시뮬레이션 엔진으로서는 Dream Kernel 을 이용하였으며 LAN 상 각각의 프로세서에 환경 시뮬레이터, Controller 및 Graphic Node 가 분산된 실시간 분산 시뮬레이션 환경을 구현하였다.

또한 실시간 시뮬레이터를 실행하기 위한 LAN 용 실행지원 장치를 개발하였고 시뮬레이션 코드를 생성하기 위한 실시간 객체지향 언어도 아울러 개발하였다.

2 절 압연공정 AGC 모델링

압연공정은 제철, 제강과 함께 철강 제조 공정을 구성하는 주요 공정이며 미국, 유럽등 선진국에서는 1960 년대 부터 컴퓨터를 이용한 자동제어 시스템을 적극적으로 도입함으로써 압연공정의 생산성 향상, 원료의 손실 절감, 품질 향상등을 효과적으로 이루어 왔다. 철강 제품의 90%는 가열 및 압연과정에서 결정되며 그 중에서도 압연기술은 특히 중요하다. 압연공정의 효과적인 제어를 위해서는 공정 모델링, 제어이론 및 컴퓨터의 사용이 필수적이며 센서 및 액츄에이터의 성능도 제품의 품질에 직접적으로 영향을 미치는 요인으로 작용한다.

AGC(Automatic Gauge Gontrol)는 자동 판 두께 제어방법이며 BISRA 방법을 비롯한 여러가지 방법이 이용되고 있다. 그러나 대부분의 AGC 시스템이 압연공정의 수학적 모델링에 기초한 제어 알고리즘을 이용하고 있기 때문에 모델링 오차의 존재시에는 응답 특성이 크게 변화될 뿐만 아니라 전체 제어 시스템에도 영향을 미치게 된다.

본 연구에서 개발한 AGC 시뮬레이터는 단일 스탠드 밀(Single Stand Mill)을 그 대상으로 하고 있다. 단일 스탠드 AGC 시스템은 크게 두 부분으로 나누어지는데 하나는 롤 스탠드, Pay-off reel, Tension reel 및 각종 액츄에이터로 이루어지는 압연공정이고 다른 하나는 압연공정을 효과적으로 동작시키기 위한 제어 시스템이다.

1. 압연공정 및 AGC의 개요

그림 1은 단일 스탠드 밀을 위한 압연 공정의 개요도이다. 여기서 전체 시스템은 압연공정부와 AGC 시스템부로 나누어지는데, 압연 공정부는 압연현상이 발생하는 롤 스탠드부, 소재의 풀림 및 감김이 일어나는 Pay-off reel부와 Tension reel부로 이루어지고, AGC부는 압연 스케쥴 및 압연공정부의 궤환 신호를 이용하여 원하는 두께의 소재를 얻을 수 있도록 각종 제어신호를 생성하는 제어시스템으로 구성된다.

압연공정부에 대한 입력신호에는 작업 롤 간격 기준신호(Sp), 롤 속도 기준신호(V_{rp}), Pay-off reel 구동모터의 기준 전류신호(i_p) 및 Tension reel 구동모터의 기준 전류신호(i_t)등이 있으며, 출력신호에는 입출력 소재두께(H, h), 입출측 장력(T_i, T_o), 압하력(P) 및 롤 간격 (S)등이 있다. AGC부의 입력신호에는 원하는 소재의 출측두께, 입측장력 기준신호, 출측장력 기준신호 및 압연공정부로부터의 여러가지 궤환신호들이 있으며, 출력신호들은 압연공정부의 입력신호들과 동일하다.

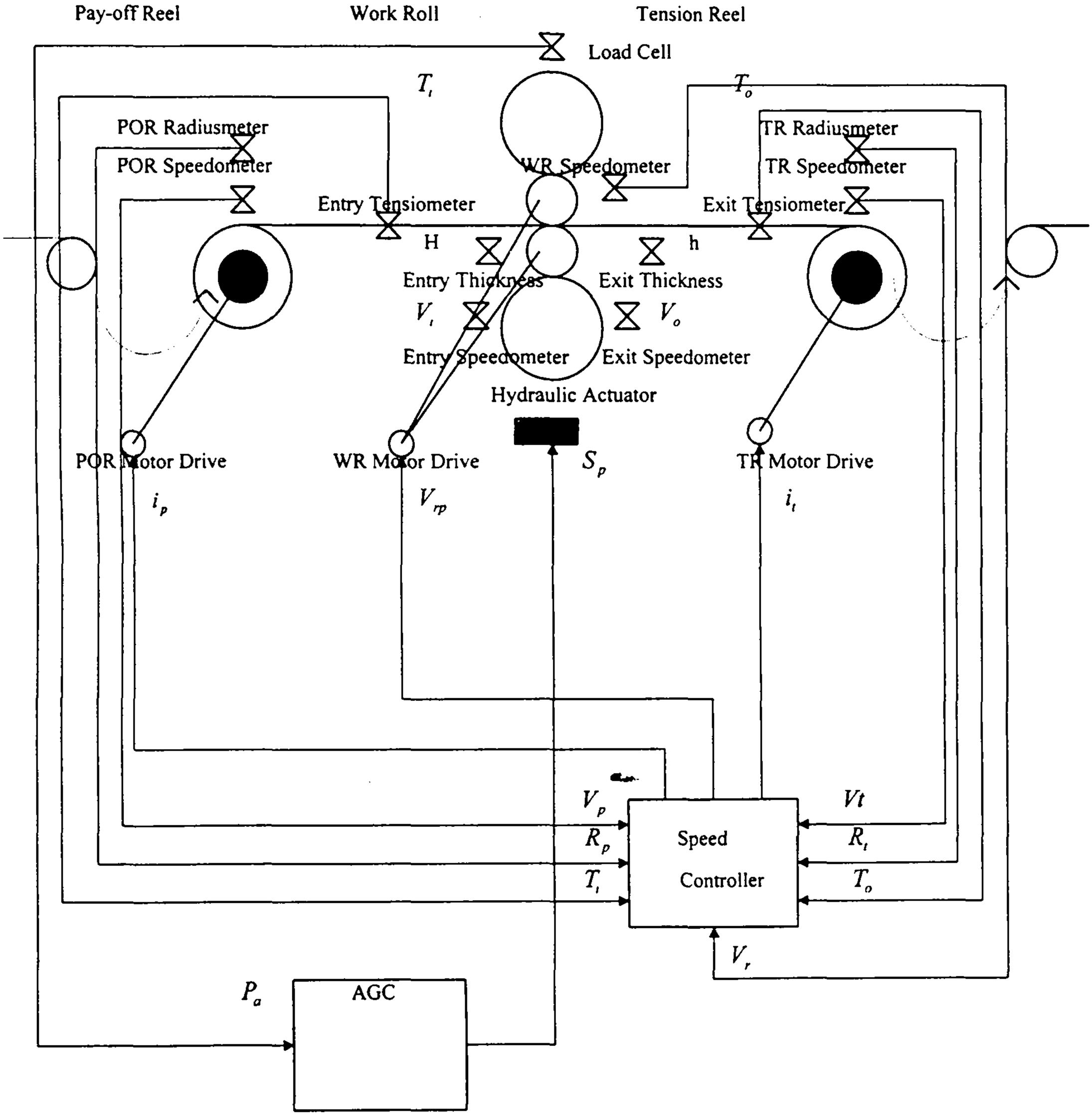


그림 1: 단일 스탠드 밀의 개요도

2. AGC 시스템의 제어방법

가. 장력 제어

압연공정에서 압연하중은 장력에 의해 영향을 받는다. 장력은 각 reel 과 스탠드 roll 의 속도차이와 소재두께에 의해서 변하게 되는데, reel 의 속도는 소재가 감겨있는 정도에 따라 즉 reel 의 반경에 따라 속도가 증가하거나 감소하게 된다. 장력은 압연하중에 영향을 미치고 있고 이로 인해 개루프 특성 및 압연공정에서의 제어에 의한 효과도 또한 줄어들게 된다. 따라서 장력을 일정하게 유지하는 작업이 필요하며, 이를 성취하기 위해 장력 제어기를 설치하여야 한다.

나. 압연원리

압연 소재의 소성특성과 압연기의 탄성특성을 간략하게 나타내면 그림 2 와 같다. 소재 두께에 따른 특성곡선 중 f 는 압연기의 탄성곡선이고 g 는 소재의 소성곡성이다. f'은 f 에서 롤 간격이 변화되었을 때 이동된 탄성곡선이고, g'은 g 에서 입측 소재두께가 변화되었을 때 이동된 소성곡선이다. 압연점은 이들 곡선의 교점에서 결정되며 그 교점에서의 소재두께가 출측 두께로 결정된다. 따라서 어떤 요인에 의해 이들 곡선에 변화가 생긴다면 출측 두께는 변하게 된다.

그림 2 의 압연특성 그래프에서 볼 수 있듯이 입측 소재두께 변동이 있을 때 소성곡선 g 는 g'로 이동된다. 이에 따라 압연점은 변동되어 A 에서 B 로 이동되고 압연하중과 출측 소재두께도 각각 ΔP, Δh 만큼씩 이동된다. 따라서 원하는 두께의 소재를 얻을 수 없다. 이 출측 두께 변화 Δh 를 제거하기 위해 롤 간격을 ΔS 만큼 이동시켜 압연점을 B 에서 C 로 이동시켜 원하는 출측 소재두께 h_0 를 얻는 것이 본 제어의 목적이다.

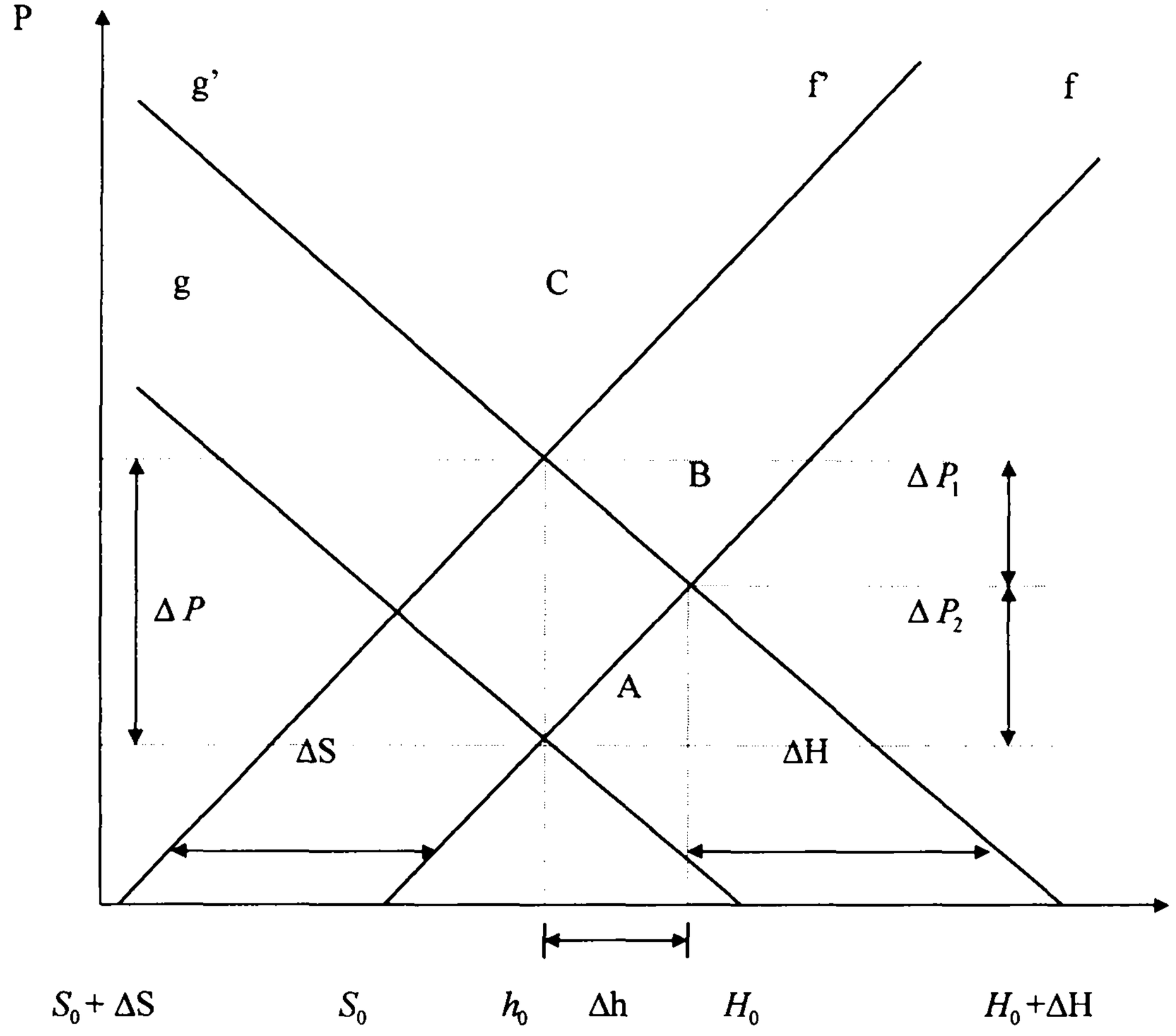


그림 2: 압연 특성 곡선

다. 두께 제어

단일 스탠드의 압연 제어는 압연기의 탄성특성과 소재의 소성특성에 기반을 두고 있는 압연원리를 이용하고 제어구조를 사용함으로써 다음과 같은 네 가지 방법이 사용되고 있다. 첫번째는 외란과 롤 간격 변화에 의한 압연하중 변화를 제어량에 이용하는 BISRA AGC 방법이다. 입측 소재두께등의 변동이나 작업 롤 간격 등이 변화하였을 때 압연하중이 변화되고, 압연원리에 따라 압연점이 이동하게 되는데, load cell 로 측정되는 압연하중 신호를 궤환하고 이를 통해 적당한 제어입력을 생성함으로써 롤 간격을 동작시키는 원리로 되어 있다. 두번째는 소재두께 측정기의 입측 소재두께 변동 신호를 이용하는 Feedforward AGC 방법이다. 이 방법에서는 외란이 되는 입측 소재두께 변동을 롤 스탠드 앞에서 측정한 후 이 값을 이용하여 적절한 압하조작을 계산된 지연시간 이후에 인가하는 원리로 되어있다. 세번째는 출측 소재두께를 측정하여 궤환하여 이용하는 Model-based feedback AGC, 그리고 네번째는 입출측 소재두께와 소재속도를 이용하는 Mass flow AGC 가 있다.

3. 단일 스탠드 압연 공정의 모델링

압연 공정 제어 모델은 상당히 복잡한 비선형 방정식의 형태를 가지고 있으며 적지 않은 연산 시간을 요구한다. 그 결과 이들을 실시간 시뮬레이션이나 제어기 설계에 사용하기에는 어려움이 따른다. 여기서는 보다 간략화된, 그러나 실제 압연 현상을 상당히 근사하게 모델링할 수 있는 모델을 소개한다.

가. 압하력식(소성 방정식)

주어진 소재의 압연시 발생하는 압하력은 소재의 두께와 폭 및 변형 저항,

롤 스탠드 입측 및 출측에서의 장력과 밀접한 관계가 있으며 이를 롤 스탠드 구성 요소와 함께 나타내면 다음과 같다.

$$P_a = Wk_mQ_pP_t\sqrt{R(H-h)}$$

여기서, P_a : 발생 압하력　　P_t : 장력 영향항

W : 소재 폭　　k_m : 변형저항

Q_p : 압하력 함수　　R : 작업롤 반경

H : 입측 소재두께　　h : 출측 소재두께

이때, $Q_p = 1.08 + (1.79\mu r\sqrt{1-r}\sqrt{\frac{R}{h}}) - 1.02r$

여기서, μ : 소재와 작업롤 사이의 마찰계수

$r = \frac{H-h}{H}$: 압하율

$$P_t = [1 - \frac{1}{k_m}(m_1t_1 + m_2t_o)]$$

여기서, t_1 : 입측 단위면적당 장력

t_o : 출측 단위면적당 장력

m_1 : 입측 장력 영향 계수

m_2 : 출측 장력 영향 계수

나. 압하력 지연식

실제 발생한 압하력을 load cell 에서 측정하기까지는 시간 지연이 존재하며 다음과 같이 나타낼 수 있다.

$$\frac{dP}{dt} = -\frac{dP}{dt} + \frac{dP_a}{dt}$$

여기서, P : load cell 에서 측정한 압하력

다. Gaugemeter 식 (탄성 방정식)

압연시 작업롤 간격과 출측 소재두께 및 이때 발생하는 압하력과의 관계는 다음과 같이 나타낼 수 있다.

$$h = S + \frac{P_a}{M}$$

여기서, S : 작업롤 간격

M : 압연기의 탄성계수

라. 유압식 액츄에이터 지연식

압연시 작업롤 간격을 변화시켜 주기 위한 액츄에이터의 제어 입력과 이것이 실제 작업롤 간격을 변화시키기까지는 시간 지연이 존재하며 다음과 같이 나타낼 수 있다.

$$\frac{dS}{dt} = -\frac{dS}{dt} + \frac{dS_p}{dt}$$

여기서, S_p : 작업롤 간격 지령치

마. 선진율식

선진율은 소재가 롤 스탠드에서 압연될 때 작업롤의 선속도와 소재의 선속도가 일치하지 않게 되는 현상을 모델링한 것으로 다음과 같이 정의된다.

$$f = \frac{V_o - V_r}{V_r}$$

여기서, f: 선진율

V_o : 출측 소재의 선속도

V_r : 작업롤의 선속도

출측 소재의 선속도를 직접 측정할 수 없기 때문에 위의 식을 이용하여 setup calculation 을 수행할 수 없으므로 다음의 근사식으로 선진율을 계산한다.

$$f = [\frac{\sqrt{r}}{2} - \frac{1}{\mu}\sqrt{\frac{h}{R}}(\frac{2r}{2-r} - \frac{t_o - t_i}{km})\phi]^2$$

여기서, km : 평균 변형저항

ϕ : 수정 계수

바. Mass flow 식

입측 소재의 선속도와 출측 소재의 선속도와의 관계는 다음과 같이 나타낼 수 있다.

$$V_i = \frac{h}{H} V_o$$

여기서, V_i : 입측 소재의 선속도

사. 작업롤 속도제어장치 지연식

작업롤 속도를 변화시켜 주기 위한 작업롤 속도 지령치와 이에 따라 작업롤 속도가 변하기까지는 시간 지연이 존재하며 다음과 같이 나타낼 수 있다.

$$\frac{dV_r}{dt} = -\frac{dV_r}{dt} + \frac{dV_{rp}}{dt}$$

여기서, V_{rp} : 작업롤 속도 지령치

아. Pay-off reel 속도식

Pay-off reel 속도는 reel의 반경 및 입측 장력이 변함에 따라 영향을 받으며 다음과 같이 나타낼 수 있다.

$$\frac{dV_p}{dt} = \frac{K_p R_p}{J_p N_p} i_p + \frac{R_p^2}{J_p N_p^2} T_i$$

여기서, V_p : Pay-off reel 선속도

K_p : Pay-off reel 전동기 토크 상수

R_p : Pay-off reel 반경

N_p : Pay-off reel 기어비

J_p : Pay-off reel 관성 모멘트

i_p : Pay-off reel 구동 전류

T_t : 입측 장력

이때, $J_p = J_{mp} + \frac{\pi}{2}\rho W(R_p^4 - R_{cp}^4) / N_p^2$

여기서, J_{mp} : Pay-off reel motor 관성 모멘트

R_{cp} : Mandrel 반경

ρ : 소재의 밀도

자. Pay-off reel 반경식

Pay-off reel에 소재가 감겨 있을 때 소재를 포함한 반경은 소재의 두께 및 reel의 회전속도 등에 따라 변하게 되는데 그 관계식은 다음과 같다.

$$\frac{dR_p}{dt} = -\frac{H}{2\pi}\frac{V_p}{R_p}$$

차. Tension reel 속도식

Tension reel 속도는 reel의 반경 및 입측 장력이 변함에 따라 영향을 받으며 다음과 같이 나타낼 수 있다.

$$\frac{dV_t}{dt} = \frac{K_t R_t}{J_t N_t} i_t + \frac{R_t^2}{J_t N_t^2} T_o$$

여기서, V_t : Tension reel 선속도

K_t : Tension reel 전동기 토크 상수

R_t : Tension reel 반경

N_t : Tension reel 기어비

J_t : Tension reel 관성 모멘트

i_t : Tension reel 구동 전류

T_o : 입측 장력

이때, $J_t = J_{mt} + \frac{\pi}{2}\rho W(R_t^4 - R_{ct}^4) / N_t^2$

여기서, J_{mt} : Pay-off reel motor 관성 모멘트

R_{ct} : Mandrel 반경

ρ : 소재의 밀도

카. Tension reel 반경식

Tension reel에 소재가 감겨 있을 때 소재를 포함한 반경은 소재의 두께 및 reel의 회전속도 등에 따라 변하게 되는데 그 관계식은 다음과 같다.

$$\frac{dR_t}{dt} = -\frac{H}{2\pi}\frac{V_t}{R_t}$$

타. 입측 장력식

입측 장력의 발생은 입측 소재의 선속도와 Pay-off reel의 선속도가 일치하지 않음으로써 발생하며 다음과 같이 나타낼 수 있다.

$$\frac{dT_i}{dt} = \frac{EHW}{L_p}(V_i - V_p)$$

여기서, E : 소재의 Young 계수

L_p : Pay-off reel 에서 롤 스탠드까지의 거리

파. 출측 장력식

출측 장력의 발생은 출측 소재의 선속도와 Tension reel 의 선속도가 일치하지 않음으로써 발생하며 다음과 같이 나타낼 수 있다.

$$\frac{dT_o}{dt} = \frac{EHW}{L_t}(V_t - V_o)$$

여기서, E : 소재의 Young 계수

L_t : Tension reel 에서 롤 스탠드까지의 거리

4. 시뮬레이션 모델링 접근방법

가. RTO.k 모델링 방법

본 연구의 시뮬레이션 모델은 RTO.k 모델링 방법을 채택하고 있다. RTO.k 모델은 전통적인 객체모델을 확장하여 실시간 분산 시스템을 효과적으로 표현하고 있다.

RTO.k 모델이 전통적인 모델과 특징적으로 구분되는 두가지 다음과 같다.

(1) SpM(spontaneous method)로 불리는 시간 구동 객체 메소드와 SvM(service

method)로 불리는 메시지 구동 메소드간의 명확한 분리

(2) SvM 에 대하여 SpM 에게 항상 수행 우선순위를 부여하는 기본 동시성 제한조건(basic concurrency constraint)

RTO.k 객체모델은 실시간에 의해 구동되는 operation 을 하나의 메소드로 집합시키고 있으므로 객체내의 실시간 태스크들의 복잡한 시간 행위(temporal behavior)를 효율적으로 기술할 수 있고 설계 오류를 비교적 쉽게 찾아낼 수 있게 해준다.

RTO.k 객체를 기반으로 하는 일관성 있는 구조적 접근방법하에서는 제어 시스템 설계와 환경 시뮬레이터의 조합은 실시간 객체들의 네트워크 형태를 갖는다.

RTO.k 객체 모델의 기본 구조는 그림 3 에 보여 진다.

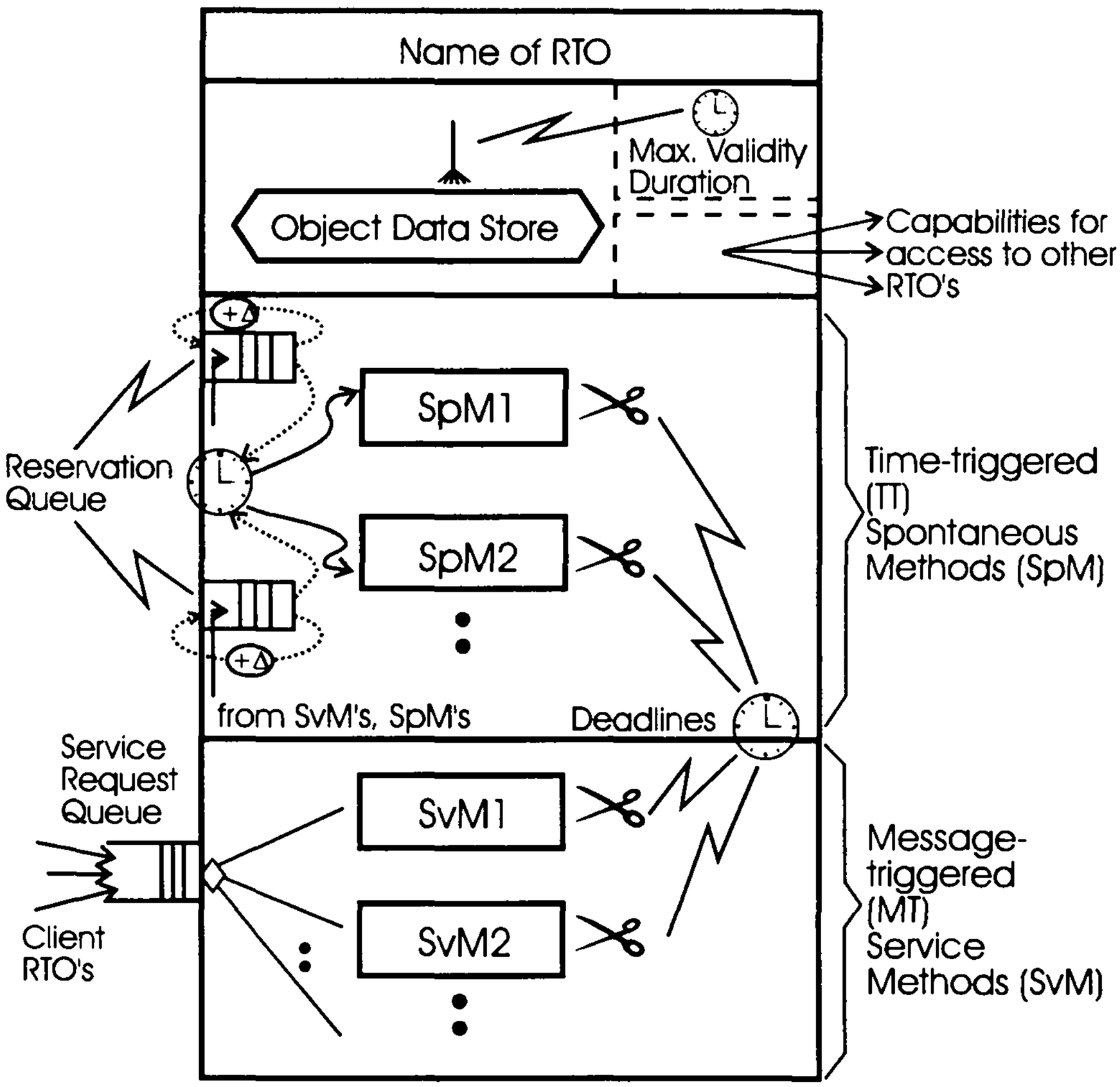


그림 3 : RTO.k 객체 모델의 구조

나. 실시간 분산 시뮬레이션

단일 스탠드 밀 시뮬레이션 모델은 기본적으로 3 개의 노드를 구성한다. 제어 시스템이 하나의 노드가 되며 환경 시뮬레이터가 또 하나의 노드가 되고 그리고 그래픽 노드가 다른 하나의 노드가 된다. 이들 3 개의 노드는 각각의 프로세서(processor)에 대응되며 LAN 을 통하여 노드간의 통신을 한다.

그림 4 에 실시간 분산 시뮬레이션의 구조를 나타내고 있다.

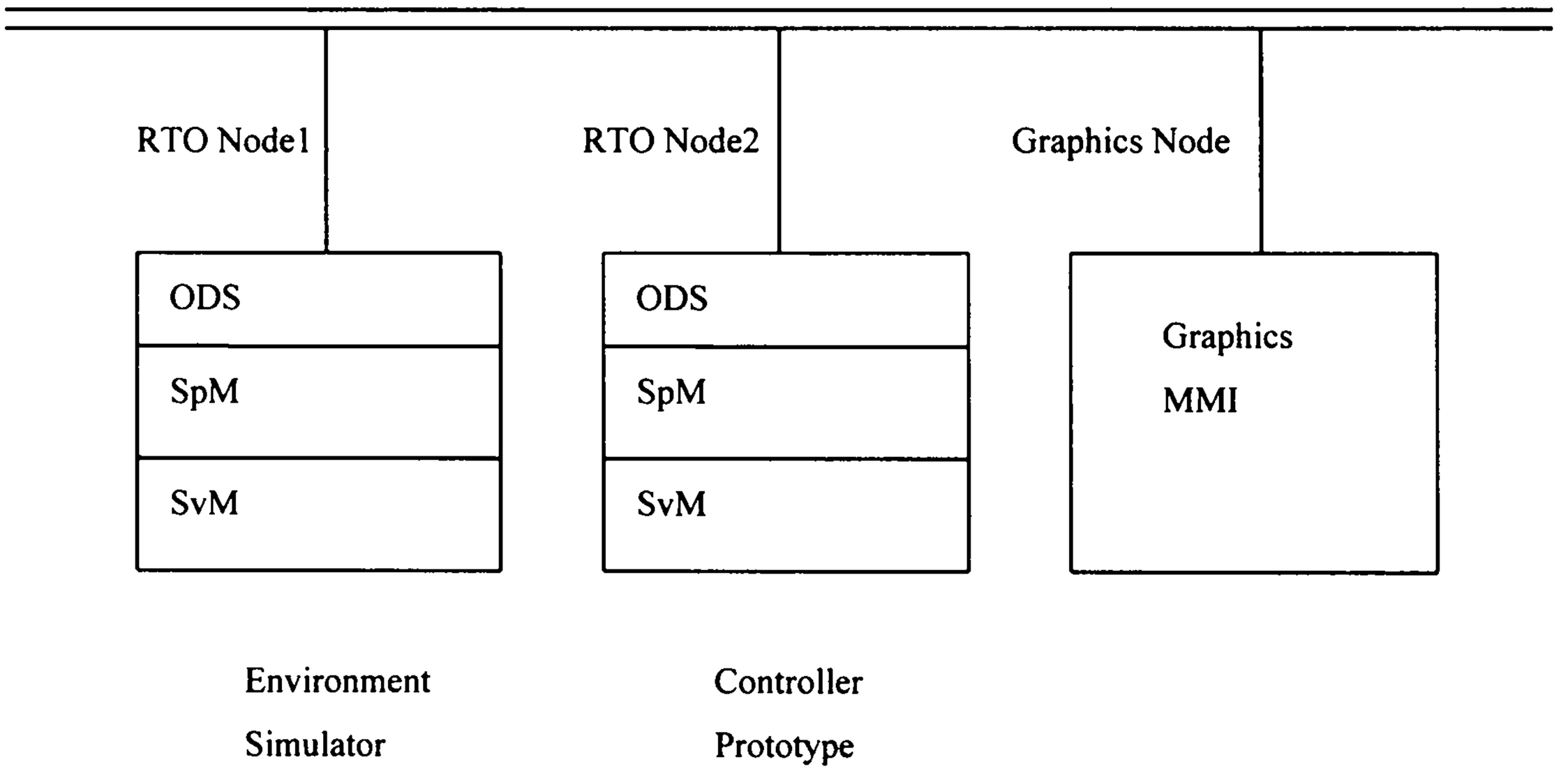


그림 4: 실시간 분산 시뮬레이션 구조

3 절 시뮬레이션 모델 설계

단일 스탠드 밀 압연공정은 시뮬레이션 환경인 압연공정과 AGC 제어기로 구분할 수 있고 다시 압연공정은 Pay-off reel, Work roll 및 Tension reel 로 구분할 수 있으며 AGC 제어기는 속도 제어기와 AGC 로 구분할 수 있다. 즉 전체 시스템은 5 개의 객체로 나누어 질 수 있다. 이들 객체들은 모두 실시간 객체로서 각각을 RTO 로 대응시킬 수 있고 각각의 RTO 는 각각 고유한 주기로 구동되는 SpM 과 외부로 부터의 메시지에 의해 구동되는 SvM 을 갖는다.

1. 모델링 개요

그림 5 에 압연공정 AGC 모델링의 개요도를 나타내었다.

Pay-off reel 객체는 POR Radiusmeter, POR Speedometer 및 Entry Tensiometer 등의 3 개의 센서로 부터 POR 상태정보를 수집하며 이들을 주기적으로 Speed Controller 객체로 전송한다. Speed Controller 에서 계산한 전류신호는 POR Motor 가 받아서 적절한 속도를 생성하며 이를 통해 일정한 장력을 유지하게 된다.

Work roll 객체는 Load Cell, WR Speedometer, Entry Thickness sensor, Exit thickness sensor, Entry Speedometer 및 Exit Speedometer 를 통해 WR 상태정보를 수집하여 Speed Controller, AGC, POR 및 TR 객체등에 전송한다. Load Cell 로 부터는 Work roll 의 압하력을 읽어들이고 이를 AGC 가 받아서 적절한 Roll Gap 를 생성하여 Work roll 에 전송한다.

Tension reel 객체는 TR Radiusmeter, TR Speedometer 및 Exit Tensiometer 등의 3 개의 센서로 부터 TR 상태정보를 수집하며 이들을 주기적으로 Speed Controller 객체로 전송한다. Speed Controller 에서 계산한 전류신호를 TR Motor 가 받아서 적절한 속도를 생성하며 이를 통해 일정한 장력을 유지하게 된다.

Speed Controller 객체는 POR, WR, TR 객체들의 센서로 부터 메시지를 수집하여 각 객체의 소재 장력을 일정하게 유지하기 위한 계산을 통해 전류을 생성하며 이를 각각의 객체에 전송한다.

AGC 객체는 Work roll 의 Load Cell 로 부터 압하력 정보를 수신하여 Target Thickness 을 제공하기 위한 계산을 한다. 이 계산의 결과에 의해 Roll Gap 지령치를 Work roll 에 송신한다.

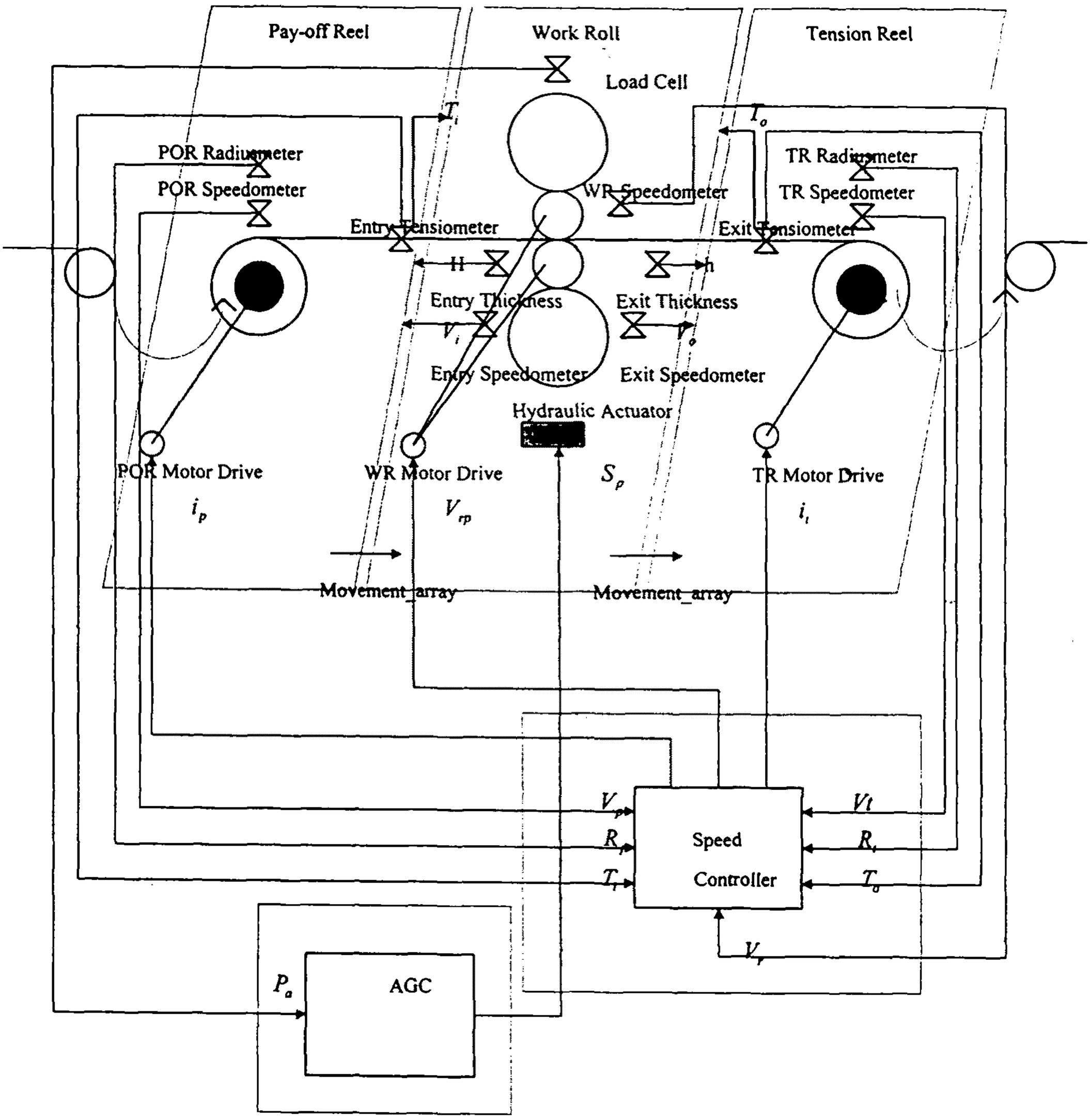


그림 5 : 압연공정 AGC 모델링 개요

2. 압연공정 RTO의 분할

RTO 모델링 방법은 우선 전체 시스템을 하나의 RTO로 정의하면서 시작한다. 대상공정을 거시적으로 보게 되면 Data space에는 Rolling Mill, AGC, Speed Controller 등의 3개의 Data가 존재하고 SpM은 각각의 Data가 자신의 상태를 주기적으로 update하는 것을 표현할 수 있고 SvM은 소재의 장입 및 배출로 기술된다. 기본적으로 이것이 가장 간단히 표현한 시스템의 개략 명세이다. 그림 6에 Top level의 RTO 명세가 나타나 있다.

	Rolling Process	
ODS	Rolling Mill AGC Speed Controller	Maximum Validity Duration ?? NextMill.LoadMaterial
SpM	"Update the states of the physical objects in the rolling process" Update the states of Rolling Mill (⊃call NextMill.LoadMaterial (Material)) Update the states of AGC Update the states of Speed Controller	
SvM	LoadMaterial (Material) Release Material (Material)	
Conditions		

그림 6 : 압연공정의 RTO

다음은 이 RTO를 분할한다. 이를 분할하게 되면 먼저 압연공정과 후속 압연공정으로 나타낼 수 있고, 압연공정은 Rolling mill과 두개의 제어기로 구분할 수 있다. 이를 RTO 분할도로 나타내면 그림 7과 같다.

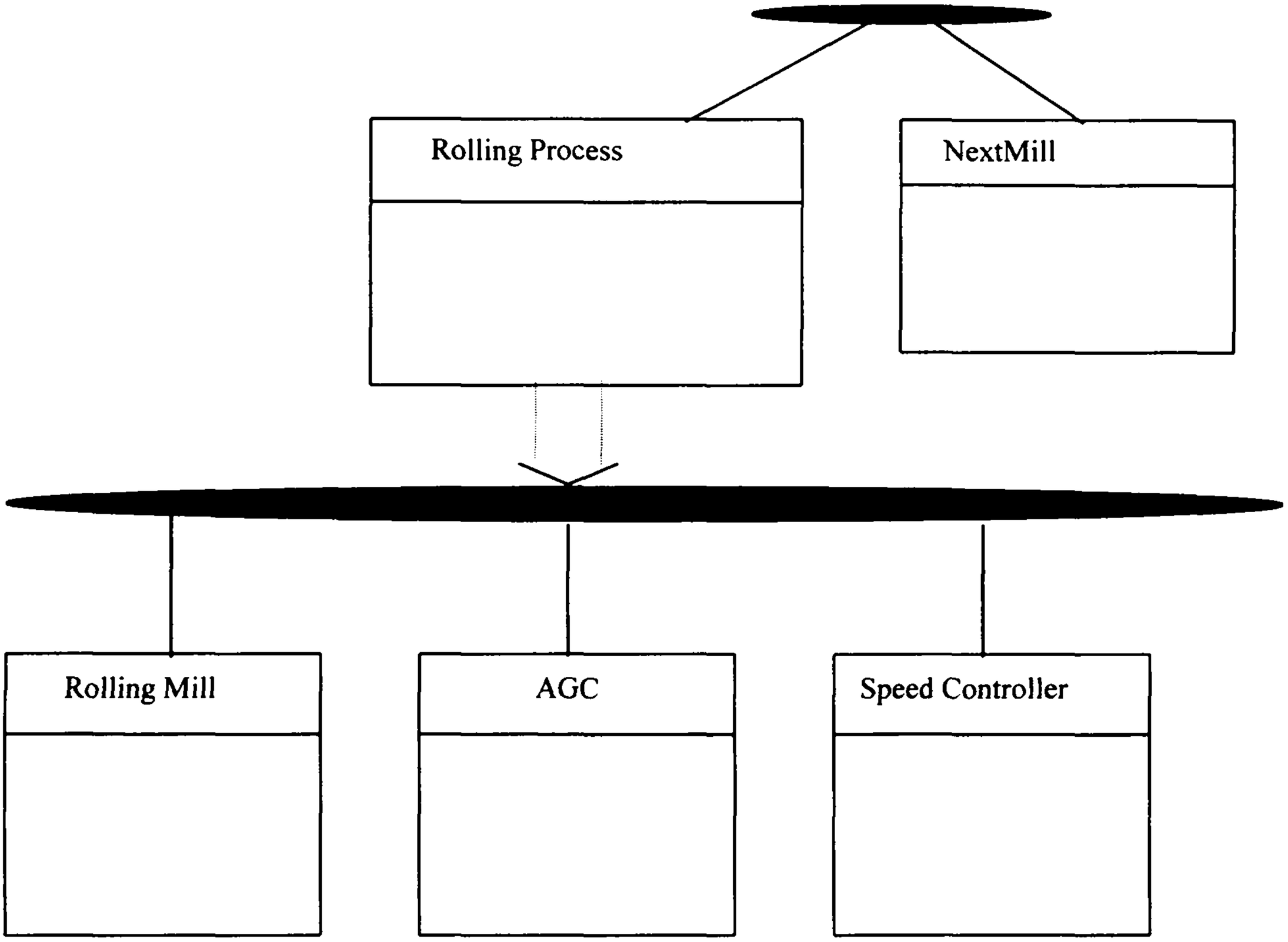
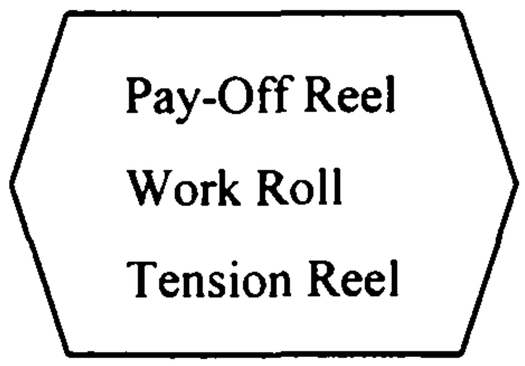


그림 7: 압연공정 RTO의 분할

압연공정 분할 네트워크 중 Rolling Mill 을 RTO 로 나타내면 그림 8 과 같다

<table>
<tr><td></td><td colspan="2">Rolling Mill</td></tr>
<tr><td>ODS</td><td>  </td><td>Maximum Validity Duration ??

- AGC
- Speed Controller
- NextMill</td></tr>
<tr><td>SpM</td><td colspan="2">Update the states of Pay-Off Reel
Update the states of Work Roll
Update the states of Tension Reel</td></tr>
<tr><td>SvM</td><td colspan="2">Receive data from AGC
Receive data from Speed Controller
Read sensor for AGC
Read Tensiometer
Load Material (Material)</td></tr>
<tr><td>Conditions</td><td colspan="2"></td></tr>
</table>

그림 8 : Rolling Mill 의 RTO

다시 전체 압연공정을 RTO 분할 네트워크로 나타내면 그림 9 와 같다.

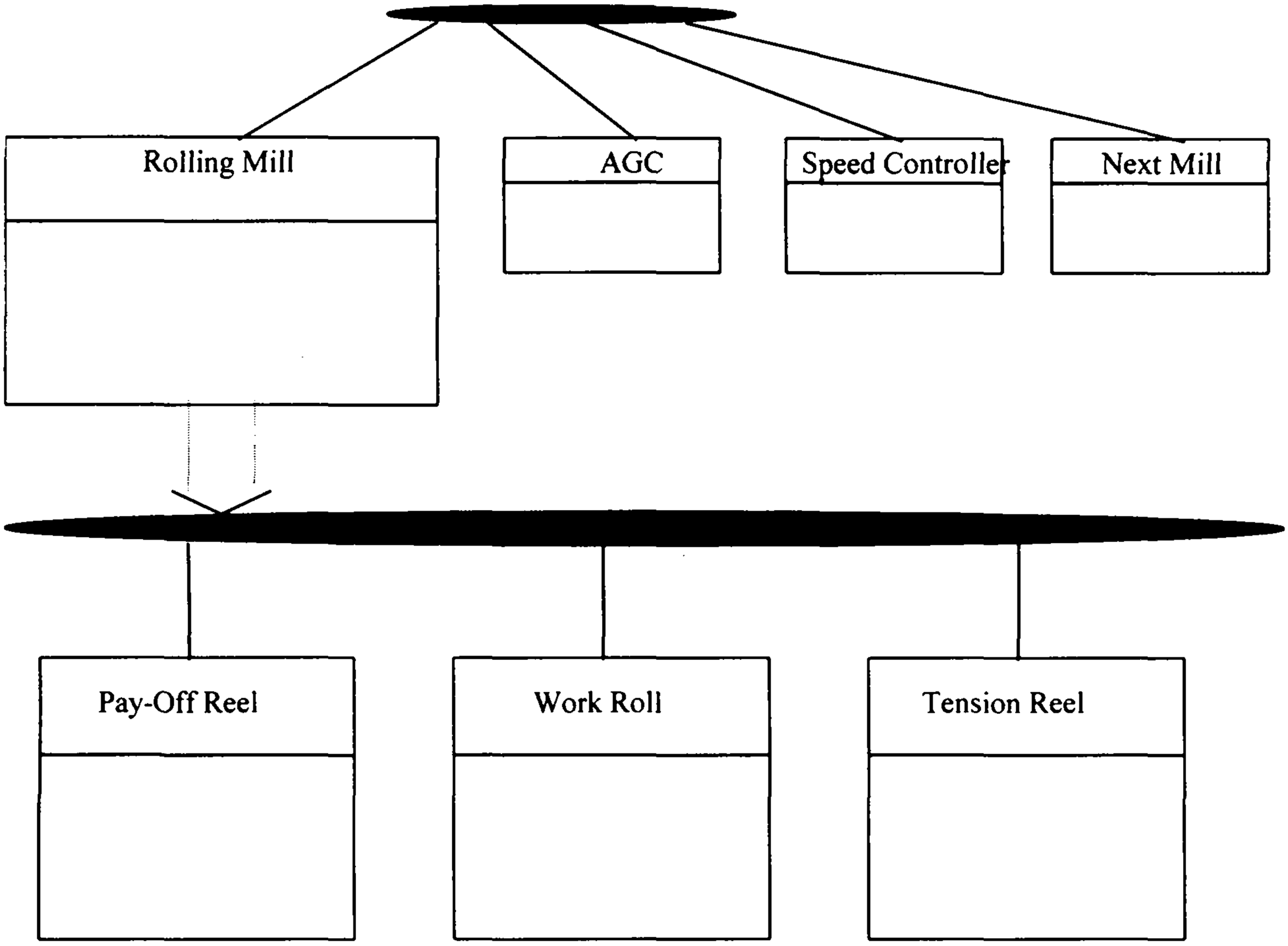


그림 9: 압연공정의 RTO 분할 네트워크

3. 설계 명세

압연공정 시뮬레이션 모델은 전체 시스템을 5 개의 RTO 로 분할 하고 있다. Pay-Off Reel, Work Roll, Tension Reel 의 3 개의 Rolling Mill 객체와 AGC, Speed Controller 의 2 개의 Controller 객체로 구성된다.

가. Pay-Off Reel

Pay-Off Reel RTO 는 SvM 에서, 소재를 장입하고 Speed Controller 로부터 전류를 받고 Work Roll 로 부터는 소재의 입측속도를 받으며 SpM 에서는 주기적으로 변화하는 POR 의 반경을 update 하고 소재의 입측 장력을 주기적으로 update 하며 Speed Controller 로 부터 받은 전류를 이용하여 POR 의 속도를 update 하여 POR 을 돌려준다. 또한 이동하는 소재의 위치를 감시하며 WR 로 주기적으로 소재를 넘겨준다. 그림 10 에 Pay-Off Reel 의 RTO 명세를 나타내었다.

	Pay-Off Reel
Use	- Speed Controller - Work Roll
ODS	POR : Radius (R_p) Distance to Work Roll (L_p) Velocity (V_p) Torque Constant (K_p) Gear Ratio (N_p) Inertia Moment (J_{mp}) Material : Width (W) Stiffness (k_m) Young Coefficient (E) Length (whole) (L) Thickness Array (for the materials in POR) Movement Array (for thicknesses in movement) Location of material (por_entryp) Entry Velocity (V_i) Entry Tension (T_i) Entry Thickness (H)
SpM	1. Update the radius of POR - Calculate the radius of POR at time T with the parameters at just past simulation tick

- Send R_p to Speed Controller via POR Radiusmeter

AC : every t msec

OS : $R_p[T] = R_p[T-t] - \frac{H[T-t]}{2\pi}\frac{V_p[T-t]}{R_p[T-t]}t$

2. Update the entry tension of material

- Calculate the entry tension at time T with the parameters at just past simulation tick
- Send T_i to Speed Controller and Work Roll via Tensiometer

AC : every t msec

OS : $T_i[T] = T_i[T-t] + \frac{EH[T-t]W}{L_p[T-t]}(V_i[T-t]-V_p[T-t])t$

3. Update the POR Velocity

- Calculate the POR Velocity at time T with the parameters at just past simulation tick

AC : every t msec

OS : $V_p[\mathrm{T}] = V_p[\mathrm{T\text{-}t}] + (\frac{K_p R_p[T-t]}{J_p[T]N_p}i_p + \frac{R_p[T-t]^2}{J_p[T]N_p^2}T_i[T-t])t$

+ ARt

A : Delay Coefficient

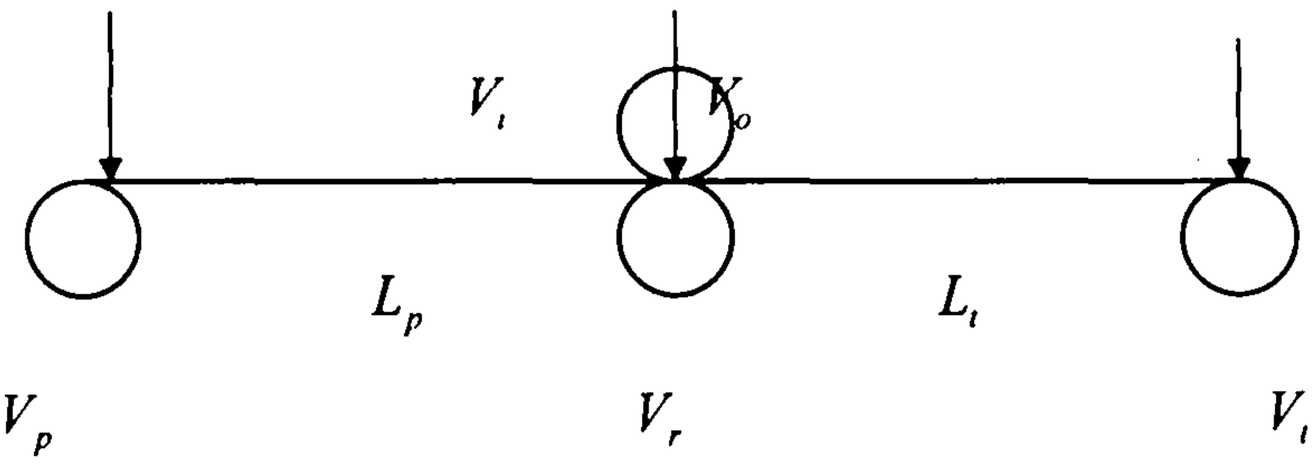
R : Random Number Generator (-1<R<1)

where

$$J_p[T] = J_{mp} + \frac{\pi}{2}\rho W(R_p[T-t]^4 - R_{cp}[T-t]^4)/N_p^2$$

4. Update the material array in POR

- Calculate the Material Position

<table>
<tr><td></td><td>- Copy the portion of POR array to Movement array
- Update H from the POR array movement
- Send Movement_array to WR_array in Work Roll
AC : every t msec
OS :
if por_entryp < L movement = V_p [T-t]* t
por_entryp = por_entryp + movement
if por_entryp >= L_p
- Move the portion of POR array thicknesses to
Movement_array by the por_entryp movement</td></tr>
<tr><td>SvM</td><td>1. Load Material
2. Load setup value file
3. Receive Current (i_p) for POR Motor Drive from Speed Controller
4. Receive Material Entry Velocity(V_t) from Work Roll</td></tr>
<tr><td>Conditions</td><td>por_entryp wr_entryp tr_entryp
 

V_p V_r V_t
- Material Thickness Array
POR_array (L) initialized by certain values of thicknesses
(The material to be rolled)</td></tr>
</table>

	WR_array (L) initialized by all zeros TR_array (L) initialized by all zeros (The finished material) Movement_array (x) ====> Move the thickness values from an array to the next array between the adjacent objects by moving the material arrays via Movement_array at every simulation tick

그림 10 : Pay-Off Reel 의 RTO 명세

나. Work Roll

Work Roll RTO 는 SvM 에서, Speed Controller 로부터 WR 속도 지령치를 받고 AGC 로 부터 Roll Gap 지령치를 받으며 POR 로 부터는 입측 장력과 소재를 전달 받으며 TR 로 부터는 출측장력을 받는다. SpM 에서는 주기적으로 변화하는 압하력을 update 하고 Speed Controller 로 부터 받은 속도 지령치를 이용하여 Work Roll 의 속도를 update 하여 WR 을 돌려준다. AGC 로 부터 받은 Roll Gap 지령치을 이용하여 Taget Thickness 를 획득하기 위한 Roll Gap 을 생성하며 소재의 입,출측 속도를 계산한다. 또한 이동하는 소재의 위치를 감시하며 생성된 Roll Gap 치를 이용하여 소재의 출측 두께를 계산, update 하며 update 된 소재를 TR 로 주기적으로 넘겨준다. 그림 11 에 Work Roll 의 RTO 명세를 나타내었다.

	Work Roll
Use	- Speed Controller - AGC - Pay-off Reel - Tension Reel
ODS	Work Roll : Roll Radius (R) Coefficient of Elasticity (M) Distance to Tension Reel(L_t) Roll Gap (S) Command Roll Gap (S_p) Roll Force (P_a) Velocity (V_r) Command Velocity (V_{rp}) Material : Width (W) Friction Coefficient (μ) Stiffness (k_m) Length (whole) Input Movement Array Output Movement Array Thickness Array (for the materials in Work Roll) Location of material (wr_entryp) Entry & Exit Velocity (V_t , V_o) Entry & Exit Thickness (H, h) Entry & Exit Tension (T_t , T_o)

SpM	1. Update the Roll Force - Calculate the Roll Force at the Work Roll caused by moving material - Send P_a to AGC via Load Cell AC : every t msec $P_a[T] = Wk_m Q_p[T] P_t[T] \sqrt{R(H[T-t] - h[T-t])}$ *where* $Q_p[T] = 1.08 + (1.79\mu r[T-t]\sqrt{1-r[T-t]}\sqrt{\frac{R[T-t]}{h[T-t]}}) - 1.02r[T-t]$ $r = \frac{H-h}{H}$: Load Ratio $P_t[T] = [1 - \frac{1}{k_m}(m_1 t_i[T-t] + m_2 t_o[T-t])]$ t_i : Entry Tension per unit area t_o : Exit Tension per unit area $m_1 + m_2 = 1$ $0.5 <= m_1 <= 0.667$ $0.333 <= m_2 <= 0.5$ $0 < P_t < 1$ 2. Update the Roll Velocity - Calculate V_r by receiving V_{rp} with time delay AC : every t msec OS : $V_r[T] = V_{rp}[T-t] + BRt$ B : Delay Coefficient R : Random Number Generator (-1 < R < 1)

3. Update the Roll Gap

- Calculate S by receiving S_p with time delay

AC : every t msec

OS : S[T] = S_p [T-t] + CRt

C : Delay Coefficient

R : Random Number Generator (-1 < R < 1)

4. Update the exit velocity of material

- Calculate V_o

- Send V_o to Tension Reel via Exit Speedometer

AC : every t msec

OS : V_o [T] = V_r [T-t] (f[T] + 1)

f[T] =

$$[\frac{\sqrt{r[T-t]}}{2} - \frac{1}{\mu}\sqrt{\frac{h[T-t]}{R}}(\frac{2r[T-t]}{2-r[T-t]} - \frac{t_o[T-t]-t_i[T-t]}{km})\phi]^2$$

ϕ : Correction Coefficient

5. Update the entry velocity of material

- Calculate V_i

- Send V_i to POR via Entry Speedometer

AC : every t msec

OS : V_i [T] = $\frac{h[T-t]}{H[T-t]} V_o[T-t]$

6. Update the material array in WR

- Calculate the material position

	- Copy the Input Movement_array to WR_array - Update H from the WR_array movement - Calculate and update h using roll gap value received from AGC - Copy the portion of WR_array to Output Movement_array - Send Output Movement-array to Tension Reel AC : every t msec OS : if wr_entryp < L movement = V_o * t wr_entryp = wr_entryp + movement - Copy Input Movement_array thicknesses to WR_array by the wr_entryp movement H = WR_array(wr_entryp) - Calculate the exit thickness of material $h = S + \frac{P_a}{M}$ - Update the WR_array thicknesses by the value for movement interval if wr_entryp >= L_t - Copy WR_array thicknesses to Output Movement_array by movement - Send Output Movement_array to Tension Reel
SvM	1. Receive Command Roll Velocity(V_{rp}) from Speed Controller 2. Receive Command Roll Gap(S_p) from AGC 3. Read Entry Tension(t_i) from Entry Tensiometer of POR 4. Receive Movement_array from POR 5. Read Exit Tension(t_o) from Exit Tensiometer of TR

Conditions	

그림 11 : Work Roll 의 RTO 명세

다. Tension Reel

Tension Reel RTO 는 SvM 에서, Speed Controller 로부터 전류를 받고 Work Roll 로 부터는 소재의 출측속도, 출측두께 및 소재를 전달 받으며 소재의 압연 끝나면 소재를 배출한다. SpM 에서는 주기적으로 변화하는 TR 의 반경을 update 하고 소재의 출측 장력을 주기적으로 update 하며 Speed Controller 로 부터 받은 전류를 이용하여 TR 의 속도를 update 하여 TR 을 돌려준다. 또한 이동하는 소재의 위치를 감시하며 Target Material 로 주기적으로 소재를 넘겨준다. 그림 12 에 Tension Reel 의 RTO 명세를 나타내었다.

	Tension Reel
Use	- Speed Controller - Work Roll
ODS	TR : Radius (R_t) Distance to Work Roll (L_t) Velocity (V_t) Torque Constant (K_t) Gear Ratio (N_t) Inertia Moment (J_{mt}) Material : Width (W) Stiffness (k_m) Young Coefficient (E) Length (whole) (L) Movement Array Thickness Array (for the materials in TR) Location of material (tr_entryp) Exit Velocity (V_t) Exit Tension (T_o) Exit Thickness (h)
SpM	1. Update the radius of TR - Calculate the radius of TR at time T with the parameters at just past simulation tick - Send R_t to Speed Controller via TR Radiusmeter AC : every t msec

OS : $R_t[\mathrm{T}] = R_t[\mathrm{T\text{-}t}] + \frac{H[T-t]}{2\pi}\frac{V_t[T-t]}{R_t[T-t]}t$

2. Update the exit tension of material

- Calculate the exit tension of material at time T with the parameters at just past simulation tick
- Send T_o to Speed Controller and Work Roll via Tensiometer

AC : every t msec

OS : $T_o[\mathrm{T}] = T_o[\mathrm{T\text{-}t}] + \frac{Eh[T-t]W}{L_t[T-t]}(V_t[T-t]-V_o[T-t])t$

3. Update the TR Velocity

- Calculate the TR Velocity at time T with the parameters at just past simulation tick

AC : every t msec

OS : $V_t[T] = V_t[\mathrm{T\text{-}t}] + (\frac{K_t R_t[T-t]}{J_t[T]N_t}i_t + \frac{R_t[T-t]^2}{J_t[T]N_t^2}T_o[T-t])t$

+ DRt

D : Delay Coefficient

R : Random Number Generator (-1<R<1)

where

$$J_t[T] = J_{mt} + \frac{\pi}{2}\rho W(R_t[T-t]^4 - R_{ct}[T-t]^4)/N_t^2$$

4. Update the material array in TR

- Calculate the Material Position
- Copy the Movement_array to TR array
- Update h from the TR array movement

	AC : every t msec OS : if tr_entryp < L movement = V_t * t tr_entryp = tr_entryp + movement Copy Movement_array to TR array by the tr_entryp movement
SvM	1. Receive Current (i_t) for TR Motor Drive from Speed Controller 2. Receive Material Exit Velocity (V_o) from Speedometer of Work Roll 3. Receive Material Exit Thickness (h) from Work Roll 4. Receive Movement_array from Work Roll 5. Release Material
Conditions	

그림 12 : Tension Reel 의 RTO 명세

라. Speed Controller

Speed Controller RTO 는 SvM 에서, POR 로부터 반경과 장력을받고 Work Roll 로 부터는 Work Roll 의 속도를 받으며 TR 로부터는 반경과 장력을 받는다. SpM 에서는 일정한 속도 및 장력을 유지시켜 주기위해 각 reel 의 Motor 를 구동시켜주는 전류를 계산하여 각각의 reel 에 이들을 전송한다. 그림 13 에 Speed Controller 의 RTO 명세를 나타내었다.

	Speed Controller
Use	- POR - Work Roll - Tension Reel
ODS	POR : Mandrel Radius, Radius (R_{cp} , R_p) Velocity (V_p) Entry Tension (T_i) Current (i_p) Torque Constant (K_p) Gear Ratio (N_p) Inertia Moment (J_{mp}) Work Roll : Velocity (V_r) Command Velocity (V_{rp}) TR : Mandrel Radius, Radius (R_{ct} , R_t) Velocity (V_t) Exit Tension (T_o) Current (i_t) Torque Constant (K_t) Gear Ratio (N_t) Inertia Moment (J_{mt}) Material : Density (ρ) Width (W)

SpM	1. Update the Current for POR Motor Drive - Calculate the Current for POR Motor Drive - Send to POR Motor Drive AC : every t msec OS : $i_p[\mathrm{T}] = -\frac{R_p[T-t]}{K_p N_p} T_i[T-t]$ 2. Update the WR Command Velocity - Calculate the WR Command Velocity - Send to WR Motor Drive AC : every t msec OS : $V_{rp}[\mathrm{T}] = V_r[\mathrm{T\text{-}t}]$ 3. Update the Current for TR Motor Drive - Calculate the Current for TR Motor Drive - Send to TR Motor Drive AC : every t msec OS : $i_t[\mathrm{T}] = \frac{R_t[T-t]}{K_t N_t} T_o[T-t]$
SvM	1. Read POR Radius (R_p) from Radiusmeter of POR 2. Read Entry Tension (T_i) from Entry Tensiometer of POR 3. Read WR Velocity (V_r) from Speedometer or WR 4. Read TR Radius (R_t) from Radiusmeter of TR 5. Read Exit Tension (T_o) from Exit Tensiometer of TR
Conditions	

그림 13 : Speed Controller 의 RTO 명세

마. AGC

AGC RTO 는 SvM 에서, Work Roll 의 Load Cell 로부터 압하력을 읽는다. SpM 에서는 소재의 Target Thickness 를 유지하기 위한 계산을 하여 Roll Gap 값을 생성하여 이를 Work Roll 에 전송한다. 그림 14 에 Tension Reel 의 RTO 명세를 나타내었다.

	AGC
Use	- Work Roll (Load Cell)
ODS	Roll Force (P_a) Sensed Roll Force (P) Roll Force Setup Value (P_0) Command Roll Gap (S_p) Roll Gap Setup Value (S_0)
SpM	1. Update the Command Roll Gap - Calculate the Command Roll Gap - Send to Hydraulic Actuator in Work Roll AC : every t msec OS : $S_p[T] = S_0 - \frac{P[T] - P_0}{M}$ $P[T] = P_a[T-t] + ERt$ E : Delay Coefficient R : Random Number Generator (-1 < R < 1)
SvM	1. Read Roll Force (P_a) from Load Cell
Conditions	

그림 14 : AGC 의 RTO 명세

4 절 압연공정 AGC 시뮬레이터의 구현

본 시뮬레이터는 실시간 시뮬레이션 엔진으로서 Dream Kernel 상에서 구현되었고 시뮬레이션 언어는 C++DL 을 사용하고 있다.

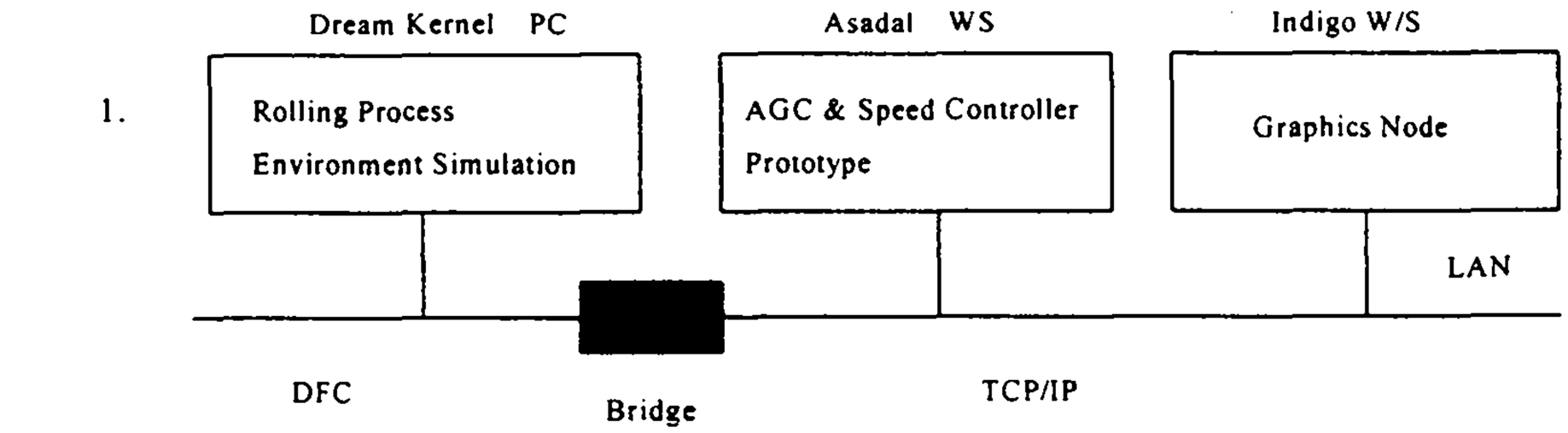
1. 시스템 구성

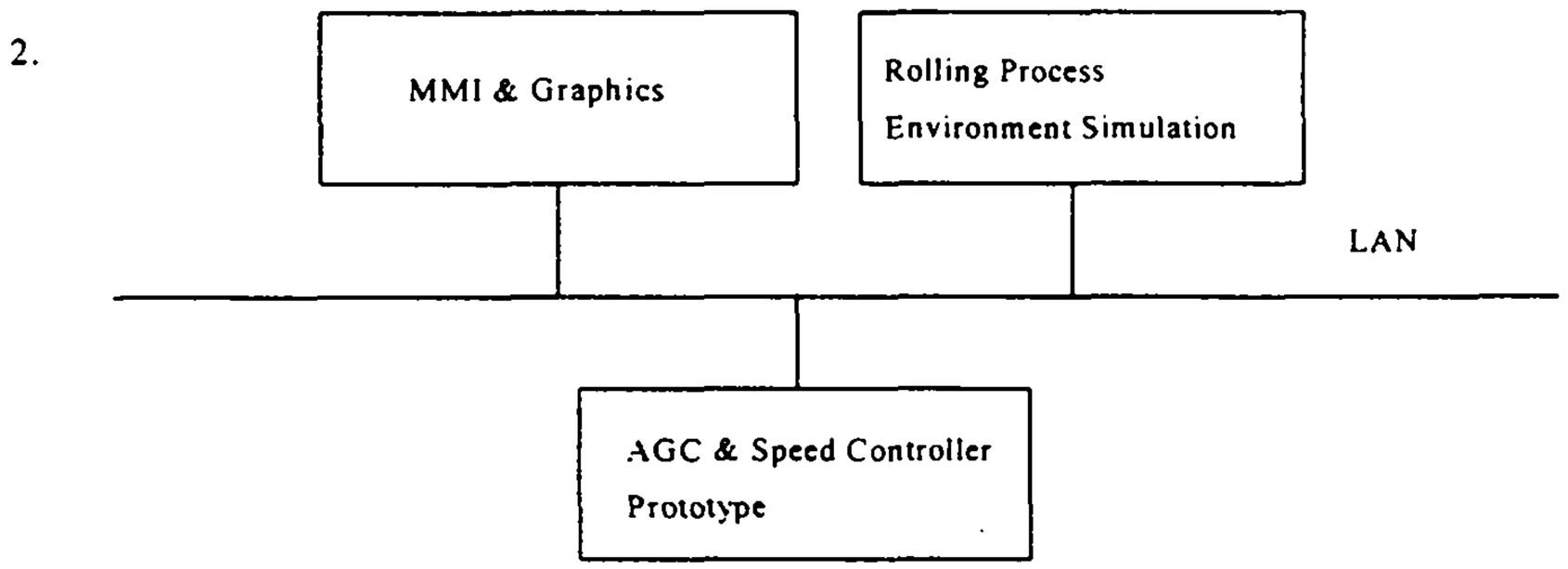
Environment Simulator, Controller 및 Graphic Node 가 Ethernet LAN 위의 Dream Net 상에서 각각의 PC 에 탑재된 분산 시스템을 구성하고 있다. Simulator 및 Controller 는 각각 실시간 주기에 따라 각 객체의 Procedure 를 수행하며 각각의 Node 및 객체간의 통신은 Dream Net 의 DFC 를 이용하여 서로간의 메시지를 주고 받는다. 그림 15 는 AGC 시뮬레이터의 구성도를 보여준다.

DFC 는 메시지 multicasting protocol 로서 Dream Kernel 에 구현되어 있는 통신 프로토콜이며 메시지의 송신측에서 메시지를 broadcasting 한다. 수신측에서는 broadcasting 된 메시지를 계속 수신하며 메시지 ID 를 check 하여 해당되는 메시지를 획득한다.

Graphic Node 는 시뮬레이터에서 broadcasting 하는 메시지를 수신하여 Simulation 변수를 나타내며 이 변수를 이용하여 Simulator 의 Graphic behavio 를 보여준다.

또한 본 연구에서는 Controller 와 Graphics 를 Unix Workstation 에서도 구현하였는데 Dream Kernel 상에서 구현된 Simulator 와 메시지를 서로 교환하여 수행된다. 이때 Controller 와 Simulator 간에는 서로 통신 프로토콜이 다르므로 이들





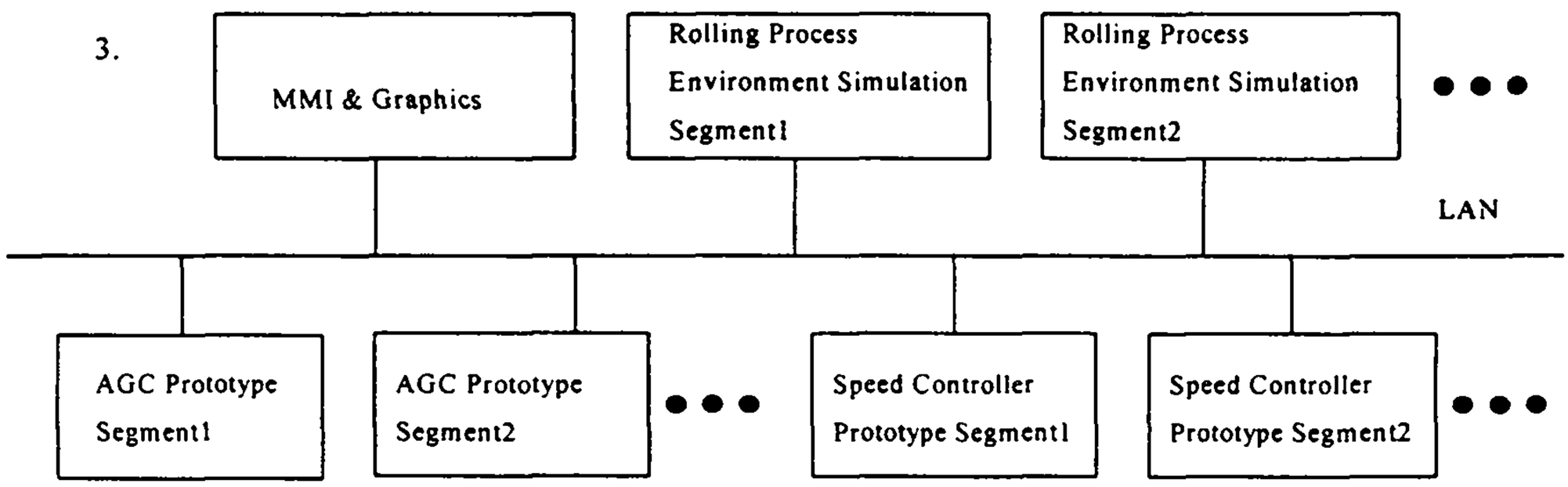


그림 15 : AGC 시뮬레이터의 구성도

사이에 Bridge Node 를 구성하여 Unix W/S 의 TCP/IP 와 Dream Kernel PC 의 DFC 간의 메시지를 번역하여 준다.

그림 15 의 1 은 Bridge 를 통한 Dream Net 상의 Simulator 와 Unix TCP/IP 의 Controller 및 Graphics Node 를 구현한 구성도이며 2 는 Dream Net 상에서 Simulator, Controller, Graphics Node 를 구현한 구성도이다. 3 은 다 스탠드 밀로 확장할 때의 구성도를 나타낸다.

2. 압연공정 매개변수

압연공정 AGC 모델의 시뮬레이션을 수행하기 위하여 실공정 매개변수를 이용하는데 이들 매개변수는 표 1 과 같다. 입측소재 두께 변동은 우선 정현파를 고려하였는데 이는 사용자의 필요에 따라 임의로 선정될 수 있다.

시뮬레이션 주기는 60msec 인데 Bridge 를 이용한 version 에서는 Bridge 의 번역 delay 로 인해 주기를 1sec 로 하였다.

표 1 의 압연공정 매개변수를 이용하여 setup calculation 을 하여 각 변수의 기준 신호 값을 얻을 수 있는데 이들은 표 2 에 나타내었다.

표 1 : 압연공정 매개변수

	Content		Symbol	Value
Rolling Schedule	Referenced Entry Thickness of material		H*	0.002 m
	Target Exit Thickness of material		h*	0.00124 m
	Target Entry Tension per unit area		t_i*	5000000 kg/m^2
	Target Exit Tension per unit area		t_o*	9800000 kg/m^2
	Target Exit Velocity of material		V_o*	5.95083 mps
Work Roll	Work Roll Radius		R	0.20019 m
	Distance to the TR-boundary		L_2	2 m
	Distance to the WR-boundary		L_1	1 m
	Distance to the WR-boundary		L_o	1 m
	Coefficient of Elasticity		M	700000000 kg/m
	Constant	Roll Force Delay	τ_p	0.02 sec
		Hydraulic Actuator Delay	τ_s	0.002 sec
		WR Speed Controller Delay	τ_v	0.02 sec
Material	Length		L	100 m
	Width		W	0.727 m
	Friction Coefficient in the Work Roll		μ	0.032439
	Average Stiffness		k_m	68137000 kg/m^2
	Young Coefficient		E	11500 kg/m^2
	Density		ρ	7.86 kg/m^3
Pay-Off Reel	Torque Coefficient of Motor		K_p	7.1450
	Inetia Moment of Motor		J_{mp}	6.55 kg. m^2
	Gear Ratio		N_p	6.1
	Mandrel Radius		R_{cp}	0.2475 m
	Distance to Work Roll		L_p	3 m

	Distance to the POR-boundary	L_1	2 m
	Initial Radius	R_p *	0.4 m
	Normal Driving Current	i_p *	99 Amp
Tension Reel	Torque Coefficient of Motor	K_t	6.8446
	Inertia Moment of Motor	J_{mt}	0.145 kg. m^2
	Gear Ratio	N_t	6.1
	Mandrel Radius	R_{ct}	0.2475 m
	Distance to Work Roll	L_t	3 m
	Distance to the TR-boundary	L_3	2 m
	Initial Radius	R_t *	0.2475 m
	Normal Driving Current	i_t *	274 Amp

표 2 : Setup Calculation 값

	Content	Symbol	Value
Work Roll	Roll Force	P*	503912 kg
	Roll Gap	S*	0.00052 m
	Entry Velocity of Material	V_i*	3.6895 mps
	Elongation factor	f*	0.00441
	Work Roll Velocity	V_r*	5.9247 mps
Pay-Off Reel	Entry Tension	T_i*	7270 kg
	POR Velocity	V_p*	3.6895 mps
Tension Reel	Exit Tension	T_o*	8834.5 kg
	TR Velocity	V_t*	5.9508 mps

3. Bridge

Bridge 는 Dream Net 의 Broadcasting Protocol 과 Unix 의 TCP/IP 간의 메시지를 번역해 주는 역할을 한다. 이것은 Unix 워크스테이션과 Dream Net PC 간을 연결해 주는 PC 에 구현되고 있으며 이 PC 는 두개의 Ethernet Adaptor 를 가지며 각각은 Dream Net 과 TCP/IP Node 를 연결하고 있다. Bridge 에는 Unix 워크스테이션의 IP 주소를 가지고 있으며 이를 통해 워크스테이션이 Bridge 를 로그인 할 수 있다. Bridge 의 연결구성도는 그림 16 과 같다.

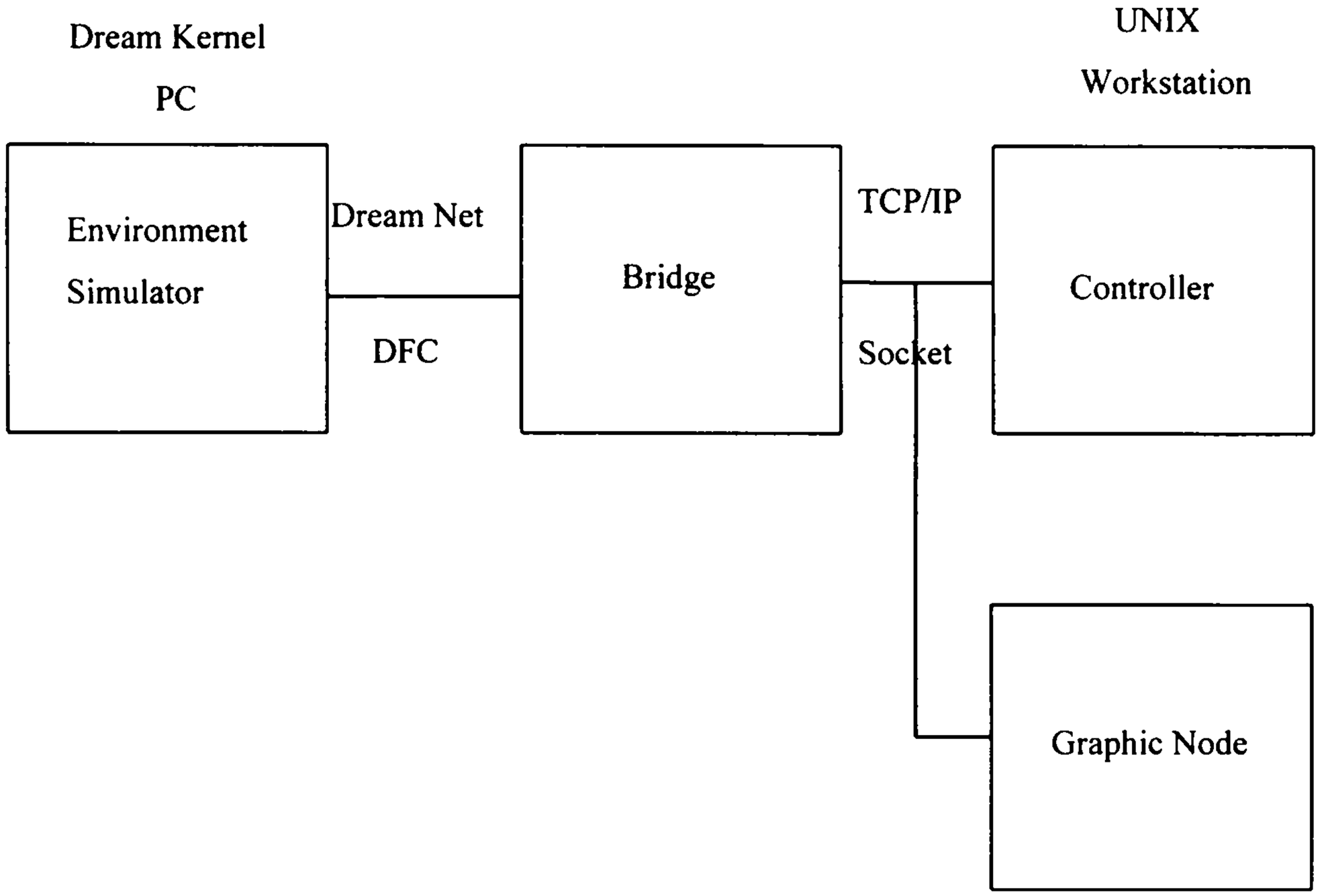


그림 16 : Bridge 를 이용한 연결 구성도

4. 개발 및 구현환경

본 시스템의 개발환경은 다음과 같다.

- Borland C++ Compiler V.4.5
- Dream Library (C++DL) V.3
- TNT Dos Extender

본 시스템의 실행환경은 다음과 같다.

가. PC 환경

- Dream Kernel
- DFC
- Borland C++
- TNT Dos Extender
- Ethernet Driver

나. W/S 환경

- Bridge
- Unix
- TCP/IP
- Ethernet Driver

5 절 LAN 용 실행지원 기능

실시간 시뮬레이션을 실행 시키기 위하여 실행지원 장치(Run-time Support)가 필요하다. 본 연구에서는 UCI 의 위탁과제를 통하여 실시간 시뮬레이션 엔진을 개발하였다.

1. LAN 용 실행지원 기능의 정의 및 설계

실시간 컴퓨터 시스템을 구성하기 위한 기본적 요소는 실시간성을 보증하여 주는 실행지원 엔진이다. 이것은 하드웨어 플랫폼과 함께 실행 엔진을 구성하고 있으며 실시간 응용 및 시뮬레이션에 실시간 서비스를 제공한다. 만약 실행 엔진이 이러한 실시간 서비스 능력을 제공하지 못하면 실시간 응용의 적시성(timeliness)을 보증할 수 없다.

RTO 객체 구조 모델을 지원하는 실행지원 장치로서 전통적인 객체 지향 구조의 실시간 확장의 고유 모델을 지원하고 RTO 객체와 다양한 하드웨어 구조간에 탄력적인 연결이 가능하도록 설계되어야 한다. 이것을 가능하게 하기위해서는 다음을 지원하여야 한다.

- 다양한 구동과 동기화 요구조건을 갖는 실시간 프로세스
- 공유 데이터 구조 감시기
- 실시간 다중전송 논리 채널

프로세스 실행 엔진으로서 실시간 시뮬레이션 엔진은 다음과 같은 세가지 유형의 병행 분산 프로그램 요소들을 지원하여야 한다.

(1) 프로세스 : 이들 응용 프로세스는 I/O 관리 같이 시스템 관리 프로세스의 역할을 하기도 한다. 프로세스는 자신의 다음 계산 세그먼트의 수행을 위해 커널상에 마감시간을 부과하기도 한다.

(2) 공유 데이터 감시기 : 공유 데이터 감시기는 배타적 읽기 및 배타적 쓰기 또는 동시 읽기 및 배타적 쓰기 의미구조를 가져야 한다.

(3) 다중전송 논리 채널 : 프로세스간의 그룹통신을 위하여 다중 전송 논리 채널의 생성 및 논리 채널에 대한 프로세스의 동적 연결을 물리적 네트워크의 고유 특성으로 부터 투명성을 제공해 주어야 한다.

2. PC LAN 용 실행지원 장치 개발

본 연구의 국제공동 연구 파트너인 Kane Kim 교수는 적시성 서비스를 보증하는 실시간 프로세스를 지원하는 운영체제 커널의 모델을 제안 하였다. 이 모델은 Dream Kernel 이라고 불린다. 이 커널은 실시간 객체 지향 구조 모델

(RTO.k)의 접근 방법을 지원하는 실행 엔진으로 개발되었다. 이 실행 엔진은 LAN 용 실행지원 기능의 정의 및 설계 개념을 구현한 것으로 실시간 객체 지향 프로그래밍을 지원하고 다중 프로세스 또는 다중 메소드의 Concurrent Execution 을 지원한다. 이는 커널상에 구현된 CREW 모니터에 의해 자원의 동시 접근의 감시 및 관리기능에 의해 지원되고 있다. 또한 실시간 분산 응용의 LAN 상의 네트워크 통신을 지원하여 이를 실시간 분산 시뮬레이션의 엔진으로 사용하고 있다. 이 실행지원 장치는 Dream Library 를 통해서 응용 프로그래머가 복잡한 하위 수준의 실시간 Construct 를 직접 접근하지 않고서도 코딩할 수 있게 하여 주고 있다. 즉 이러한 Construct 를 Dream Library 에 Class 로 구축하여 놓고 응용 프로그램에서는 해당 Class 를 inherit 하여 쓰면 되는 것이다. 또한 기존의 실시간 프로세스 처리처리 방식에서 RTO 객체지향 구조 프로그램을 지원하도록 구현되었다. 이는 커널 내부에서 실시간 프로세스를 RTO 메소드와 mapping 시켜서 구현되었다.

6 절 실시간 객체지향 언어

RTO 모델을 명확하게 정의하기 위하여 RTO 객체를 구성하는 실용적인 언어 구조가 필요하다. RTO class 의 언어구조는 C++의 확장이며 BNF(Backus-Naur Form)형태로 정의한다.

이 언어는 RTO.k 모델을 지원하며 RTO instance 를 동적으로 생성, 소멸하는 능력을 가져야 하며 다음과 같은 규칙을 따르고 있다.

1) 모든 ODS 와 member 함수들은 RTO instance 의 메소드에 대해 private 하다.
2) RTO 간의 유일한 통신 방법은 DFC 이며 이것은 SvM 호출로 구현되고 사용자에게는 투명하다.

3) Constructor, Initialization_function, New operator 및 SvM 호출을 제외하고는 public member fuction 을 사용하는 RTO 접근 방법은 존재하지 않는다.

4) SvM 호출은 함수 호출과 같이 보이지만 이것은 실제로 메시지 대기 프로세스이다.

5) 모든 RTO 들은 flat 구조를 가진다. 즉 초기의 모든 RTO 정의 및 instance 할당은 전역적이여야 한다. 따라서 국소적 범위 규칙이 적용되지 않는다. 이것은 RTO instance 가 수행되는 메소드를 가지기 때문이다.

6) 하위 수준의 concurrent construct 즉 process, monitor, semaphore, readers/writers 모니터 및 Data Field 는 갖지 않는다.

7) 활성화된 RTO instance 내에서 주어진 절차에 따라 RTO instance 를 동적으로 생성하는 것은 가능하다. 그러나 생성된 instance 는 생성자와 범위관계를 같지 않는다. 이것은 생성된 후 RTO 의 유일한 통신 수단이 DFC 이기 때문이다.

8) 다른 RTO instance 에 의한 RTO 의 동적 소멸은 가능하다.

9) 모든 RTO 를 초기화하는 최상위 프로세스는 main 이다.

10) RTO 에 대하여 모든 C++ class 의 특성들은 보존된다. 예외사항은 생성, 소멸, RTO 의 국부 할당과 RTO 에 대한 public 접근 뿐이다.

11) RTO instance 의 생성 (정적 생성)

RTO instance 의 생성 과정은 2 개의 과정으로 나뉜다. 첫번째는 원래의 C++ constructor 와 똑같은 RTO 데이터와 멤버함수의 할당에 관련된 것이다 두번째는 SpM 과 SvM 의 활성화에 관련된 것이다. 이러한 메소드들은 하위수준의 concurrent process 로서 구현된다. RTO instance 내에서 메소드의 이러한 활성화는 Dream Kernel 이 초기화된 후에 이루어져야 한다. 따라서 RTO instance 의 생성과정은 다음과 같이 두단계로 나누어진다.

1 단계 : // Instance allocation

Speed_Controller POR_Motor (arguments to constructor);

2 단계 : //After Dream.Init() is done,

POR_Motor.Activate();

12) RTO instance 의 생성 (동적 생성)

다른 활성화된 instance 에 의해 RTO instance 가 생성되는 것은 가능하다 또

한 정적인 RTO 활성화의 두단계와 일관성을 유지하기 위하여 동적 생성 역시 두단계로 나뉜다.

Creation : //instance allocation and activation

AGC *ptr;

ptr = new AGC(arg_list);

ptr->Activate();

13) RTO 의 소멸 (동적 소멸)

RTO instance 의 소멸은 RTO instance 의 완전한 제거를 의미한다. 이것은 Dream kernel call 을 호출함으로써 RTO instance 자신이 수행한다.

TSC.Rto_Exit();

이 호출은 다음과 같은 일을 수행한다.

가) RTO 내에 정의된 모든 메소드의 deactivation

나) RTO 의 생명주기동안 할당된 모든 자원의 deallocation

다) 모든 비활성된 메소드의 제거

라) 할당된 모든 ODSS 저장소의 제거 (C++ class 의 삭제)

RTO instance 가 다른 활성화된 RTO 를 제거하기 원한다면 RTO 는 다음과 같이 프로그램되어야 한다.

A Rto instance wanting removal of another Rto.

(Non_blocking) call to SvM that is reponsible for exiting

A Rto instance that accepts the removal request.

An SvM waiting for the removal request

Activated at being called

calls TSC.Rto_Exit();

만약 메소드 제거 요청이 RTO instance 의 종료의 제거와 동기화 되기를 원한다면 클라이언트는 SvM 에 대한 blocking call 을 사용한다. 응답 메시지를 생성하기 위하여는 RTO instance 의 종료가 Rto_Exit() 대신 다음의 kernel call 을 호출하여야 한다.

TSC.Rto_Exit_Reply (Caller_addr, Return_Arguments);

일반적으로 SvM 을 호출함으로써 메시지를 교환하는 것은 메소드 사이에서 수행된다. 그러나 이 경우에, RTO 의 종료는 메소드를 갖지 않는다. 왜냐하면 응답 메시지가 종료 후 전송되어야 하기 때문이다. 따라서 응답 메시지는 위의 kernel call 을 호출함으로써 Dream kernel 에 의해 전송된다.

7 절 결론

본 연구에서는 1 차년도에 개발한 실시간 객체지향 모델링 기법을 이용하여 실시간 시뮬레이션 모델을 설계 구축하였다. 실시간 시뮬레이션 설계 도구로서의 RTO.k 모델은 설계 과정의 오류 검출력과 설계 명확성을 확보할 수 있으며 설계를 용이하게 하여주는 점에 있어서 실시간 시스템 또는 시뮬레이션 개발에 유용한 설계도구임을 확인하였다. 또한 시뮬레이션 코드 생성을 위해

사용한 C++DL 언어는 복잡하고 하위 수준인 실시간 특성들을 Library 로 구축하여 프로그래머로 하여금 class 를 inherit 하여 용이하게 코드를 작성하게 하여준다.

본 연구에서 개발한 압연공정 AGC 시뮬레이터는 실시간 객체지향 개념에 입각하여 구현되었다. 따라서 시뮬레이터의 기능 변경 및 확장이 객체의 비교적 단순한 추가로 쉽게 할 수 있다. 또한 LAN 상의 분산 환경을 이루고 있으며 분산 노드의 변경 또한 손쉽게 할 수 있게 구현되어 있다. 따라서 이 시뮬레이터의 로직 변경이나 향후 확장이 매우 용이하다.

시뮬레이터의 엔진인 Dream kernel 은 분산환경의 실시간 지원능력이 우수함이 확인되었고 실시간 객체지향 언어는 실시간 응용의 코드 생성에 효율적인 언어임을 확인하였다.

3 차년도에는 2 차년도 에서 개발된 단일 밀 AGC 시스템을 다 스탠드 밀 시스템으로 확장하고 시뮬레이션 엔진이 병렬 프로세서를 지원하도록 확장하여 보다 정밀한 시뮬레이션을 구현하도록 할 것이다. 또한 시뮬레이터의 Graphic 기능을 보완 확충하여 다양한 Graphic Animation 을 지원토록 할 예정이며 산업체의 복잡한 실시간 시뮬레이션 응용을 발굴하여 이를 통한 미시적 시뮬레이터 구축에 많은 노력을 기울일 것이다.

여 백

제 2 차년도
연차 보고서

RTS 기술개발

RTS Technology Development

Development of a Real-Time Object Model and Supporting Operating System Facilities for Real-Time Simulation

연구기관

University of California, Irvine

과 학 기 술 처

여 백

제 5 장 실시간 객체 지향모델 및 OS 지원 기능 개발

여 백

Proc. SCS 1996 Western Multiconf. - Object-Oriented Simulation Conf. (eds. by C. Roberts and J. Wallace), Jan. '96, Lajolla, CA, pp.230-235.

An Approach to Real-Time Simulation Based on the RTO.k Object Modeling

K. H. (Kane) Kim & Phillip Sheu

Dept. of Electrical & Computer Engineering
University of California
Irvine, California, U.S.A.

William McCoy & Cuong Nguyen
USN Naval Surface Warfare Center

Chanmo Park & Larry Peterson
Postech, Korea USN NRaD

Abstract: In real-time simulation, the simulation objects are designed to show the same timing behavior that the simulation targets do. The RTO.k object based approach to real-time simulation is discussed in this paper. The RTO.k object is capable of uniformly and accurately representing both real-time embedded computer systems and application environments. This simulation approach has many attractive features, e.g., modifiability, adaptability for efficient parallel processing, etc. In spite of its promising nature, the approach is a fresh one in many respects and some desirable directions for future work aimed toward maturing the technology are also discussed.

1. Introduction

Not only description but also simulation of application environments is often performed as integral steps of validating control computer system designs [Ell93, Guy93, Kim94b, Zei93]. A desirable "accurate" mode of simulating application environments in such situations is the real-time simulation in which *the simulation objects show the same timing behavior that the simulation targets do.*

In order to support such real-time simulation, new approaches to model the simulation targets, especially those which enable precise and concise representation of the timing behavior of the simulation target, are needed. In our view, a preferred modeling approach for use in real-time simulation should be of object-oriented (OO) type, considering the modularity, generality, and natural abstraction benefits that OO approaches bring in.

Since existing object models do not possess adequate capabilities for representing the timing behavior precisely and flexibly, we are forced to search for a proper extension, albeit a drastic one, of the basic object model. Moreover, the object model which possesses strong capabilities for representing the timing behavior precisely and flexibly, is needed to support cost-effective design of real-time embedded computer systems as well. Therefore, finding such extended object models has emerged as one of the most important research issues in the real-time computing field in 1990's [Kim95a]. Ideally, a model which is capable of uniformly and accurately representing both real-time embedded computer systems and application environments, is the most desirable.

We have established such a desirable real-time object model called the RTO.k object model, also called the *time-triggered real-time object* (TT-RTO) model. An initial abstract framework of the model was formulated several years ago jointly by the first co-author and Hermann Kopetz at Technical University of Vienna. This initial framework has evolved into a concrete syntactic structure associated with unambiguous execution semantics in recent years [Kim94a, Kim94b]. The RTO.k object model was partially validated through two specification, design, and real-time simulation experiments conducted recently:
(1) An experiment that involved an application of the RTO.k structuring scheme to both the development of a defense system and that of an environment simulator and
(2) An experiment that involved an application of the RTO.k structuring scheme to both the development of a freeway traffic control system and that of an environment simulator.
These experiments reinforced our belief that the RTO.k model had the necessary representational power and also offered an efficient and rigorous way to develop complex real-time systems and their application environment simulators. The RTO.k object based approach to real-time simulation has many attractive features, e.g., broad applicability, expandability and modifiability, adaptability for efficient parallel processing, etc.

Execution of RTO.k objects (or any other real-time extensions of the basic object model) requires a new type of execution engines which are more reliable and predictable in meeting the timing requirements than existing widely used operating systems including those called real-time operating systems are. Such an execution engine must possess guaranteed timely service capabilities. Otherwise, timely service capabilities of RTO.k objects cannot be guaranteed.

Recently an execution engine model called the DREAM (Distributed Real-Time Ever Available Microcomputing) kernel was defined and then its first prototype, called the DREAM kernel v.D2, was

implemented to run on a network of PC's connected by Ethernet [Kim95b]. It actually supports conventional processes and shared data monitors as well and thus it contains full features of a general-purpose kernel. The DREAM kernel was used in the two major experiments mentioned above.

2. Definition and applications of real-time simulation

We define the real-time simulation as a mode of simulation in which *the simulation objects show the same timing behavior that the simulation targets do.* Real-time simulation involves the use of a real-time clock.

The useful role of real-time simulation during development of complex real-time control computer system was already mentioned in the preceding section. After a control computer system has been implemented, it is often highly useful to connect it to a simulator of the application environment for validation of the control computer system rather than directly proceeding to interface the computer system with the application environment. A highly desirable simulator here is one capable of accurately imitating the timing behavior of the environment, i.e., real-time simulation of the environment.

Also, before the control computer system is fully implemented, it is often useful to do real-time simulation of the control computer system to be implemented. Obviously, such a simulator will not contain all the detailed control algorithms needed in the real control computer systems but instead perform approximate or even dummy control computations and take certain actions at the times at which the real control computer system would take the corresponding actions.

So, when we simulate physical objects in the environment, computation objects performing real-time simulation of the physical objects must show the exact timing behavior of the physical objects. The same is true for simulating computer system designs.
On the other hand, when we simulate both the application environment and a computer system design together, we may choose to scale up or down uniformly in the time dimension both the environment simulator and the computer system simulator, if it serves useful purposes.

3. RTO.k object structuring scheme

3.1 The RTO.k object model

Only a brief overview is given in this section. More details can be found in [Kim94a, Kim95a].

The basic structure of an RTO.k object is depicted in Figure 1. It is an extension of the conventional object model(s) and two most important and unique extensions are the following:

(a) Two clearly separated groups of methods:

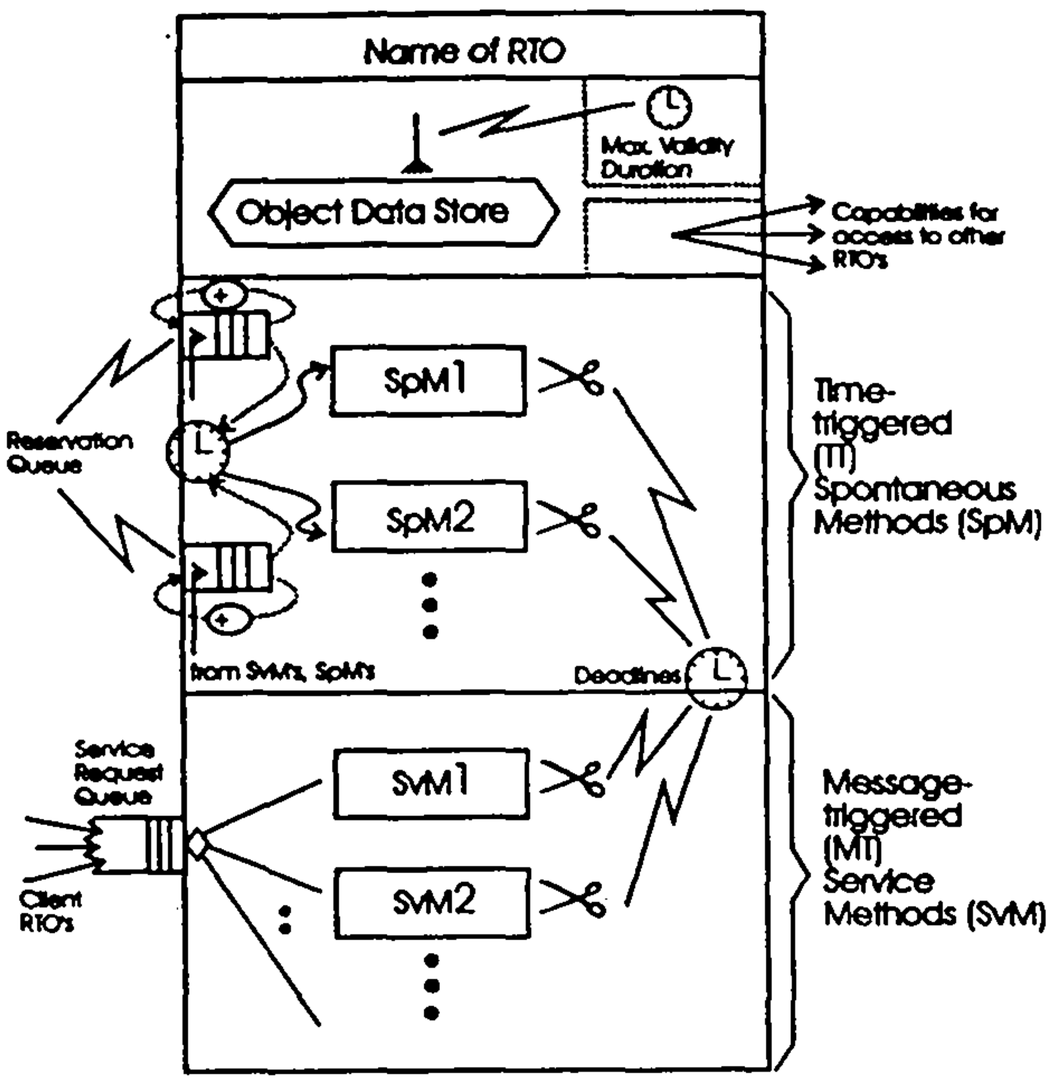


Figure 1. Structure of the RTO.k object model (Adapted from [Kim95a])

For some methods of an RTO.k object, a real-time clock serves as the mechanism for triggering the method executions as the clock reaches some values specified at design time and such methods are called time-triggered (TT-) methods, also called the spontaneous methods (SpM's), and *clearly* separated from the conventional service methods (SvM's) triggered by messages from clients. The two types of methods in an RTO.k object are different not only in the way their executions are triggered but also in that

> "actions to be taken at real times *which can be determined at the design time* can appear only in SpM's".

Therefore, actions of the type "at constant-clock-value do S" or the type "sleep-until constant-clock-value" can appear only in SpM's. SpM's bring in new potential for concurrent executions of object methods in addition to the potential for concurrent executions of SvM's that exist in conventional objects:

> (Type I) Concurrency among SpM executions: E.g., two SpM's designed to be triggered at 10 am.
> (Type II) Concurrency between SpM executions and SvM executions.

(b) Basic concurrency constraint (BCC):

In order to dramatically reduce the designer's efforts in guaranteeing timely service capabilities of RTO.k objects, the execution rule which prevents conflicts between SpM's and SvM's is incorporated. Basically, *activation of an SvM triggered by a message from an external client is allowed only when potentially*

conflicting SpM executions are not in place. To be exact, when a message-triggered SvM is not free of conflict with an SpM in accessing the same portion of the object data space (ODS), execution of the former method (SvM) must not be allowed in a time zone earmarked for a TT-execution of the latter method (SpM). This restriction is called the basic concurrency constraint (BCC). Therefore, SpM's are given higher priorities for execution over the SvM's. Note that this BCC does not impose any restriction on concurrent execution of SpM's or concurrent execution of SvM's. Therefore, executions of SpM's are not disturbed by SvM executions and triggering times of SpM's are fixed at the design time. If a statement of the type "at 10am do S" appears in an SpM, its timely execution can be easily assured.

The above two features make the RTO.k object model clearly distinguished from other proposed real-time object models. In addition, the RTO.k object contains the following features not found in the conventional object model(s):

(c) For each execution of a method of an RTO.k object, a deadline is imposed;

(d) Real-time data contained in an RTO.k object become invalid after the interval called the maximum validity duration passes.

Triggering times for SpM's must be fully specified as constants during the design time. Those real-time constants appear in the first clause of an SpM specification called the autonomous activation condition (AAC) section. An example of an AAC is

"for t = from 10am to 10:50am every 30min
start-during (t, t+5min) finish-by t+10min"

which has the same effect as

{"start-during (10am, 10:05am) finish-by
start_time+10min",
"start-during (10:30am, 10:35am) finish-by
start_time+10min"}.

A provision is also made for making the AAC section of an SpM contain only candidate triggering times, not actual triggering times, so that a subset of the candidate triggering times indicated in the AAC section may be dynamically chosen for actual triggering. Such a dynamic selection occurs when an SvM within the same RTO.k object requests future executions of a specific SpM. The AAC specifying candidate triggering times rather than actual triggering times starts with a declaration "if-demanded". Therefore, there are two different modes of determining triggering times for SpM's:

(a) fully determined during the system design, in which case the SpM is said to be *statically scheduled*, and
(b) determined during the run time when an SvM requests executions of the SpM and designates a subset of the candidate triggering times prepared during the design time as actual triggering times, in which case the SpM is said to be *partially dynamically scheduled.*

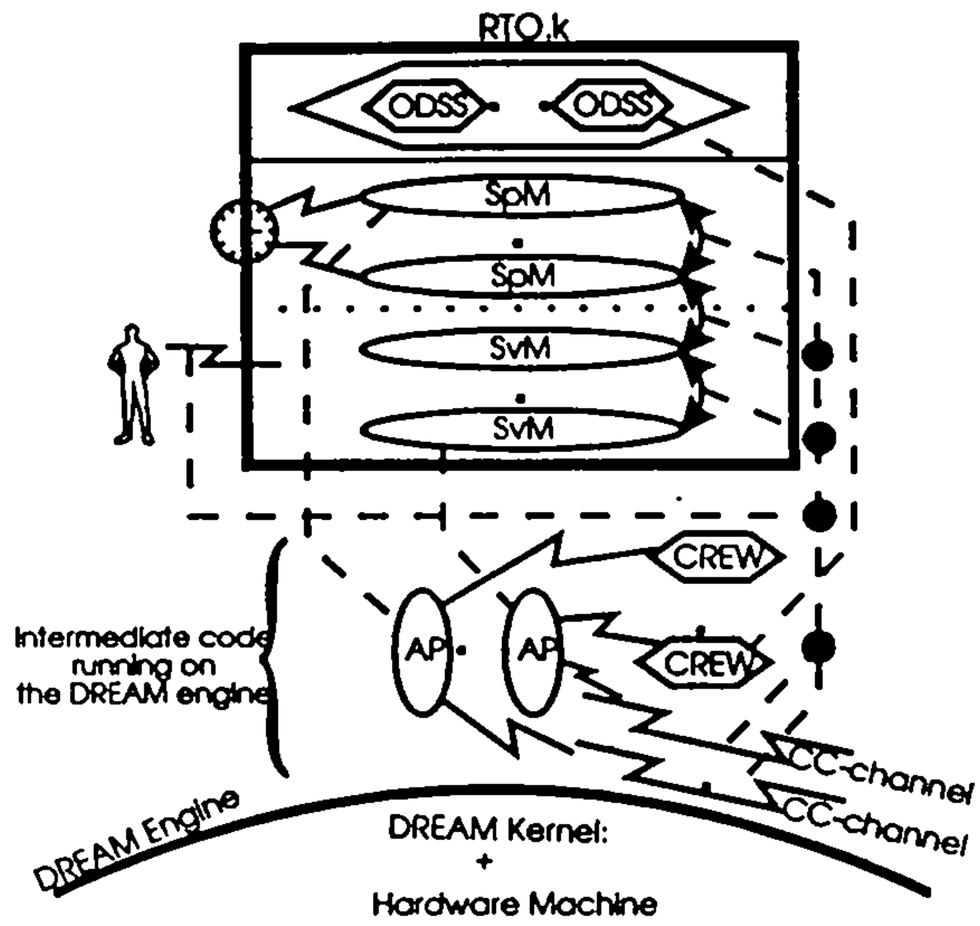


Figure 2. Mapping an RTO.k object to a process-structured-program (Adapted from [Kim95b])

An underlying design philosophy of the RTO.k object model is that an RTCS will always take the form of a network of RTO.k objects. RTO.k objects interact via calls by client objects for service methods in server objects. The caller may be an SpM or an SvM in the client object. In order to facilitate highly concurrent operations of client and server objects, non-blocking (sometimes called asynchronous) types of calls (i.e., service requests) can be made to SvM's.

The designer of each RTO.k object provides a guarantee of timely service capabilities of the object by indicating the *deadline for every output* produced by each SvM (and each SpM which may be executed on requests from SvM's) in the specification of the SvM (and some relevant SpM's) advertised to the designers of potential client objects. Before determining the deadline specification, the server object designer must convince himself/herself that with the object execution engine (hardware plus operating system) available, the server object can be implemented to always execute the SvM such that the output action is performed within the deadline. Again, the BCC contributes to major reduction of these burdens imposed on the designer.

3.2 DREAM kernel as an execution engine for RTO.k objects

When the DREAM kernel executes RTO.k objects, it actually executes equivalent programs composed of *processes*, shared data structure monitors called *CREW (concurrent-read exclusive-write) monitors*, and logical multicast channels called *CC- (content code) channels.* Figure 2 depicts the mapping relationship between components of an RTO.k object and the corresponding components of an equivalent process-structured program. As shown,

(1) object methods, both SpM's and SvM's, are mapped to processes,
(2) *object data space (ODS) segments* (ODSS's) to CREW monitors,
(3) access paths to SvM's to CC-channels, and
(4) result-return paths to clients to CC-channels.
This way of utilizing CC-channels facilitates the transparency of object locations.

When an SpM is mapped to a process, the process must present to the DREAM kernel (1) the AAC of the SpM, (2) the associated deadline specification, and (3) access rights of the SpM for ODSS's. Similarly, when an SvM is mapped to a process, the process must present to the DREAM kernel (1) the associated deadline specification, (2) access rights of the SvM for ODSS's, and (3) other information related to its pipelined execution possibilities.

As mentioned earlier, the first prototype, called the DREAM kernel v.D2, was implemented to run on a network of PC's connected by Ethernet. It contains full features of a general-purpose kernel.

4. An RTO.k object model based approach to real-time simulation

The object model was initially formulated and used in simulation applications [Dah72]. Therefore, use of the object model in modeling the environment objects, i.e., modular entities in the environment which have time-varying internal states, has been practiced for a long time. However, as mentioned in the preceding section, the RTO.k object model, supports more accurate detailed modeling of environment objects.

Suppose the application environment chosen is a sky+land+sea segment of interest, called the "theater",

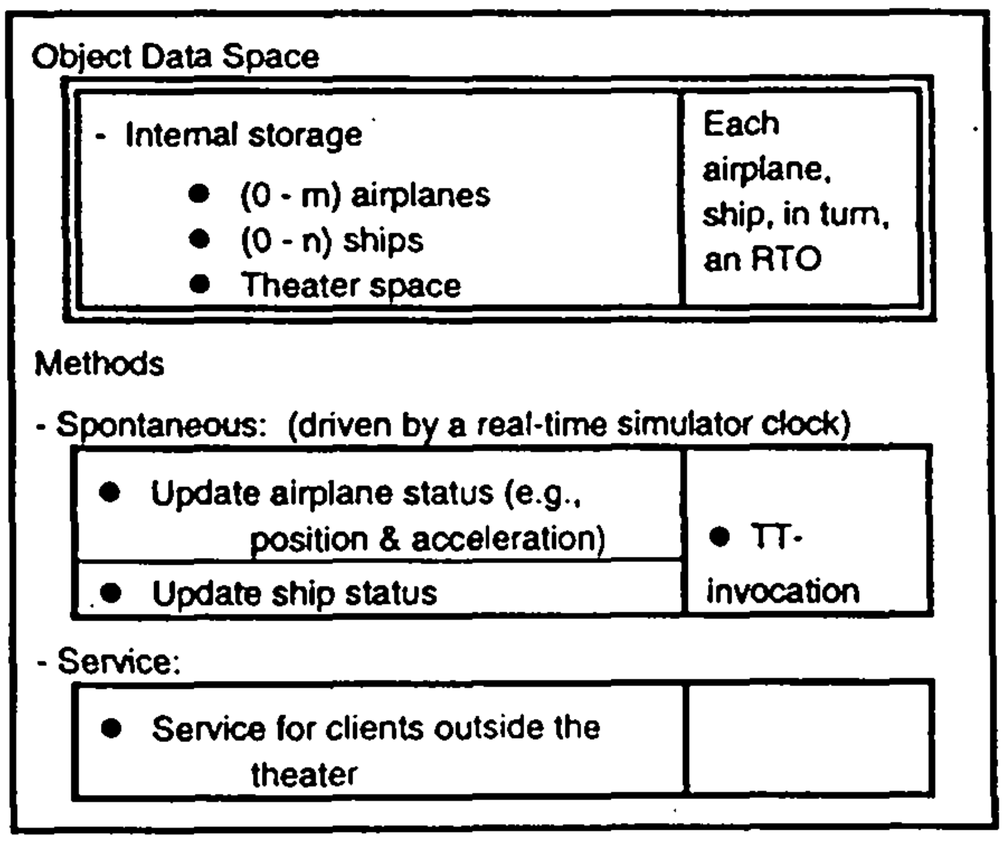


Figure 3. An RTO.k structured simulation model for a theater (Adapted from [Kim94a])

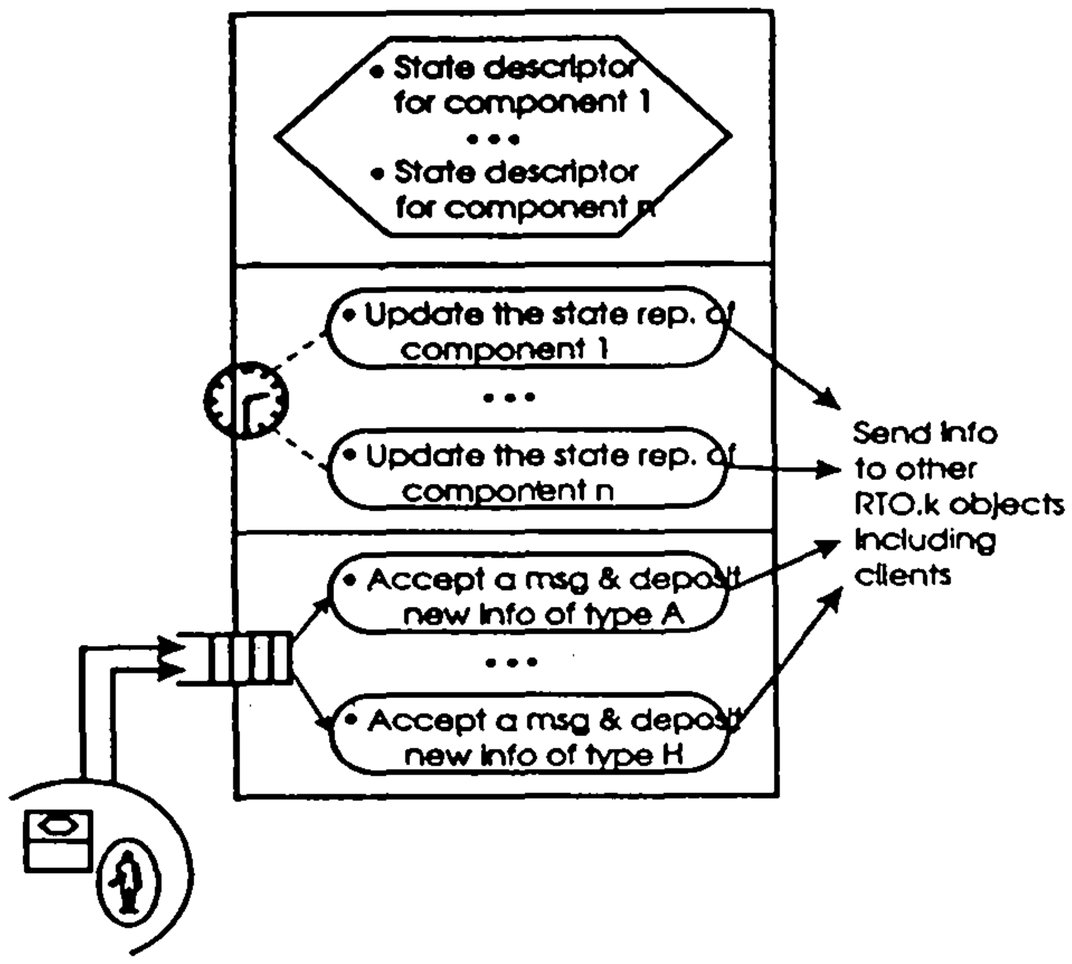


Figure 4. The general structure for the RTO.k object based simulation model

and includes moving objects such as ships and airplanes, etc. This environment can be represented and simulated by the RTO.k object in Figure 3.

The object data space (ODS) in this RTO.k object contains state representations of the airplanes, the ships, and the theater space.

Each TT-method, when executed, updates a variable-set in the ODS representing the state of some physical object (i.e., airplane, ship) to reflect the current state of the physical object. Ideally the TT-methods should be *activated continuously* and each of their executions be *completed instantly*. This is fine when the object is used merely as a description rather than an executable program. However, such an execution engine for the RTO.k object cannot exist and thus we must adopt the less precise version of the model in which the time domain is a discrete domain, as an executable simulation model. That is, the limited power of the simulation engine dictates the activation frequency of any TT-method to be no more than once per every *simulator clock tick* while allowing each execution to be completed before or by the time of the following activation of the same method. Therefore, TT-methods are the mechanisms for simulating continuous state changes that occur naturally in the environment objects.

The natural parallelism that exists among the environment objects is precisely represented by use of multiple TT-methods which may be activated simultaneously. In general, the accuracy of an RTO.k object structured simulation of the environment is a direct function of the activation frequencies of TT-methods.

Figure 4 depicts the RTO.k based real-time simulation approach in a generic form.

The single RTO.k representation in Figure 3 can be expanded into a more detailed representation structured in the form of a network of RTO.k objects, each representing

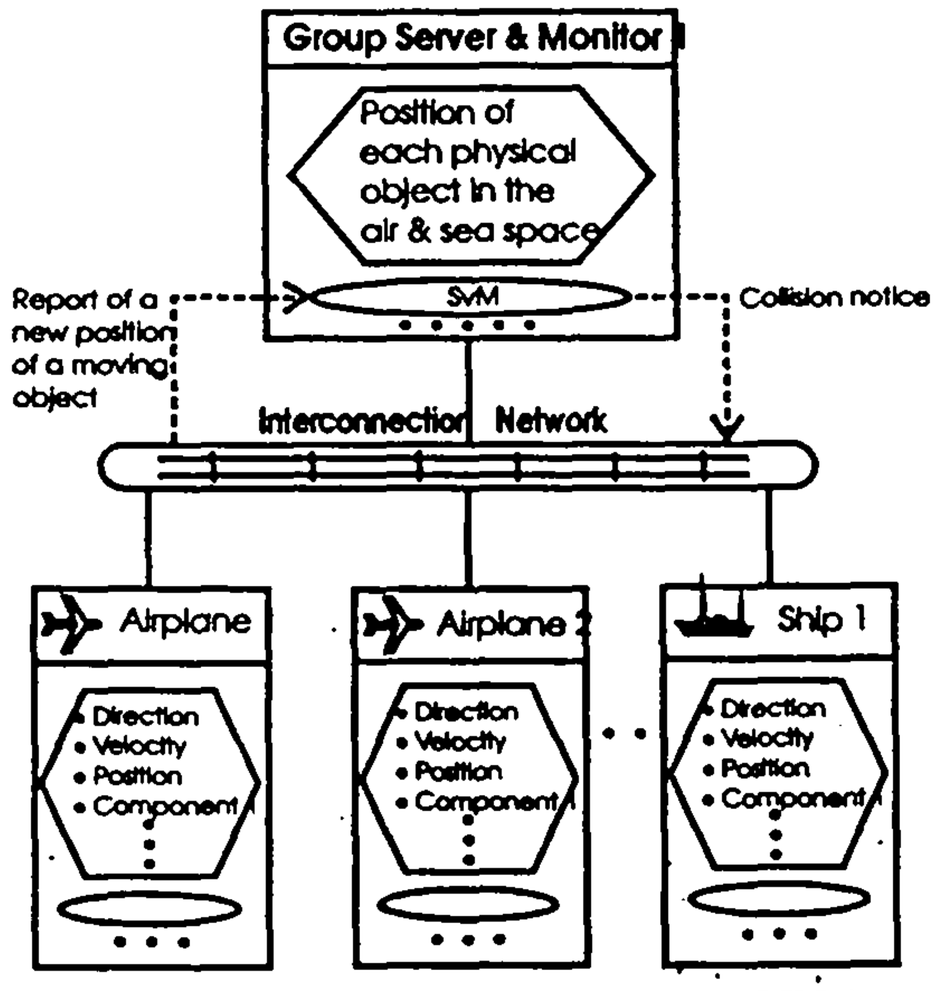


Figure 5. Decomposition of the theater RTO.k in Figure 3.

an airplane or ship, as shown in Figure 5.

Figure 5 also depicts an interesting role played by the RTO.k object representing the air and sea space. This RTO.k object maintains information on how the space is occupied. It facilitates detecting collisions between moving objects such as airplanes, etc. As the RTO.k object structured simulator of an airplane progresses after each simulator clock tick, the position of the airplane is updated and recorded within the RTO.k object. In addition, this RTO.k object notifies the RTO.k object representing the space. The latter RTO.k object in turn checks if the airplane has now collided with any other environment object such as an airplane, ship, etc., and, if so, notifies the RTO.k objects simulating the collided environment objects. The notified RTO.k objects then start simulating the post-collision behavior of their simulation targets.

In a sense, the RTO.k object representing the space supports the RTO.k objects simulating the moving environment objects. Therefore, it is called a group server and monitor (GSM) RTO.k object. If the workload for a GSM object becomes too large, then multiple GSM objects, each supporting a different group of RTO.k objects simulating the physical environment objects, can be utilized.

Therefore, the RTO.k object model is an effective mechanism not only for variable-degree abstraction of real-time distributed computer systems (DCS's) under design but also for variable-accuracy simulation of the application environments. Structures and notations used will be of the same kind in both control system design and environment simulation. The combination of a control system design and an environment simulator takes the form of a network of RTO.k objects. Therefore, the RTO.k object structuring scheme facilitates uniform structuring of both control computer systems and application environment simulators. This presents considerable potential benefits to the system engineers.

Besides the uniform structuring capabilities, the RTO.k object based real-time simulation approach has some other unique advantages. First, due to the strong modularity characteristics of the RTO.k object, changes in the simulation target can be easily handled while achieving real-time simulation. For example, appearance of a new type of an environment object entails just adding a new RTO.k object without requiring modifications of the other existing RTO.k objects and the real-time simulation capability remains intact. Similarly, changes in some timing or logical behavior of an environment object will cause only some changes inside corresponding RTO.k objects and again the real-time simulation capability is maintained just with that.

Secondly, in establishing a large-scale simulator, distributed and parallel (D&P) execution of simulation objects is often necessary. Such execution techniques are almost essential tools for realizing full potential benefits of real-time simulation of complex application environments or system designs. Conventional D&P simulation approaches involve message communication among the simulator execution nodes for the purpose of keeping them synchronized. In one type of D&P approaches, the processing nodes progress in parallel in a lock-step mode. They all advance to the next simulation step after ensuring that all nodes have completed the current simulation step. In another type of D&P approaches, they advance somewhat asynchronously but as they often discover after exchanging messages that they have executed some incorrect simulation steps, they backtrack to undo those steps.

In contrast, no synchronization messages need to be exchanged in D&P execution of RTO.k simulator objects. This is because each simulation step with an RTO.k simulator object is triggered by a real-time clock. Therefore, as long as the triggering interval in each SpM (i.e., the inverse of the state updating frequency) in an RTO.k simulator object is chosen to be sufficiently long to cover the execution of a simulation step by any processing node, the nodes can safely start the next step simultaneously without exchanging synchronization messages. This can be more efficient in large-scale simulations which tax the power of a large-scale D&P processing system.

Thirdly, as mentioned earlier, when we simulate both the application environment and a computer system design together, we may choose to scale up or down uniformly in the time dimension both the environment simulator and

the computer system simulator, depending upon the power of the simulator execution engine available. For example, by the simple action of slowing down the real-time clock in the simulator execution engine, we can reduce the load presented to the overloaded execution engine. Such scaled real-time simulation can still serve most of the originally intended objectives of closed loop real-time simulators since the order of events and the ratio of the duration of one activity over that of another are still accurately exhibited.

As mentioned in the introduction, the RTO.k object model was partially validated through two specification, design, and real-time simulation experiments conducted recently, one based on a defense application scenario and the other based on an advanced freeway traffic control scenario. Due to the space limit, these experimental study are not discussed in this paper.

5. Future directions

The real-time simulation can play useful roles in many phases of the complex system engineering cycle, in particular, validation and evaluation of control computer system designs as they evolve through abstract designs to detailed implementations. Not only logical behavior but also timely performance and dependability characteristics can be effectively validated and evaluated via real-time simulation.

The RTO.k object based approach to real-time simulation has been presented in this paper. The RTO.k object is capable of uniformly and accurately representing both real-time embedded computer systems and application environments. The RTO.k object based approach to real-time simulation has many attractive features, e.g., broad applicability, expandability and modifiability, adaptability for efficient parallel processing, etc. Much further work is needed to develop supporting tools for the RTO.k object based real-time simulation. First of all, a high-level language tool for RTO.k object structuring needs to be established along with a supporting translator. We are currently developing an extension of C++, which we have named C++T. Secondly, an efficient execution engine, especially one capable of effectively taking advantage of the raw processing power of highly parallel machines, needs to be developed. Adapting the DREAM kernel to a highly parallel machine such as Intel Corp.'s Paragon, is considered to be a worthwhile research and development effort.

One important feature that is essential for a real-time simulation system is visualization. Ideally, real-time display of the dynamically changing states of the simulation targets should not fall behind the real-time simulation beyond a certain fixed time interval. This often means the need for an efficient visualization algorithm. The visualization algorithms should preferably be incremental in the sense that given a scene and some object movements, the new scene can be computed with minimal effort. With such an algorithm, the display of a scene can be composed from pieces of contributions made by the objects. Visualization of RTO.k objects is thus another meaningful topic for future research and development.

Acknowledgment: The research work reported here was supported in part by US Navy, NSWC Dahlgren Division under Contract No. N60921-92-C-0204, in part by the California Transportation Department via the UCI ITS, in part by the University of California MICRO Program under Grant No. 93-080, in part by Hitachi, Ltd, and in part by Postech. The efforts of Larry Peterson were also supported by US Navy NRaD.

References

[Dah72] Dahl, O.J., "Hierarchical Program Structuring", in Dahl, Dijkstra, & Hoare eds., '*Structured Programming*', Acad. Press, NY, 1972.

[Ell93] Ellenberger, R., Ling, R., Buscher, D., Uhde-Lacovara, J., and Shuler, R., "Automatic Generation of Real-Time Ada Simulations for Space Station Freedom", *Simulation*, Nov.1993, pp.337-345.

[Guy94] Guyse, C., Buscher, D., and Ellenberger, R., "Real-time Environment and Vehicle Dynamics Simulations for Space Station Freedom Integrated Test and Verification Environment", *Simulation*, Apr.1994, pp.230-239.

[Kim94a] Kim, K.H. et al., "Distinguishing Features and Potential Roles of the RTO.k Object Model", *Proc. 1994 IEEE CS Workshop on Object-oriented Real-time Dependable Systems (WORDS)*, Oct. '94, Dana Point, pp.36-45.

[Kim94b] Kim, K.H. and Kopetz, H., "A Real-Time Object Model RTO.k and an Experimental Investigation of Its Potentials", *Proc. 1994 IEEE CS Computer Software and Applications Conf. (COMPSAC)*, Nov. 1994, Taipei, pp.392-402.

[Kim95a] Kim, K.H., "Toward New-Generation Real-Time Object-Oriented Computing", *Proc. IEEE CS 5th Workshop on Future Trends of Distributed Computing Systems (FTDCS)*, Cheju Island, Aug. '95, pp.520-529.

[Kim95b] Kim, K.H. et al., "A Timeliness-Guaranteed Kernel Model -- DREAM Kernel -- and Implementation Techniques ", *Proc. RTCSA '95 (1995 Int'l Workshop on Real-Time Computing Systems & Applications)*, Tokyo, Oct. '95, pp.80-87.

[Zei93] Zeigler, B., and Kim, J., "Extending the DEVS-Scheme Knowledge-Based Simulation Environment for Real-Time Event-Based Control", *IEEE Trans. on Robotics and Automation*, Vol.9, No.3, 6/93, pp.351-356

Proc. COMPSAC '96 (IEEE CS 20th Int'l Computer Software & Applications Conf.), Seoul, August 1996, pp. 176-183.

Real-Time Simulation Techniques Based on the RTO.k Object Modeling

K. H. (Kane) Kim

Dept. of Electrical & Computer Engineering
University of California
Irvine, California, U.S.A.

Cuong Nguyen
USN Naval Surface Warfare Center, U.S.A.

and

Chanmo Park
Postech, Korea

Abstract: Real-time simulation is an advanced mode of simulation in which the simulation objects are designed to show the same timing behavior that the simulation targets do. A new approach to real-time simulation which is based on the RTO.k object modeling is discussed in this paper. The RTO.k object, which is a real-time extension of the well-established object structure, is capable of uniformly and accurately representing both real-time embedded computer systems and application environments. This simulation approach has many attractive features, e.g., expandability, modifiability, adaptability for efficient parallel processing, etc. In spite of its promising nature, the approach is an immature one in many respects and some desirable directions for future work aimed toward maturing the technology are also discussed.

Index Terms: real-time simulation, real-time object, simulator clock, parallel processing, distributed computing, real-time clock, RTO.k

1. Introduction

An accurate mode of simulation called the real-time simulation in which *the simulation objects show the same timing behavior that the simulation targets do*, is under increasing demands. For example, continuing advances in virtual reality applications accompany increasing demands for more powerful real-time simulation capabilities. Numerous other examples can also be found in the real-time computer control field. Not only description but also simulation of application environments is often performed as integral steps of validating control computer system designs [Ell93, Guy93, Kim94b, Zei93]. A desirable mode of simulating application environments in such situations is the real-time simulation.

In order to support such real-time simulation, new approaches to model the simulation targets, especially those which enable multi-fidelity representation of the timing behavior of the simulation target, are needed. In our view, a preferred modeling approach for use in real-time simulation should be of object-oriented (OO) type, considering the modularity, generality, and natural abstraction benefits that OO approaches bring in.

Since existing object models do not possess adequate capabilities for representing the timing behavior accurately, we are forced to search for a proper extension, albeit a drastic one, of the basic object model. Moreover, the object model which possesses strong capabilities for representing the timing behavior accurately and varying degrees of precision, is needed to support cost-effective design of real-time embedded computer systems as well. Therefore, finding such extended object models has emerged as one of the most important research issues in the real-time computing field in 1990's [Kim95a]. Ideally, a model which is capable of uniformly and accurately representing both real-time embedded computer systems and application environments, is the most desirable.

We have established one candidate for such a desirable real-time object model, called the RTO.k object model, which is also called the *time-triggered real-time object* (TT-RTO) model. An initial abstract framework of the model was formulated several years ago jointly by the first co-author and Hermann Kopetz at Technical University of Vienna. This initial framework has evolved into a concrete syntactic structure associated with unambiguous execution semantics in recent years [Kim94a, Kim94b]. The RTO.k object model was partially validated through two specification, design, and real-time simulation experiments conducted recently, one based on a defense application scenario and the other based on an advanced freeway traffic control scenario. These experiments reinforced our belief that the RTO.k model had the necessary representational power and also offered an efficient and rigorous way to develop complex real-time systems and their application environment simulators.

The RTO.k object based approach to real-time simulation has many attractive features, e.g., broad applicability, expandability and modifiability, adaptability for efficient parallel processing, etc. Real-time simulation of complex simulation targets, e.g., microscopic simulation of metropolitan area automobile traffic, cannot be realized without efficient use of distributed and/or parallel processing capabilities. The RTO.k object structuring scheme supports efficient distributed and/or parallel processing.

Execution of RTO.k objects (or any other real-time extensions of the basic object model) requires a new type of execution engines which are more reliable and predictable in meeting the timing requirements than existing widely used operating systems (including those called real-time operating systems) are. Such an execution engine must possess guaranteed timely service capabilities. Otherwise, timely actions of RTO.k objects cannot be guaranteed.

Recently an execution engine model called the DREAM (Distributed Real-time Ever Available Microcomputing) kernel was defined and then a series of its prototypes, of which the latest was called the DREAM kernel v.D3, was implemented to run on a network of PC's connected by Ethernet [Kim95b, Kim96]. It actually supports conventional processes and shared data monitors as well and thus it contains full features of a general-purpose kernel. The DREAM kernel was used in the two major experiments mentioned above.

In this paper, basic principles of the RTO.k object based real-time simulation scheme are presented. Illustrations are made by using examples drawn from the two application areas, the simulation of a defense environment and the simulation of an automobile traffic. In Section 2, the definition as well as essential properties of real-time simulation are discussed. After an overview of the RTO.k object structuring scheme is given in Section 3, the essence of an RTO.k object based approach to real-time simulation is discussed in Section 4. Section 5 provides a brief discussion on the experiments conducted recently and the paper concludes with the discussion on future research issues in Section 6.

2. Definition, essential properties, and applications of real-time simulation

The real-time simulation was already defined in the introductory section as a mode of simulation in which the simulation objects show the same timing behavior that the simulation targets do. Real-time simulation involves the use of a real-time clock.

The useful role of real-time simulation during development of complex real-time control computer system was already mentioned in the preceding section. After a control computer system has been implemented, it is often highly useful to connect it to a simulator of the application environment for validation of the control computer system rather than directly proceeding to interface the computer system with the application environment. A highly desirable simulator here is one capable of accurately imitating the timing behavior of the environment, i.e., real-time simulation of the environment.

Also, before a complex control computer system is fully implemented, it is often useful to do real-time simulation of the control computer system to be implemented. Obviously, such a simulator will not contain all the detailed control algorithms needed in a fully implemented control computer system but instead perform approximate or even dummy control computations and take certain actions at the times at which a fully implemented control computer system would take the corresponding actions.

So, when we simulate physical objects in the environment, computation objects performing real-time simulation of the physical objects must show the exact timing behavior of the physical objects. The same is true for simulating control computer systems.

On the other hand, when we simulate both the application environment and a control computer system (to be implemented) together, we may choose to scale up or down uniformly in the time dimension both the environment simulator and the computer system simulator, if it serves useful purposes.

Another growing application field of the real-time simulation is the virtual reality field. A virtual reality environment corresponding to a dynamic physical environment, e.g., a compartment in a moving train or a flying airplane, is not of high quality if the virtual environment does not change at the same tempo at which the physical counterpart changes.

In a real-time simulator, the *simulator clock* must "tick" at a steady rate. Each tick of the simulator clock is commenced and administered by referencing a real-time clock in the *simulation execution engine* (a computer running the simulation program). The ticking rate of the simulator clock in a real-time simulator must be chosen such that during any ticking interval, all (real-time) events which should be simulated during that interval may be treated as being "contemporary". In other words, the microscopic order in which events are simulated during a ticking interval should be immaterial as far as simulation results are concerned. This means that all events may be treated as ones occurring at the end of the ticking interval (i.e., right before the next ticking of the simulator clock). This is a fundamental requirement, which may be called the simulator clock atomicity requirement. All computational activities taking place during a ticking interval of the simulator clock may be viewed as one *simulation-step*.

For example, consider the case of simulating automobiles moving on a freeway. Since the moving pattern of each automobile is affected by the moving automobiles in the neighborhood area, the selection of the ticking interval of the simulator clock is an important matter. If the ticking interval is chosen to be 5 seconds, then the state of every automobile will be updated every 5 seconds. However, if a typical automobile can change its lane in one second and a lane change by an automobile can have big impacts on subsequent behaviors of some other automobiles, updating the states of all automobiles every 5 seconds means a very low-fidelity low-accuracy simulation. Especially, if automobile collisions are to be

dealt with in this simulation, it is possible that a collision occurring at a certain time can lead to the same simulation result as a collision occurring 4.9 seconds later does. This is due to the simulator clock atomicity requirement. That is, all collisions occurring during a ticking interval may be treated as ones occurring at the end of the ticking interval.

On the other hand, if the states of all (say, 10000) automobiles were to be updated every 100 milli-seconds for the sake of realizing high-accuracy real-time simulation, then the simulation model may impose excessive computational load on the execution engine. Therefore, the ticking interval of the simulator clock cannot be made indefinitely small.

If the simulation activities to take place during a ticking interval of the simulator clock always require a portion of the uni-processor engine power available, executing the real-time simulation model is not a difficult problem. The problem which is often non-trivial is to formulate such a real-time simulation model which contains an appropriate ticking rate of the simulator clock while realizing sufficient-accuracy real-time simulation. The probability of a low-load situation mentioned above arising becomes higher as the ticking interval of the simulator clock becomes larger, which in turn means that the simulation model becomes more crude. Suppose a large-scale simulation model is given. As the ticking interval becomes smaller, the probability of not completing all relevant simulation activities within a ticking interval becomes higher and thus the necessity of using a multi-processor (or parallel and/or distributed computer) execution engine increases. In a multi-processor execution engine, the clocks used in different processor-nodes must all be tightly synchronized.

3. RTO.k object structuring scheme

3.1 The RTO.k object model

Only a brief overview is given in this section. More details can be found in [Kim94a, Kim96].

The basic structure of an RTO.k object is depicted in Figure 1. It is an extension of the conventional object model(s) and two most important and unique extensions are the following:

(a) Two clearly separated groups of methods:

For some methods of an RTO.k object, a real-time clock serves as the mechanism for triggering the method executions as the clock reaches some values specified at design time and such methods are called time-triggered (TT-) methods, also called the spontaneous methods (SpM's), and *clearly* separated from the conventional service methods (SvM's) triggered by messages from clients. The two types of methods in an RTO.k object are different not only in the way their executions are triggered but also in that

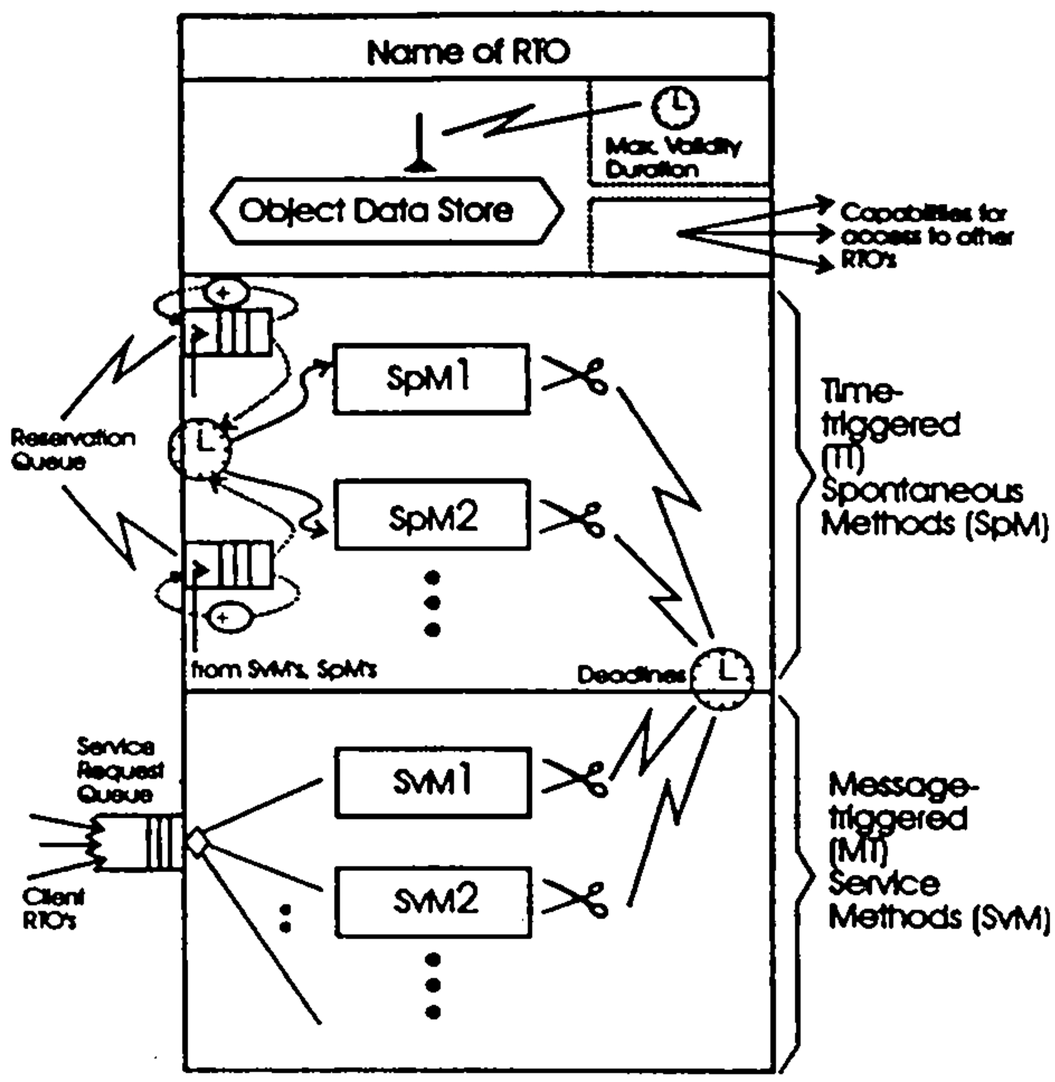


Figure 1. Structure of the RTO.k object model (Adapted from [Kim94a])

"actions to be taken at real times *which can be determined at the design time* can appear only in SpM's".

Therefore, actions of the type "at constant-clock-value do S" or the type "sleep-until constant-clock-value" can appear only in SpM's. SpM's bring in new potential for concurrent executions of object methods in addition to the potential for concurrent executions of SvM's that exist in conventional objects:

(Type I) Concurrency among SpM executions: E.g., two SpM's designed to be triggered at 10 am.

(Type II) Concurrency between SpM executions and SvM executions.

(b) Basic concurrency constraint (BCC):

In order to dramatically reduce the designer's efforts in guaranteeing timely service capabilities of RTO.k objects, the execution rule which prevents conflicts between SpM's and SvM's is incorporated. Basically, *activation of an SvM triggered by a message from an external client is allowed only when potentially conflicting SpM executions are not in place.* To be exact, when a message-triggered SvM is not free of conflict with an SpM in accessing the same portion of the object data space (ODS), execution of the former method (SvM) must not be allowed in a time zone earmarked for a TT-execution of the latter method (SpM). This restriction is called the basic concurrency constraint (BCC). Therefore, SpM's are given higher priorities for execution over the SvM's. Note that this BCC does not impose any restriction on concurrent execution of SpM's or concurrent execution of SvM's. Therefore, executions of

SpM's are not disturbed by SvM executions and triggering times of SpM's are fixed at the design time. If a statement of the type "at 10am do S" appears in an SpM, its timely execution can be easily assured.

The above two features make the RTO.k object model clearly distinguished from other proposed real-time object models. In addition, the RTO.k object contains the following features not found in the conventional object model(s):

(c) For each execution of a method of an RTO.k object, a deadline is imposed;

(d) Real-time data contained in an RTO.k object become invalid after the interval called the maximum validity duration passes.

Triggering times for SpM's must be fully specified as constants during the design time. Those real-time constants appear in the first clause of an SpM specification called the autonomous activation condition (AAC) section. An example of an AAC is

"for t = from 10am to 10:50am every 30min
start-during (t, t+5min) finish-by t+10min"

which has the same effect as

{"start-during (10am, 10:05am) finish-by
start_time+10min",
"start-during (10:30am, 10:35am) finish-by
start_time+10min"}.

An underlying design philosophy of the RTO.k object model is that an RTCS will always take the form of a network of RTO.k objects. RTO.k objects interact via calls by client objects for service methods in server objects. The caller may be an SpM or an SvM in the client object. In order to facilitate highly concurrent operations of client and server objects, non-blocking (sometimes called asynchronous) types of calls (i.e., service requests) can be made to SvM's.

The designer of each RTO.k object provides a guarantee of timely service capabilities of the object by indicating the *deadline for every output* produced by each SvM (and each SpM which may be executed on requests from SvM's) in the specification of the SvM (and some relevant SpM's) advertised to the designers of potential client objects. Before determining the deadline specification, the server object designer must convince himself/herself that with the object execution engine (hardware plus operating system) available, the server object can be implemented to always execute the SvM such that the output action is performed within the deadline. Again, the BCC contributes to major reduction of these burdens imposed on the designer.

3.2 DREAM kernel as an execution engine for RTO.k objects

When the DREAM kernel executes RTO.k objects, it actually executes equivalent programs composed of *processes*, shared data structure monitors called *CREW (concurrent-read exclusive-write) monitors*, and logical

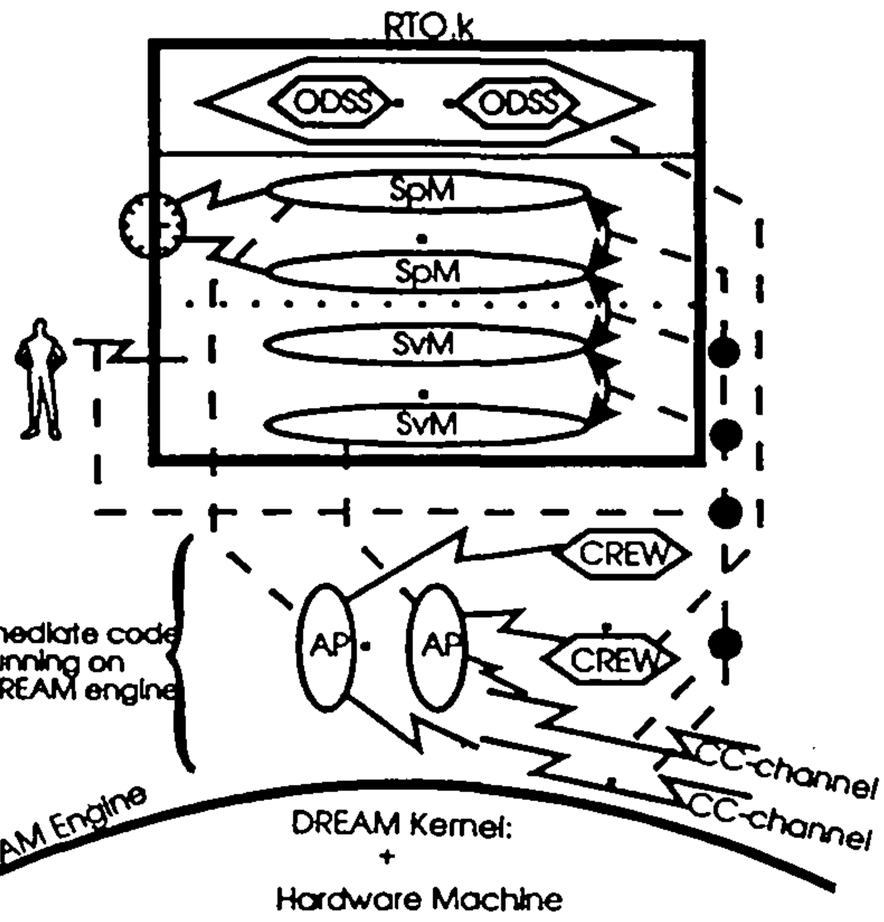


Figure 2. Mapping an RTO.k object to a process-structured-program (Adapted from [Kim95b])

multicast channels called *CC- (content code) channels*. Figure 2 depicts the mapping relationship between components of an RTO.k object and the corresponding components of an equivalent process-structured program. As shown,
(1) object methods, both SpM's and SvM's, are mapped to processes,
(2) *object data space (ODS) segments* (ODSS's) to CREW monitors,
(3) access paths to SvM's to CC-channels, and
(4) result-return paths to clients to CC-channels.
This way of utilizing CC-channels facilitates the transparency of object locations.

When an SpM is mapped to a process, the process must present to the DREAM kernel (1) the AAC of the SpM, (2) the associated deadline specification, and (3) access rights of the SpM for ODSS's. Similarly, when an SvM is mapped to a process, the process must present to the DREAM kernel (1) the associated deadline specification, (2) access rights of the SvM for ODSS's, and (3) other information related to its pipelined execution possibilities.

As mentioned earlier, several prototypes, including the up-to-date version called the DREAM kernel v.D3, was implemented to run on a network of PC's connected by Ethernet. It contains full features of a general-purpose kernel.

4. An RTO.k object model based approach to real-time simulation

The object model was initially formulated and used in simulation applications [Dah72]. Therefore, use of the object model in modeling the environment objects, i.e.,

modular entities in the environment which have time-varying internal states, has been practiced for a long time. However, as mentioned in the preceding section, the RTO.k object model, supports more accurate detailed modeling of environment objects.

4.1 Basic approach

Suppose the application environment chosen is a sky+land+sea segment of interest, called the "theater", and includes moving objects such as ships and airplanes, etc. This environment can be represented and simulated by the RTO.k object in Figure 3.

The object data space (ODS) in this RTO.k object contains state representations of the airplanes, the ships, and the theater space.

Each TT-method, when executed, updates a variable-set in the ODS representing the state of some physical object (i.e., airplane, ship) to reflect the current state of the physical object. Ideally the TT-methods should be *activated continuously* and each of their executions be *completed instantly*. However, the limited power of the execution engine dictates the *activation frequency* of any TT-method, which may be viewed as *the ticking rate of the physical object simulator clock*, to be a fraction of the ticking rate of the real-time clock in the execution engine. Each execution of a TT-method must be completed within one ticking interval of the physical object simulator clock. Therefore, TT-methods are the mechanisms for approximately simulating continuous state changes that occur naturally in the environment objects.

The natural parallelism that exists among the environment objects is precisely represented by use of multiple TT-methods which may be activated simultaneously. In general, the accuracy of an RTO.k object structured simulation of the environment is a direct

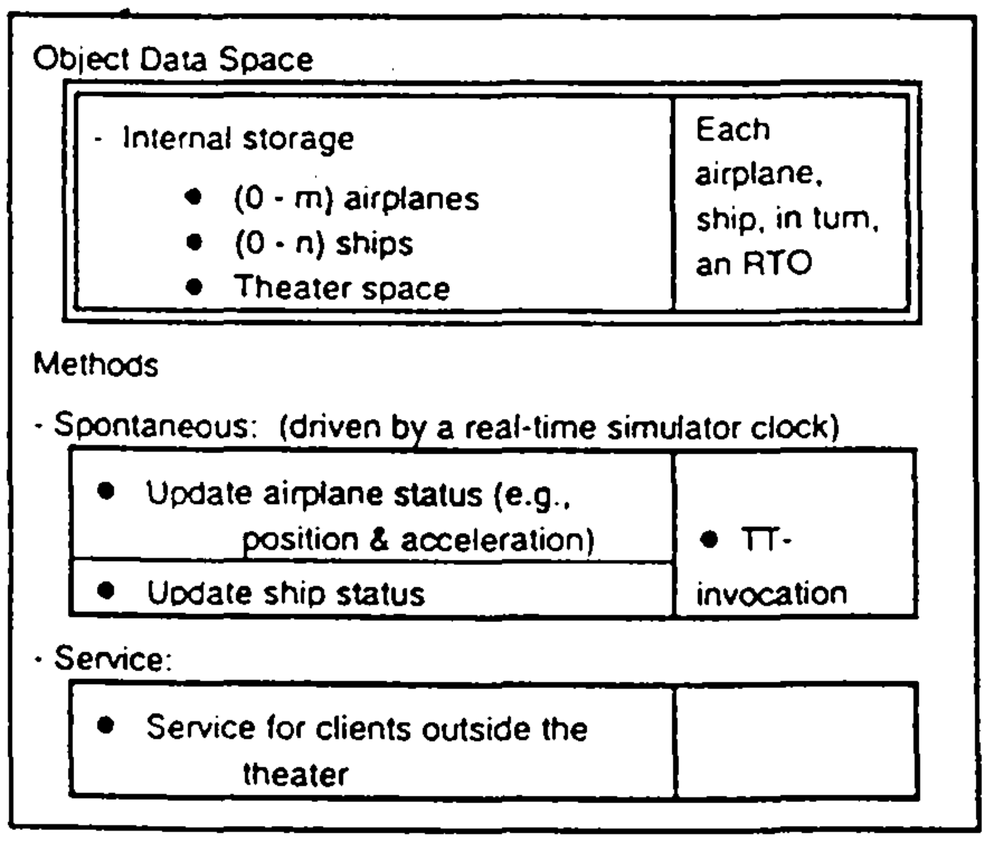


Figure 3. An RTO.k structured simulation model for a theater (Adapted from [Kim94b])

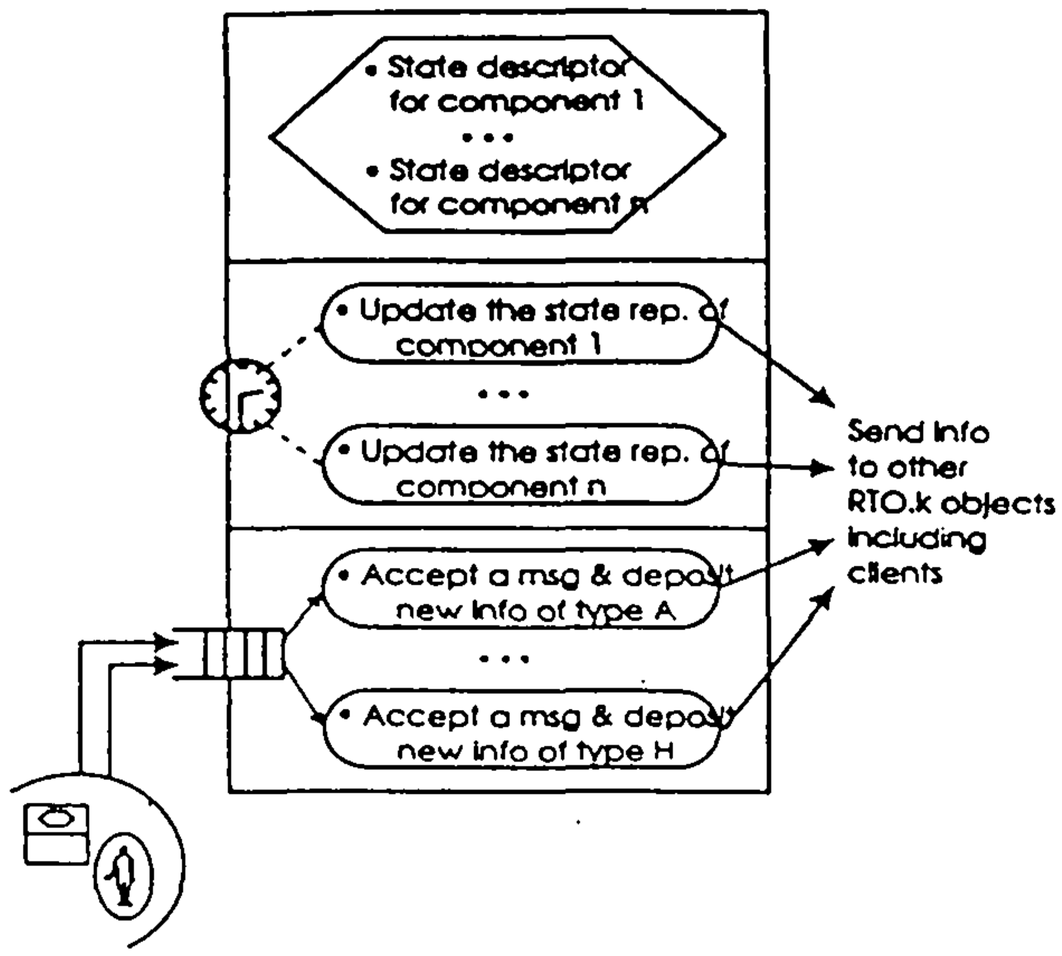


Figure 4. The general structure for the RTO.k object based simulation model

function of the activation frequencies of TT-methods (which are equivalent to the ticking rates of the physical object simulator clocks).

Figure 4 depicts the RTO.k based real-time simulation approach in a generic form.

4.2 Group server and monitor (GSM) object

The single RTO.k representation in Figure 3 can be expanded into a more detailed representation structured in the form of a network of RTO.k objects, each representing

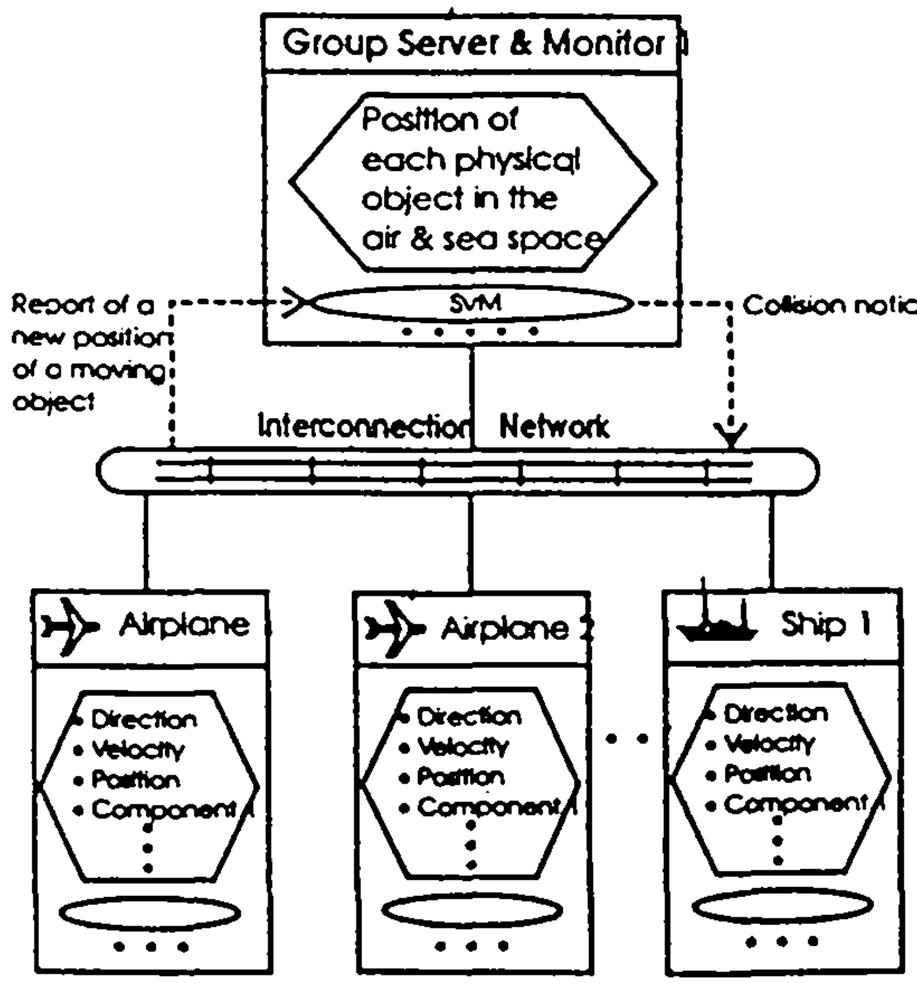


Figure 5. Decomposition of the theater RTO.k in Figure 3.

an airplane or ship, as shown in Figure 5.

Figure 5 also depicts an interesting role played by the RTO.k object representing the air and sea space. This RTO.k object maintains information on how the space is occupied. It facilitates detecting collisions between moving objects such as airplanes, etc. As the RTO.k object structured simulator of an airplane progresses after each simulator clock tick, the position of the airplane is updated and recorded within the RTO.k object. In addition, this RTO.k object notifies the RTO.k object representing the space. The latter RTO.k object in turn checks if the airplane has now collided with any other environment object such as an airplane, ship, etc., and, if so, notifies the RTO.k objects simulating the collided environment objects. The notified RTO.k objects then simulate the post-collision behavior of their simulation targets starting with the following tick of the simulator clock.

In a sense, the RTO.k object representing the space supports the RTO.k objects simulating the moving environment objects. Therefore, it is called a group server and monitor (GSM) RTO.k object. Each ticking interval of the simulator clock must cover the time spent in interaction between a GSM object(s) and monitored RTO.k objects. If the workload for a GSM object becomes too large, then multiple GSM objects, each supporting a different group of RTO.k objects simulating the physical environment objects, can be utilized.

Figure 6 depicts another example of an RTO.k object structured simulator. Automobile traffic in the Los Angeles (LA) - Orange County (OC) area is simulated. The LA district is modeled as one RTO.k object and the OC district is modeled as another. Inside each RTO.k object, dynamically changing states of cars can be traced by one or more SpM's. If the LA area simulator RTO.k object contains three SpM's which are responsible for updating the states of cars in three different segments of the LA area, respectively, and if the RTO.k object runs on a multi-processor execution engine, the three SpM's can be concurrently executed on three different processors.

Cars may move from one district to another. This is simulated by a call of the SpM for the SvM of the

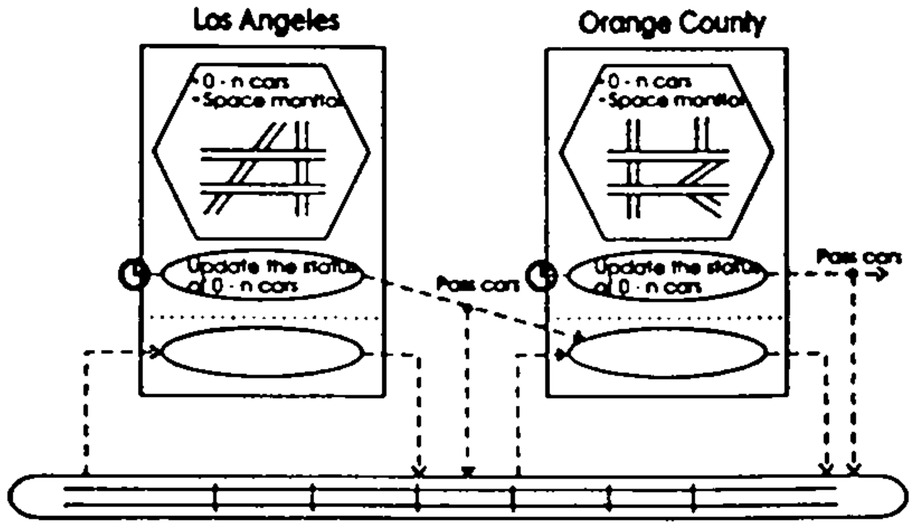


Figure 6. Simulation of cars moving
from one district RTO.k to another district RTO.k

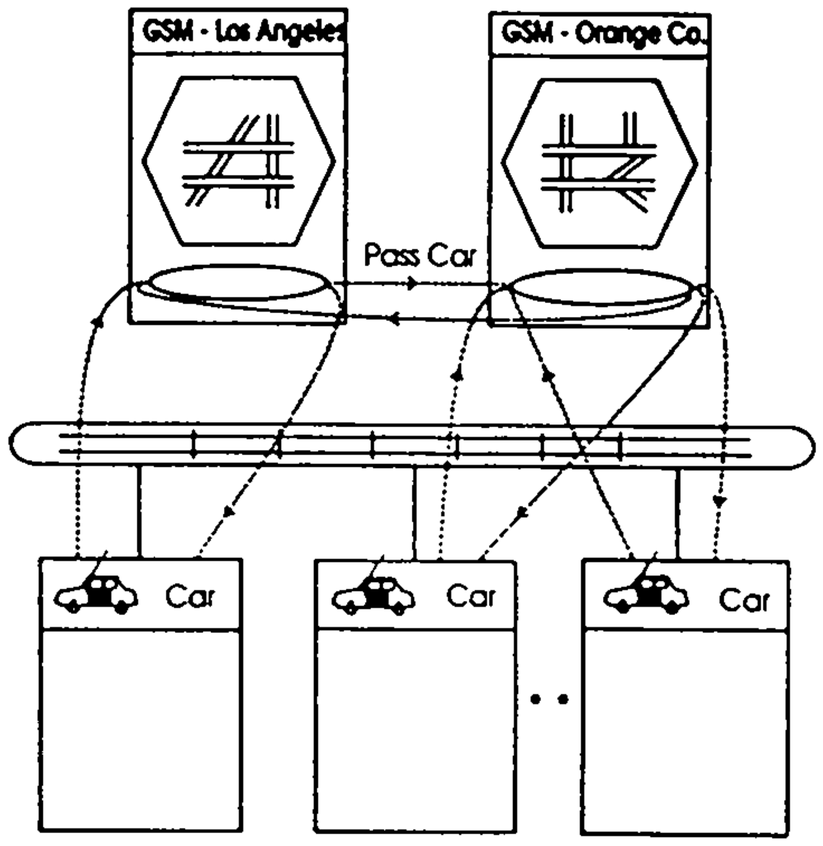


Figure 7. Car RTO's moving
from one district to another district

destination district RTO which will in turn record the state of a newly entering car in the ODS.

Figure 7 depicts a variation of the simulator in Figure 6. Each car is now represented as a separate RTO. Two GSM's track space usage of the two districts, respectively.

4.3 Assessment

As discussed earlier, the RTO.k object model is an effective mechanism not only for variable-degree abstraction of real-time distributed computer systems (DCS's) under design but also for variable-accuracy simulation of the application environments. Structures and notations used will be of the same kind in both control system design and environment simulation. The combination of a control system design and an environment simulator takes the form of a network of RTO.k objects. Therefore, the RTO.k object structuring scheme facilitates uniform structuring of both control computer systems and application environment simulators. This presents considerable potential benefits to the system engineers.

Besides the uniform structuring capabilities, the RTO.k object based real-time simulation approach has some other unique advantages. First, due to the strong modularity characteristics of the RTO.k object, changes in the simulation target can be easily handled while achieving real-time simulation. For example, appearance of a new type of an environment object entails just adding a new RTO.k object without requiring modifications of the other existing RTO.k objects and the real-time simulation capability remains intact. Similarly, changes in some timing or logical behavior of an environment object will cause only some changes inside corresponding RTO.k

objects and again the real-time simulation capability is maintained just with that.

Secondly, in establishing a large-scale simulator, distributed and parallel (D&P) execution of simulation objects is often necessary. Such execution techniques are almost essential tools for realizing full potential benefits of real-time simulation of complex application environments or system designs. Conventional D&P simulation approaches involve message communication among the simulator execution nodes for the purpose of keeping them synchronized [Fuj90, Mis86].

In one type of D&P approaches, the processing nodes progress in parallel in a lock-step mode. They all advance to the next simulation step after ensuring that all nodes have completed the current simulation step. In large-scale real-time simulation, this message exchange overhead can be substantial since every node must generate a completion report message. It leads to a large ticking interval of the simulator clock, which in turn leads to inefficient real-time simulation.

In another type of D&P approaches, they advance somewhat asynchronously but as they often discover after exchanging messages that they have executed some incorrect simulation steps, they roll back to undo those steps. These rollback approaches also causes the ticking interval of the simulator clock to become very large, which may result in highly inefficient real-time simulation.

In contrast, no synchronization messages need to be exchanged in D&P execution of RTO.k simulator objects. This is because each simulation step with an RTO.k simulator object is triggered by a real-time clock. Therefore, as long as the triggering interval in each SpM (i.e., the inverse of the state updating frequency) in an RTO.k simulator object is chosen to be sufficiently long to cover the execution of a simulation step by any processing node, the nodes can safely start the next step simultaneously without exchanging synchronization messages. This can be more efficient in large-scale simulations which tax the power of a large-scale D&P processing system.

Thirdly, as mentioned earlier, when we simulate both the application environment and a computer system design together, we may choose to scale up or down uniformly in the time dimension both the environment simulator and the computer system simulator, depending upon the power of the simulation execution engine available. For example, by the simple action of reducing the activation frequency of each SpM without changing the body of the SpM, we can reduce the load presented to the overloaded execution engine. Such down-scaled real-time simulation, which requires a lower-power execution engine, can still serve most of the originally intended objectives of closed loop real-time simulators since the order of events and the ratio of the duration of one activity over that of another are still accurately exhibited.

5. Experiments conducted

As mentioned in the introduction, the RTO.k object model was partially validated through two specification, design, and real-time simulation experiments conducted recently:
(1) An experiment that involved an application of the RTO.k structuring scheme to both the development of a defense system and that of an environment simulator and
(2) An experiment that involved an application of the RTO.k structuring scheme to both the development of a freeway traffic control system and that of a freeway traffic simulator.

In the experiment based on the defense application scenario, the environment considered was a much more complicated version of the theater shown in Figure 3. In this environment, there were hostile flying objects, radars, and interceptors (both on the land and on ships) in addition to the airplanes and ships. Control computer systems were used both on the land and on ships.

In the experiment based on the advanced freeway traffic control scenario, electro-magnetic loop detectors (which detect cars moving over the loops) and ramp meters were considered in addition to freeway lanes. Car accidents were also dealt with. Control computer systems were used to collect the loop detector data and control ramp meters under heavy traffic conditions.

In both experiments, each entire application environment was treated as a single RTO.k object initially. Then the single RTO.k object specification was gradually refined into a network of RTO.k object specifications, each corresponding to a different environment object or a computing object. RTO.k objects were then distributed among multiple PC's running the DREAM kernel and exchanging messages over an Ethernet. Some PC's were didicated to graphic display functions.

These experiments reinforced our belief that the RTO.k model had the necessary representational power and also offered an efficient and rigorous way to develop complex real-time systems and their application environment simulators. Although the experiments were conducted on a local area network of PC's rather than on a sizable parallel computer, the benefits mentioned in Section 4.3 could still be confirmed by and large. Some real-time fault tolerance techniques were also incorporated into these experimental control computer systems.

6. Future directions

The real-time simulation can play useful roles in many phases of the complex system engineering cycle, in particular, validation and evaluation of control computer system designs as they evolve through abstract designs to detailed implementations. Not only logical behavior but also timely performance and dependability characteristics

can be effectively validated and evaluated via real-time simulation.

The RTO.k object based approach to real-time simulation has been presented in this paper. The RTO.k object is capable of uniformly and accurately representing both real-time embedded computer systems and application environments. The RTO.k object based approach to real-time simulation has many attractive features, e.g., broad applicability, expandability and modifiability, adaptability for efficient parallel processing, etc. Much further work is needed to develop supporting tools for the RTO.k object based real-time simulation. First of all, a high-level language tool for RTO.k object structuring which is more abstract in nature than the language C++ combined with an RTO.k object component library, e.g., the DREAM library in [Kim96], needs to be established along with a supporting translator. We are currently developing an extension of C++, which we have named C++T. Secondly, an efficient execution engine, especially one capable of effectively taking advantage of the raw processing power of highly parallel machines, needs to be developed. Adapting the DREAM kernel to a highly parallel machine such as Intel Corp.'s Paragon, is considered to be a worthwhile research and development effort.

One important feature that is essential for a real-time simulation system is visualization. Ideally, real-time display of the dynamically changing states of the simulation targets should not fall behind the real-time simulation beyond a certain fixed time interval. Visualization of RTO.k objects is thus another meaningful topic for future research and development.

Acknowledgment: The research work reported here was supported in part by US Navy, NSWC Dahlgren Division under Contract No. N60921-92-C-0204, in part by the California Transportation Department via the UCI ITS, in part by the University of California MICRO Program under Grant No. 93-080, in part by Hitachi, Ltd, and in part by Postech.

References

[Dah72] Dahl, O.J., "Hierarchical Program Structuring", in Dahl, Dijkstra, & Hoare eds., '*Structured Programming*', Acad. Press, NY, 1972.

[Ell93] Ellenberger, R., Ling, R., Buscher, D., Uhde-Lacovara, J., and Shuler, R., "Automatic Generation of Real-Time Ada Simulations for Space Station Freedom", *Simulation*, Nov.1993, pp.337-345.

[Fuj90] Fujimoto, R.M., "Parallel Discrete Event Simulation", Communications of the ACM, Vol. 33, No. 10, Oct. 1990, pp.30-53.

[Guy94] Guyse, C., Buscher, D., and Ellenberger, R., "Real-time Environment and Vehicle Dynamics Simulations for Space Station Freedom Integrated Test and Verification Environment", *Simulation*, Apr.1994, pp.230-239.

[Kim94a] Kim, K.H. et al., "Distinguishing Features and Potential Roles of the RTO.k Object Model", *Proc. 1994 IEEE CS Workshop on Object-oriented Real-time Dependable Systems (WORDS)*, Oct. '94; Dana Point, pp.36-45.

[Kim94b] Kim, K.H. and Kopetz, H., "A Real-Time Object Model RTO.k and an Experimental Investigation of Its Potentials", *Proc. 1994 IEEE CS Computer Software and Applications Conf. (COMPSAC)*, Nov. 1994, Taipei, pp.392-402.

[Kim95a] Kim, K.H., "Toward New-Generation Real-Time Object-Oriented Computing", *Proc. IEEE CS 5th Workshop on Future Trends of Distributed Computing Systems (FTDCS)*, Cheju Island, Aug. '95, pp.520-529.

[Kim95b] Kim, K.H. et al., "A Timeliness-Guaranteed Kernel Model -- DREAM Kernel -- and Implementation Techniques ", *Proc. RTCSA '95 (1995 Int'l Workshop on Real-Time Computing Systems & Applications)*, Tokyo, Oct. '95, pp.80-87.

[Kim96] Kim, K.H., Subbaraman, C., and Kim, Y.S., "The DREAM Library Support for PCD and RTO.k programming in C++", *Proc. 1996 IEEE CS Workshop on Object-oriented Real-time Dependable Systems (WORDS)*, Feb. '96, Laguna Beach, pp.59-68.

[Mis86] Misra, J., "Distributed Discrete-Event Simulation", ACM Computing Surveys, Vol. 18, No. 1, March 1986, pp.39-65.

[Zei93] Zeigler, B., and Kim, J., "Extending the DEVS-Scheme Knowledge-Based Simulation Environment for Real-Time Event-Based Control", *IEEE Trans. on Robotics and Automation*, Vol.9, No.3, 6/93, pp.351-356.

The DREAM Library Support for PCD and RTO.k programming in C++

K. H. (Kane) Kim, Chittur Subbaraman, and Yuseok Kim
Dept. of Electrical and Computer Engineering
University of California, Irvine

Abstract: In order to realize real-time computing in the form of a generalization of non-real-time computing and yet allowing system engineers to confidently produce certifiable RTCS's for safety-critical applications, we have established a real-time object-oriented (OO) structuring approach called the RTO.k object structuring scheme in recent years. Also, a model of an operating system kernel which can support both conventional real-time processes and RTO.k objects with guaranteed timely services, has been formulated. The model has been named the DREAM kernel. The DREAM kernel supports three powerful building blocks of process-structured real-time distributed programs: real-time process, concurrent-read-&-exclusive-write (CREW) shared data structure monitor for intra-node inter-process communication, and data field channels (DFC's) which are real-time logical multicast channels for inter-node and intra-node inter-process-group communication. Programs composed of these components are called PCD programs. A DREAM library that consists of a collection of specific C++ classes provides a user-friendly interface to DREAM kernel services for real-time application programs. An implemented prototype of the DREAM kernel encapsulated by the DREAM library supports not only well-structured PCD programming but also efficient programming of RTO.k objects in C++. The essence of PCD and RTO.k object programming in C++ with the aid of the DREAM library is discussed.

Index terms: GT design, DREAM library, C++ PCD programming, C++ RTO.k programming

1. Introduction

A few years ago we became convinced that it was time to start vigorously pursuing the following idealistic real-time computing paradigms [Kim95b]:

(1) *General-form design style*: Future real-time computing must be realized in the form of a generalization of the non-real-time computing, rather than in a form looking like an esoteric specialization.

(2) *Design-time guarantee of timely service capabilities of subsystems*: To meet the demands of the general public on the assured reliability of future real-time computing systems (RTCS's) in safety-critical applications, it is inevitable to require the system engineer to produce design-time guarantees for timely service capabilities of various subsystems (which will take the form of objects in object-oriented system designs).

The motivating factors behind these paradigms which may be called the GT (General-form Timeliness-guaranteed) design are the newly improved hardware economy and component reliability which provide impetus in expanding the real-time computing application field.

To realize the idealistic GT design, a powerful structuring scheme capable of dealing with all practically useful real-time and non-real-time computing requirements must be established. In the last several years, there has been a growing trend of research activities aimed for extending the conventional OO-structuring approaches to support RTCS design [Att91, Ish92, Kim94b, Kop90, Tak92]. However, most of those works have not been aimed for supporting the design-time guarantee of timely service capabilities of objects, which is one of the fundamental parts of GT design. In recent years, the first co-author and his collaborating researchers together have established a real-time OO-structuring approach with the purpose of facilitating GT design. The new object structuring approach is called the RTO.k object structuring scheme [Kim94a, Kim94b]. The RTO.k object structuring scheme, though an extension of the conventional OO-structuring approaches, has several unique features of fundamental nature which will be reviewed later in Section 2.

Until a highly abstract programming language for RTO.k object structuring and efficient execution facilities for such objects become available, the most cost-effective approach to practicing the RTO.k object structuring appears to be to program RTO.k objects by use of conventional concurrent and distributed process structuring languages supported by real-time operating systems. The first co-author recently formulated a model of an operating system kernel which can support real-time processes with guaranteed timely services. The model is named the DREAM (Distributed Real-time Ever Available Micro-computing) kernel. The DREAM kernel supports three powerful building blocks of process-structured real-time distributed programs:
(1) real-time processes subject to various activation and synchronization requirements,
(2) shared data structure monitors called concurrent-read-&-exclusive-write (CREW) monitors &,
(3) real-time multicast logical (RML-) channels called data field channels (DFC's).
These three basic components fully represent the general facilities for concurrent and distributed program structuring. Programs composed of these three components are called PCD (process-CREW-DFC)

programs. In fact, the DREAM kernel can support both process-structured real-time application software (i.e., PCD programs) and RTO.k object structured application software. Components of an RTO.k object are treated as special types of PCD components. The key emphasis in formulating the DREAM kernel was in realization of guaranteed timely service capabilities with minimal loss of hardware utilization. The DREAM kernel can thus be viewed as a model of a general-purpose timeliness-guaranteed OS kernel.

In addition, the authors have designed a library of functions that eases not only PCD programming but also RTO.k object programming in C++. This library called the DREAM Library is a collection of several C++ classes, each class functioning as an interface between a PCD or RTO.k object program and a specific DREAM kernel service call (KSC) group. The DREAM library hides various details of parameter passing between application and the DREAM kernel and thus offers a user-friendly interface to the application programmer. An implemented prototype of the DREAM kernel encapsulated by the DREAM library supports not only well-structured PCD programming but also efficient programming of RTO.k objects in C++.

Several prototype implementations of the DREAM kernel, have been produced by the authors to run on networks of PC's connected by Ethernet. An earlier prototype kernel has been used to run an RTO.k structured non-trivial defense C^3 application, together with an RTO.k structured real-time simulator of the application environment.

The paper starts in Section 2 with a discussion on the essence of the RTO.k object structuring scheme. The major features of the DREAM kernel offered to application programmers are briefly presented in Section 3. The essence of PCD program structuring in C++ with the support of the DREAM library is discussed in Section 4. Section 5 then discusses the programming of RTO.k objects in C++ with the support of the DREAM library. The next section, Section 6, looks at the various advantages and limitations of the PCD based structuring scheme for RTO.k objects. The paper concludes in Section 7 with discussions on the issues to be resolved via future research.

2. Overview of the RTO.k Object Structuring Scheme

An initial abstract framework of the *RTO.k object model*, also called the time-triggered real-time object (TT-RTO) model, came out of the attempt by the first co-author and Hermann Kopetz at the Technical University of Vienna to find a proper extension of the basic object model which is highly cost-effective in development of *hard-real-time* application systems. Based on the initial abstract framework formulated in late 1980's [Kop90], a

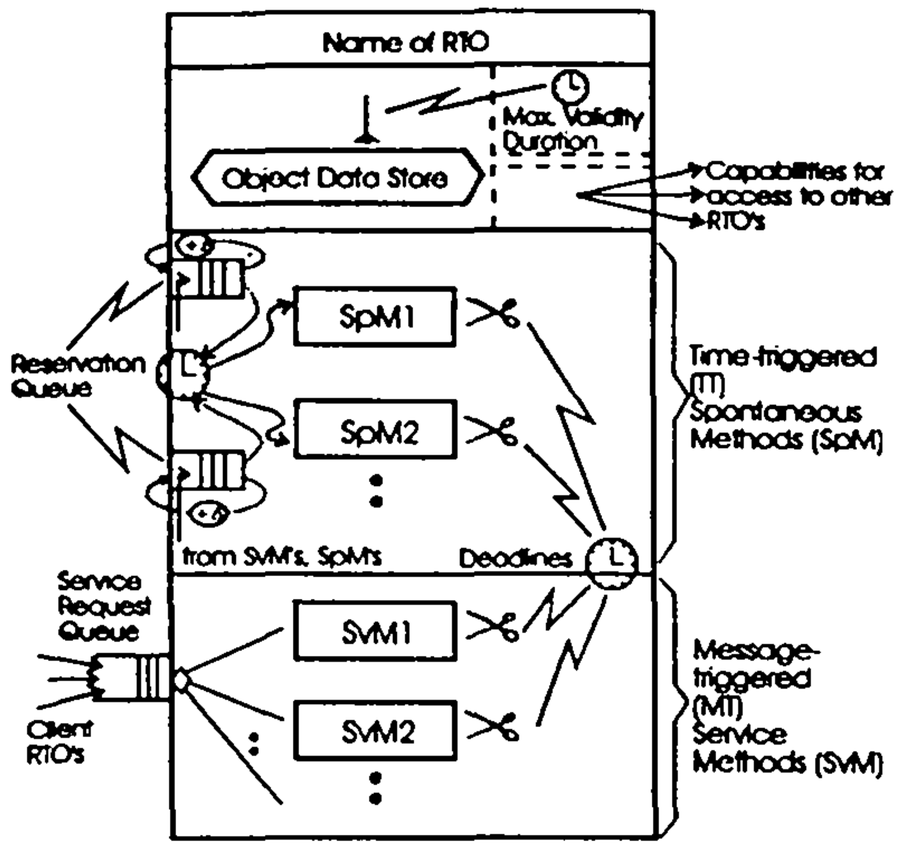


Figure 1. Structure of the RTO.k Object Model (Adapted from [Kim94b])

concrete syntactic structure and execution semantics was developed in recent years [Kim94a, Kim94b].

The basic structure of an RTO.k object is depicted in Figure 1. It is an extension of the conventional object model(s) and four most important extensions are the following:

(a) *Two clearly separated groups of methods*:

For some methods of an RTO.k object, a real-time clock serves as the mechanism for triggering the method executions as the clock reaches some values specified at design time and such methods are called time-triggered (TT-) methods, also called the spontaneous methods (SpM's), and *clearly* separated from the conventional service methods (SvM's) triggered by messages from clients. The two types of methods in an RTO.k object are different not only in the way their executions are triggered but also in that "actions to be taken at real times *which can be determined at the design time* can appear only in SpM's".

Therefore, actions of the type "at constant-clock-value do S" or the type "sleep-until constant-clock-value" can appear only in SpM's. Incorporation of SpM's means introducing the potential for the following two new types of concurrent executions of object methods in addition to the potential for concurrent executions of SvM's that exist in conventional objects:

(Type I) Concurrency among SpM executions: This concurrency is specified in an implicit but natural manner, e.g., two SpM's designed to be triggered at 10 am.

(Type II) Concurrency between SpM executions and SvM executions.

(b) *Basic concurrency constraint (BCC)*:

In order to dramatically reduce the designer's efforts in guaranteeing timely service capabilities of

RTO.k objects, the execution rule which prevents conflicts between SpM's and SvM's is incorporated. Basically, *activation of an SvM triggered by a message from an external client is allowed only when potentially conflicting SpM executions are not in place.* To be exact, when a message-triggered SvM is not free of conflict with an SpM in accessing the same portion of the object data space (ODS), execution of the former method (SvM) must not be allowed in a time zone earmarked for a TT-execution of the latter method (SpM). This restriction is called the basic concurrency constraint (BCC). Therefore, SpM's are given higher priorities for execution over the SvM's. Note that this BCC does not impose any restriction on concurrent execution of SpM's or concurrent execution of SvM's. Therefore, executions of SpM's are not disturbed by SvM executions and triggering times of SpM's are fixed at the design time. At least this makes it very easy to analyze the execution time behavior of SpM's. For example, if a statement of the type "at 10am do S" appears in an SpM, its reliable execution can be easily assured.

(c) For each execution of a method of an RTO.k object, a *deadline* is imposed.

(d) Real-time data contained in an RTO.k object become invalid after the interval called the maximum validity duration passes.

The first two features (a) and (b) mentioned above make the RTO.k object model clearly distinguished from other proposed real-time object models [Att91,Ish92, Tak92].

Triggering times for SpM's must be fully specified as constants during the design time. Those real-time constants appear in the first clause of an SpM specification called the autonomous activation condition (AAC) section. An example of an AAC is

> "for t = from 10am to 10:50am every 30min
> start-during (t, t+5min) finish-by t+10min"
>
> which has the same effect as
>
> {"start-during (10am, 10:05am)
> finish-by 10:10am",
> "start-during (10:30am, 10:35am)
> finish-by 10:40am"}.

A provision is also made for making the AAC section of an SpM contain only candidate triggering times, not actual triggering times, so that a subset of the candidate triggering times indicated in the AAC section may be dynamically chosen for actual triggering. Such a dynamic selection occurs when an SvM within the same RTO.k object requests future executions of a specific SpM. The AAC specifying candidate triggering times rather than actual triggering times starts with a declaration "if-demanded".

An underlying design philosophy of the RTO.k object model is that an RTCS will always take the form of a network of RTO.k objects. RTO.k objects interact via calls by client objects for service methods in server objects. The caller may be an SpM or an SvM in the client object. In order to facilitate highly concurrent operations of client and server objects, non-blocking (sometimes called asynchronous) types of calls (i.e., service requests) can be made to SvM's.

The designer of each RTO.k object provides a guarantee of timely service capabilities of the object by indicating the *deadline for every output* produced by each SvM (and each SpM which may be executed on requests from SvM's) in the specification of the SvM (and some relevant SpM's) advertised to the designers of potential client objects. Before determining the deadline specification, the server object designer must convince himself/herself that with the object execution engine (hardware plus operating system) available, the server object can be implemented to always execute the SvM such that the output action is performed within the deadline. Again, the BCC contributes to major reduction of these burdens imposed on the designer.

The RTO.k object model is effective not only in the multiple-level abstraction of real-time (computer) control systems under design but also in the accurate representation and simulation of the application environments. In fact, it enables uniform structuring of control computer systems and application environment simulators [Kim94a] and this presents considerable potential benefits to the system engineers.

3. Major Features of the DREAM Kernel

An essential building-block for construction of timeliness-guaranteed real-time application systems is a *timeliness-guaranteed operating system* which together with the hardware platform forms an execution engine that provides guaranteed timely services to concurrent and distributed real-time application software. If the operating system lacks such guaranteed timely service capabilities, then timely service capabilities of real-time applications cannot be ensured. Existing commercial operating systems lack the capability for providing guaranteed timely services required by a great deal of real-time application software. The DREAM kernel has been formulated as a model of an operating system kernel that can support both PCD-based real-time processes and RTO.k objects with guaranteed timely services. An experimental prototype of the DREAM kernel was implemented in the DREAM laboratory of UC Irvine. The experimental version runs on a network of PC's connected via Ethernet and equipped with i486 or Pentium processors, DOS-BIOS device drivers, and the Packet Ethernet driver. This section briefly discusses some major features of the DREAM kernel facility offered to real-time C++ PCD and C++ RTO.k program designers.

3.1 Guaranteed worst-case response

One of the most distinguishing features of the DREAM kernel is that it provides guaranteed timely services to real-time applications with minimal loss of hardware utilization.

The guaranteed timely service capabilities of the DREAM kernel is realized by adopting a unique approach for the layering of its components [Kim95b]. The kernel consists of five layers L0 through L4 in total.

This approach is based on the organizational principle called the time-leasing machine layering, which is of fundamental nature. Under the time-leasing machine layering principle, the bottom layer (L0 in the DREAM kernel) owns the full power of the hardware machine. So, the bottom layer uses the hardware machine at its own will. The remainder of the hardware machine time after the bottom layer use of the hardware machine, is "leased" to the next upper layer (L1 in the DREAM kernel). The time leasing relationship is recursive, i.e., it is maintained through all the layers that compose the kernel (in the DREAM kernel layers L0-L4). Moreover, there is a tight bound on the amount of machine time that each layer uses. This sometimes implies a bound on the frequency of certain types of interrupts that are accepted.

3.2 Exploitation of parallelism within the kernel by use of kernel-threads

In order to maintain a high degree of hardware utilization, parallelism is exploited extensively within the DREAM kernel through the introduction of kernel-threads. The kernel-thread, or thread for short, is an active concurrency unit operating inside the DREAM kernel. The set of threads is fixed at the operating system loading time. All the threads share the same address space and they are strictly *periodic threads* which make the analysis of their worst-case execution behavior a trivial task. Currently, four basic kernel-threads are used in the DREAM kernel. Additional application-specific custom kernel-threads can be introduced if fast response activities specific to the application must be supported.

4. DREAM Library and DREAM Kernel Support for PCD Programming in C++

As mentioned in Section 1, the DREAM kernel supports the execution of both process-structured and RTO.k object structured real-time application software. This section focuses on the construction of real-time PCD programs in C++ and the support offered by the DREAM library and DREAM kernel.

4.1 Basic components of process-structured real-time concurrent and distributed PCD programs

In Section 1, we briefly mentioned the PCD components. Let us now consider each of these components in some detail.

(1) Processes: These are real-time processes with various synchronization and activation requirements. In this paper, these are also called *application processes* (AP's) although in reality, some of these processes may play the roles of system management processes, e.g., processes managing I/O. The processes may frequently impose deadlines on the kernel for execution of their next computation-segments.

(2) CREW (concurrent-read-&-exclusive-write) monitors: The CREW monitor is a shared data structure monitor and an extension of the *monitor* defined in [Bri77] in that the former possesses the *readers-writers* semantics (i.e., concurrent-read-&-exclusive-write semantics) instead of the exclusive-read-&-exclusive-write semantics associated with the latter.

(3) Data field channels (DFC's) in the HU-DF scheme [Kim95a]: The *HU-DF (HU data field) scheme* for inter-process-group communication is an extension of the original *data field* scheme developed by Mori and other researchers in Hitachi, Ltd. [Mor93]. The essence of the data field scheme is to facilitate dynamic creation of *logical multicast channels* called data field channels (DFC's) and dynamic connection of processes to the DFC's in such a way that the idiosyncrasies of the physical communication networks are transparent to the process designer. If the physical communication facility has the broadcast capability, then a logical multicast channel is facilitated by making all processing nodes using the channel to broadcast (through the physical communication facility) messages with the headers containing the ID of the channel called the content code. The processing nodes connected to the logical channel can see all the messages coming through the physical broadcast facility but will pay attention only to those messages containing relevant content codes. The HU-DF scheme differs from the original data field scheme in that the former allows dynamic flexible connection of processes to the logical channels and supports not only conventional *event messages* but also *state messages* which are based on the distributed replicated memory semantics (a variation of the state message semantics in [Kop89]).

The three basic components (process, CREW and DFC's) represent fully general facilities for concurrent and distributed program structuring. Therefore, any operating system kernel that supports these three components along with various I/O operations can be viewed as a general purpose kernel.

In addition to these three program components, a PCD program may also use semaphores for inter-process communication and synchronization. Also, a PCD program may use ports for inter-process communication apart from the DFC's [Ras89]. A port is a message communication channel with only a single AP as receiver. An AP may notify its intention to the underlying kernel to function as a receiver to a specific port. In such a case, the kernel makes sure that no other

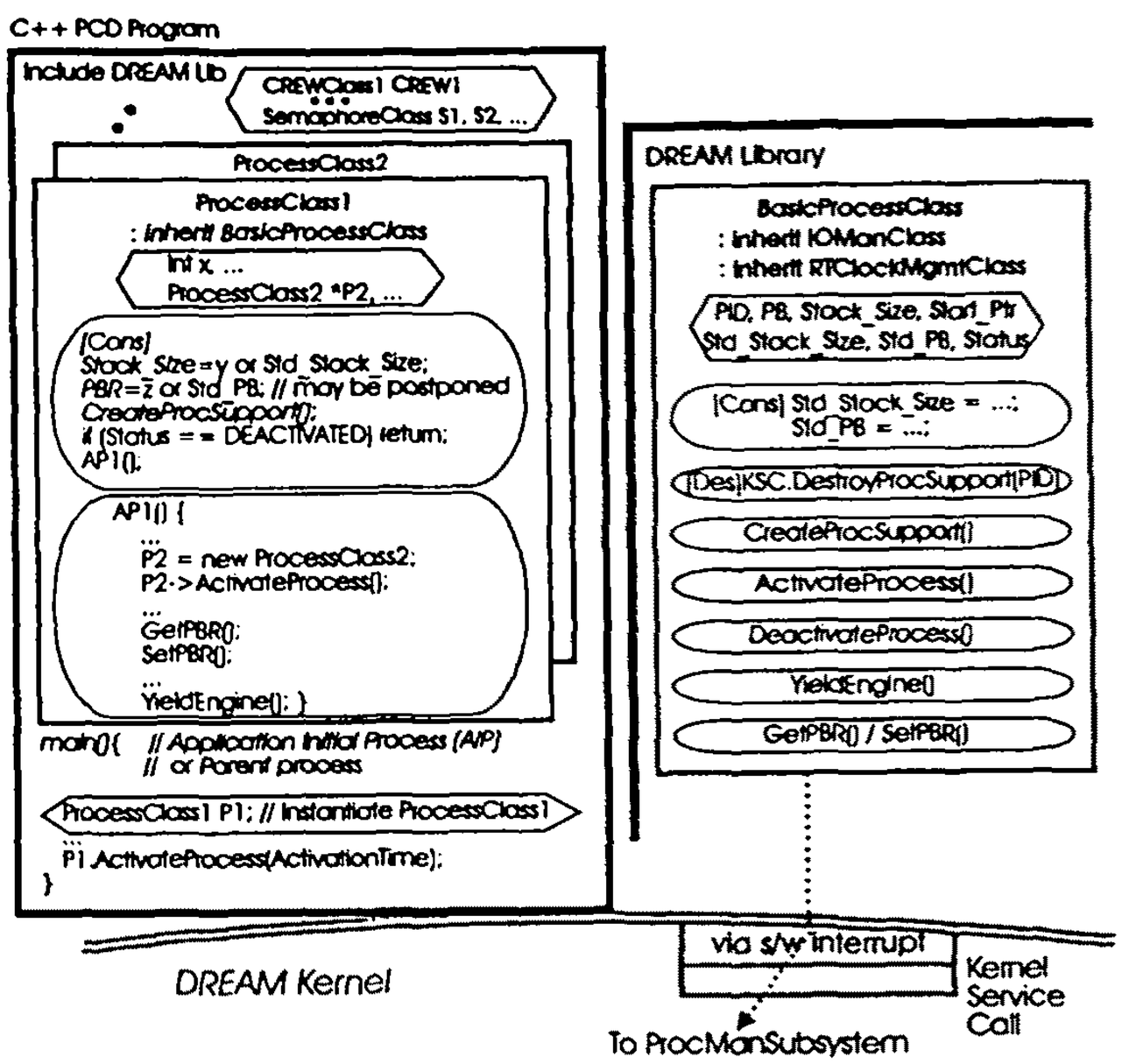


Figure 2. An application process constructed with DREAM library support

AP's are granted the receive rights to that port. The DFC based communication facility is more powerful than the port based communication facility since the former supports group communication whereas the latter supports only point-to-point communication.

4.2 Construction of C++ PCD programs with the support of the DREAM Library

Application programmers may construct each PCD program as a collection of C++ objects and the main function which is treated as the body of the special process called the Application Initial Process (AIP). The PCD program may request the services of the DREAM kernel during its execution by making kernel service calls (KSC's) to the DREAM kernel. An interface to the DREAM kernel which is friendly to the PCD programmer has been constructed. This interface is called the DREAM library. The DREAM library is a collection of several C++ classes, each class functioning as an interface between a PCD program and a specific KSC group. For instance, the *BasicProcessClass* of the DREAM library provides a PCD program with the interface to KSC's related to AP management. A DREAM library class houses a set of pre-defined data structures and member functions all of which are well-known to the PCD programmer. In order to pass parameters to the DREAM kernel, in most cases the PCD programmer just needs to initialize the appropriate data structure(s) of the pertinent DREAM library class and then call the appropriate member function of this class. This member function is responsible for packaging the input parameters supplied by the PCD program, making the correct KSC, and possibly receiving output parameters from the DREAM kernel and saving it in the appropriate data structure defined in the class. The PCD program can then retrieve the appropriate output parameters.

Figure 2 shows the construction of an AP in a PCD program. The user first defines a *ProcessClass* which *inherits* the *BasicProcessClass* of the DREAM library. The BasicProcessClass provides a user-friendly interface to the programmer wishing to make KSC's related to AP management. The AP management KSC's include creating and destroying the kernel support for an AP, activating an AP, deactivating an AP, obtaining and setting priority values for the AP, etc. The BasicProcessClass defines default values for the process stack size and priority. The user may use the default value for the stack size and process priority or may redefine these values. Once the application-specific ProcessClass is defined, the user creates an instance of a ProcessClass inside the AIP. The AIP is the starting point of the entire application and is responsible for initializing the various child processes located in the node. When an instance of a ProcessClass is created, kernel support for this AP is automatically created since the constructor of the ProcessClass contains the appropriate DREAM library call (*CreateProcSupport*). Since the ProcessClass inherits the BasicProcessClass, it can access the capabilities provided by the BasicProcessClass. Figure 2 also shows how parameters are passed from the PCD program to the DREAM library.

In addition to the AP class, the user may also define an AP group class that encapsulates a group of communicating AP's within a node. The AP's in the AP group class would use semaphores and/or CREW monitors to communicate with each other.

In order to construct a CREW monitor, the PCD programmer typically defines a *CREWClass* which inherits the *CREWMgtClass* provided by the DREAM Library. The CREWMgtClass provides the PCD programmer with a user-friendly interface to KSC's for creating and destroying CREW support and obtaining and releasing READ-WRITE or READ-ONLY access rights to the CREW monitor. When an instance of a CREWClass is created, kernel support for the CREW monitor is automatically created since the constructor of the CREWMgtClass (inherited by the CREWClass) makes the appropriate KSC.

The DFC's provide an alternate means of communication among AP's within a local node and the main means of communication among a group of AP's located in different nodes distributed in a network such as a Local Area Network (LAN). For highly efficient communication, AP's within a local node are usually designed to use CREW monitors. However, this would prevent efficient migration of these AP's to other nodes. In order to design an AP in a PCD program to use the DFC facilities for sending and receiving event messages the designer may create an instance of the *DFCClass* provided by the DREAM library. The DFCClass provides an interface to KSC's such as creating and destroying DFC support, connecting an AP to and disconnecting an AP from a DFC, sending, forwarding, and receiving event messages via DFC, etc. When an AP connects itself to a DFC, it basically joins the multicast group of AP's which use that DFC for multicasting. An AP may also send and read state messages [Kim95a] which are shared among the AP's connected to the same DFC. State messages contain up-to-date information about the state of some entity in the system, for instance, the temperature of a room. The DREAM library offers support for the creation, sending and receiving of state messages through DFC's, etc.

As we mentioned in the previous subsection, a PCD program may also use semaphores and ports for interprocess communication. Similar to the DFCClass, the DREAM library provides a *PortClass* and a *SemaphoreClass* with the appropriate data structures and member functions.

4.3 Real-time processes and scheduling

The DREAM kernel supports the execution of real-time application processes (AP's) with various synchronization and activation requirements. The AP's may frequently impose deadlines on the kernel for execution of their next computation-segments. The AP management subsystem of the DREAM kernel facilitates the static and dynamic creation and destruction of AP's, static and dynamic setting of process priority values, voluntary yield of the execution engine by an AP, etc.

The DREAM kernel provides a flexible platform to easily test out a variety of real-time AP scheduling policies. In the current prototype version of the DREAM kernel, v.D2, the real-time AP scheduling policy incorporated is the *priority bracket (PB) scheduling* policy. The PB scheduling policy is quite powerful in that it can simulate a variety of other scheduling policies such as round robin, rate monotonic and other fixed priority scheduling policies, deadline-driven scheduling policies, etc. Nevertheless, the priority bracket scheduling policy is just one of many AP scheduling policies that can be incorporated into the DREAM kernel.

5. DREAM Library and DREAM Kernel Support for RTO.k Programming in C++

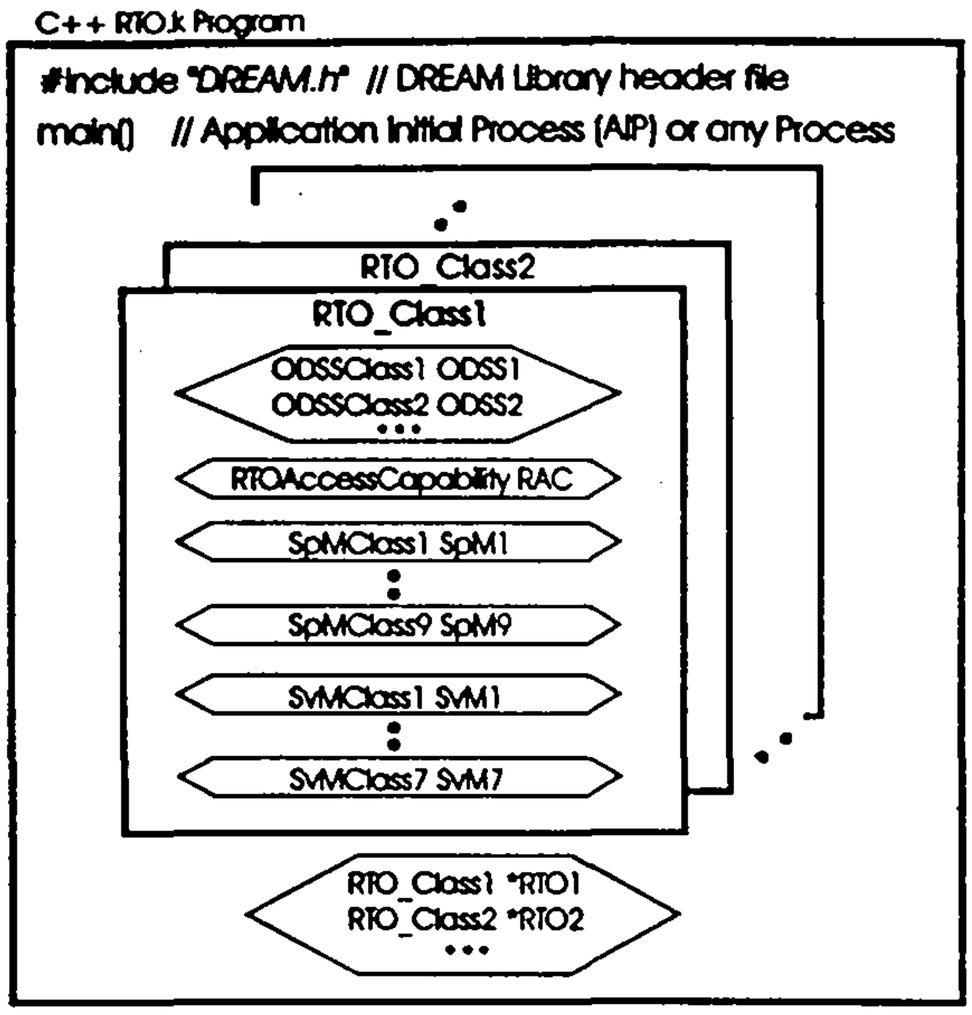


Figure 3. The definition of RTO class

In the previous section, we have seen the construction of C++ PCD programs with the DREAM library and DREAM kernel support. We now discuss the construction of C++ RTO.k programs with the DREAM library and DREAM kernel support.

5.1 Definition of an RTO.k object

As discussed in section 4.2, the application program begins execution from the AIP. The C++ RTO.k programmer typically defines an RTO.k class as a C++ class and creates an instance of the RTO.k class inside the AIP. Figure 3 shows such RTO.k classes defined as C++ classes. In this figure, the *RTO_Class1* represents one RTO.k class. It consists of
(1) instances of user-defined *ODSSClasses* which represent the various ODSS's of the RTO.k object,
(2) an instance of a user-defined *RTOAccessCapabilityClass*; Client RTO's need to obtain capabilities for accessing server RTO's before requesting for service. The access capability will be provided to a client RTO in the form of the service request queue (SRQ) ID that should be used to request for service. The RTOAccessCapabilityClass facilitates a client RTO to obtain capabilities for accessing server RTO's, and
(3) instances of user-defined *SpMClasses* and *SvMClasses*. These classes represent the SpM's and SvM's of this RTO.k object.

In subsequent subsections, we will describe the details on the user-defined classes mentioned in this subsection.

5.2 Overview of the approach for composing an RTO.k object by using PCD program components

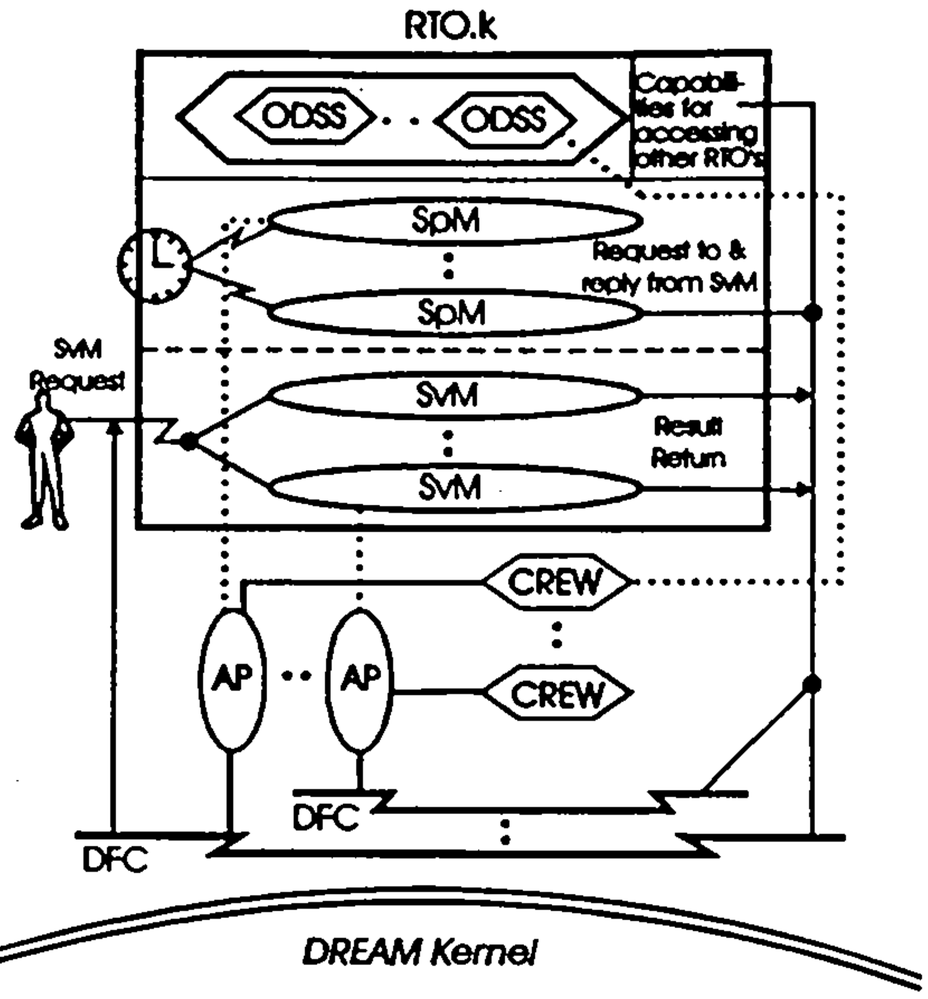


Figure 4. Mapping of an RTO.k object to a PCD program

The DREAM kernel treats components of RTO.k objects as special types of PCD components, i.e., special types of processes, CREW monitors, and DFC's. Figure 4 depicts the mapping relationship between components of an RTO.k object and the corresponding components of an equivalent PCD-program.

As shown in Figure 4,
(1) both SpM's and SvM's, are treated as application-specific program bodies of processes (called method execution processes or MEP's),
(2) access paths (i.e., SRQ's) to SvM's as DFC's created by the SvM's, and
(3) result-return paths to clients as DFC's created by the clients,
(4) ODS segments (ODSS's) as special CREW monitors,
(5) capabilities for accessing other RTO's as DFC ID's.
Note that this way of utilizing DFC's facilitates the transparency of object locations.

An RTO.k object method should be assigned to an MEP before the method can be executed. This assignment may be specified by the application programmer or may be left unspecified in which case the DREAM kernel will perform this function. We distinguish the following two types of RTO.k object method executions:
(1) An RTO.k object method executes on an MEP that is bound (by design) to this method
(2) An RTO.k object method executes on an MEP that is not bound to any method (i.e. the MEP is unbound).

When an MEP is bound to an object method, a permanent connection is established between the object method and the MEP. In this case, the corresponding object method will always execute on the same MEP to which it is bound. Also, in this case, we distinguish whether a single method is mapped to an MEP or a set of methods is mapped to the MEP. If only a single object method is bound to an MEP, then the MEP always executes just this method. In other words, the MEP is *permanently dedicated* to the object method. In contrast, a set of methods may share the execution facility offered by the same MEP. However, at any time the MEP is only *temporarily dedicated* to any one of the methods executing on it. One important requirement for sharing an MEP among several object methods is that these object methods cannot be active concurrently. This requirement has been imposed in order to reduce the complexity of implementation. The RTO.k designer who wishes to use the shared mapping facility should ensure that the methods that share the same MEP do not have timing requirements that necessitate their concurrent execution. The shared mapping of object methods onto the same MEP has the advantage of increased resource utilization (e.g., memory space) and would facilitate a large number of object methods to be executed on the same local node.

When a certain execution run of an RTO.k object method is assigned by the DREAM kernel to an unbound MEP, the MEP is *temporarily dedicated* to this method just for this run. The next execution run of this object method may be assigned to a different MEP. Thus in the case of an unbound MEP, no permanent connection exists between a method and the unbound MEP. SpM's are typically executed on bound MEP's. SvM's may be executed on bound or unbound MEP's, but should be executed on unbound MEP's when the pipelined execution of the SvM is enabled.

5.3 Construction of ODSS

In this section, we consider the construction of the Object Data Space Segment (ODSS) of an RTO.k object as a special CREW monitor. The ODSS of an RTO.k object may be concurrently accessed by multiple methods of the same RTO.k object. Hence a mechanism for concurrency control is necessary to maintain consistency within the ODSS and among the ODSS and its users. The use of a CREW monitor to construct an ODSS seems to be a natural choice.

The C++ RTO.k programmer can start designing the ODSSClass by inheriting the DREAM library class, *BasicODSSClass*. This BasicODSSClass in turn inherits the *CREWMgtClass* of the DREAM library. Due to this inheritance, the BasicODSSClass can provide the interface to kernel services such as: creating and destroying ODSS (CREW) support, obtaining and releasing ODSS access rights, etc. The following sample program shows the use of the ODSS class:

```
class ODSSClass:public BasicODSSClass {
    int x,...;  // private data area
  public:
    void Method1();  // a member function
```

```
    };
    void
ODSSClass::Method1() {
        // Obtaining RW
access rights to the CREW
                // monitor
by calling the member
function of the          //
CREWMgtClass of the
DREAM library
    EnterCREW_RW();

        ...; // access the
data area
        // Releasing RW
        // access rights to
            // the
CREW                     //
monitor
        ExitCREW_RW();
    }
    // Instantiation of the
    // ODSS class
    ODSSClass ODSS1;
```

5.4 Construction of RTO access capabilities

The RTO access capability section specifies the capabilities of the methods in this RTO.k object to call the services of other RTO objects. The access capability will be provided to a client RTO.k in the form of the DFC ID that should be used to request for execution of the corresponding SvM. In the approach adopted by the DREAM kernel, the creator of a server RTO.k should inform all potential clients about the creation of certain services by broadcasting the service-request DFC ID's of the SvM's of the server RTO.k via a special global broadcast DFC. At a client site, the client RTO.k, during initialization, makes a KSC to obtain the DFC ID of the external server method.

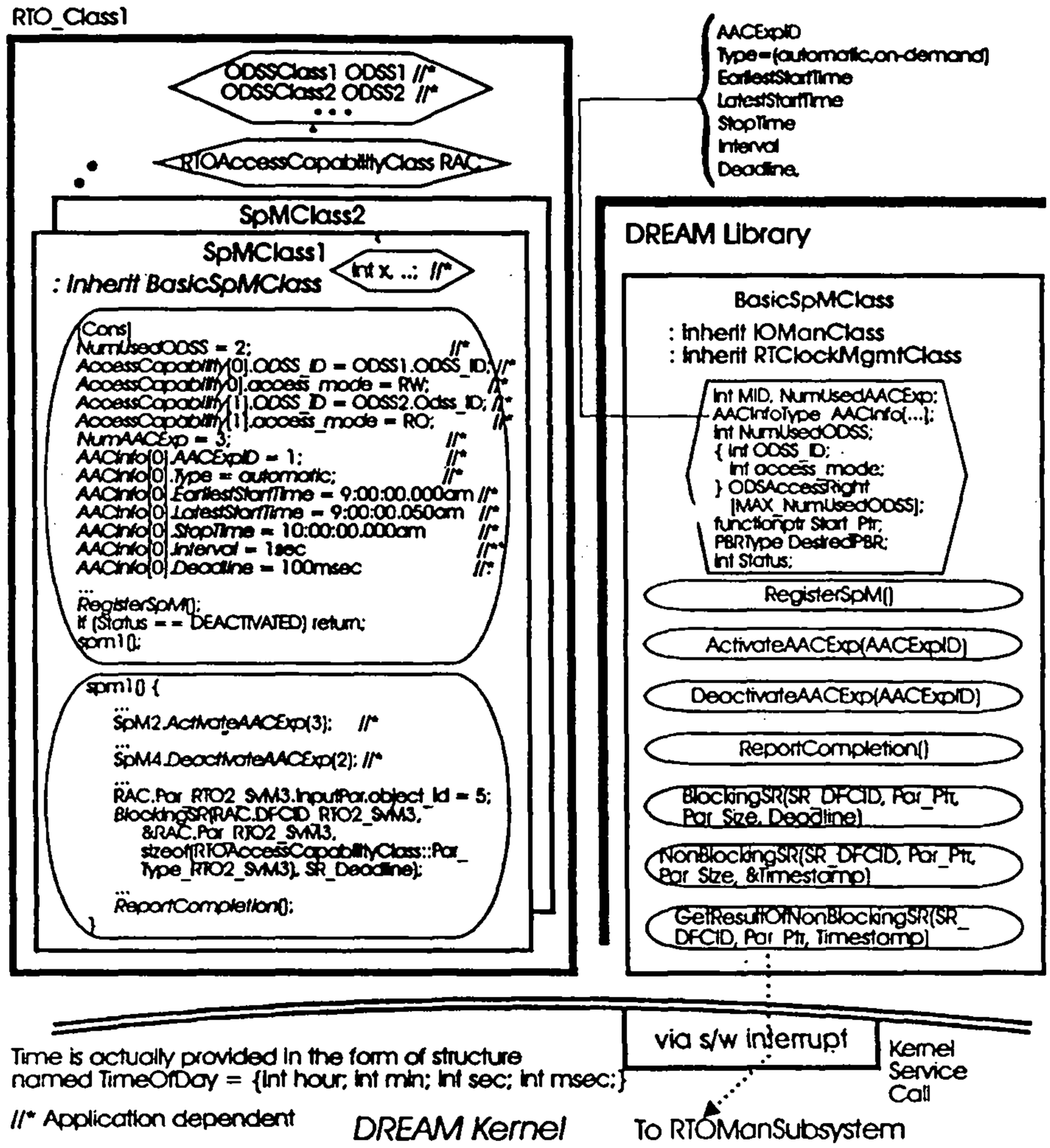


Figure 5. The SpMClass and the DREAM library support for SpM management

The RTO access capabilities are managed by a DREAM library class called the *BasicRTOAccessCapabilityClass*. The C++ RTO.k programmer can start designing an RTOAccess-CapabilityClass by inheriting the BasicRTOAccess-CapabilityClass. The BasicRTOAccessCapabilityClass provides the interface to the KSC for obtaining the capability for accessing an SvM in another RTO.k object. This KSC RegisterRTOAccessCapability() obtains the DFC ID of the SvM whose symbolic name is specified as input parameter. It is a good practice to define inside the RTOAccessCapabilityClass the parameters for each service request to be sent from this RTO. The following illustrates construction of RTO access capabilities:

```
class RTOAccessCapabilityClass: public
BasicRTOAccessCapabilityClass
{
    public:
     // parameter for access to SvM3 of RTO2
     struct ParType_RTO2_SvM3 {
          ...;
     } Par_RTO2_SvM3;
     // Service-request DFC ID of SvM3 of RTO2
     int DFCID_RTO2_SvM3;
     RTOAccessCapabilityClass(); // constructor
};
RTOAccessCapabilityClass::
     RTOAccessCapabilityClass()
{
     RegisterRTOAccessCapability
```

```
    ("RTO2", "SvM3", &DFCID_RTO2_SvM3);
}
// Instantiation of the RTOAccessCapability class
RTOAccessCapabilityClass RAC;
```

5.5 Construction of SpM's

As we mentioned in section 5.2, SpM's may be bound to MEP's or may be left unbound. In case the SpM's are bound to MEP's, these MEP's may be generic or custom-tailored for the SpM. In the case of *generic MEP's*, the process priority, process stack size, etc. are fixed to some pre-defined standard values. On the other hand, these values may be custom-tailored for *custom MEP's*. Each SpM may request the services of one or SvM's through the appropriate service-request DFC. The result-returns from the SvM's are obtained through a unique reply DFC associated with the SpM. This unique reply DFC is allocated to an SpM by the DREAM kernel during the SpM initialization (registration) time. An SpM client may make two types of service requests [Kim94a]:
(1) *blocking* service requests in which the client is blocked after making a service request until the result is obtained from the server or until a pre-specified timeout expires.
(2) *non-blocking* service requests in which the client proceeds after making a service request and retrieves the result of the request later during its execution.

The DREAM library supports the registering of the SpM with the DREAM kernel, mapping of the SpM to an MEP, blocking and non-blocking service request calls, etc. Figure 5 shows the DREAM library support offered for SpM construction and management. The C++ RTO.k programmer defines an *SpMClass* inside an *RTOClass*. The SpMClass inherits the *BasicSpMClass* of the DREAM library. As shown in figure 5, the constructor of the SpMClass initializes several data items of the BasicSpMClass and registers the SpM with the DREAM kernel. The data items initialized include: number and ID's of the ODSS's accessed by this SpM along with the respective access modes, autonomous activation condition (AAC) information, start pointer to the function body of this SpM, etc.

After registering the SpM with the kernel, the user may create an MEP and bind the SpM to the MEP. This binding is typically done inside the AIP using DREAM library calls. The SpM can be permanently bound to one MEP or can be one among several SpM's that are mapped to the same MEP (shared binding), or left unbound at the end of the initialization. All unbound methods are mapped to (unbound-free) MEP's at the activation time of the method by the DREAM kernel.

5.6 Construction of SvM's

SvM's are *usually not bound* to any MEP unless the pipelined execution is explicitly disabled. When the SvM is activated, the kernel maps the method to an unbound-free MEP and executes it. If the service request frequency for an SvM is sufficiently high, the designer may decide to execute the SvM in a pipelined fashion. In other words, when multiple service requests arrive, each service request is handled by a separate invocation of the same SvM. In such cases, the kernel maps each SvM invocation to a different MEP. Each SvM is associated with a unique service-request DFC through which it may receive service requests. This service-request DFC is assigned to the SvM by the DREAM kernel during its registration time. Each SvM invocation is associated with a unique reply DFC through which it may receive replies from server methods to which it sent requests. The ID's of these reply DFC's are determined by the DREAM kernel at the time of the activation of the SvM invocation and are hidden from the RTO.k application programmer. The service requests from a client method to an SvM carry the ID of the reply DFC to which the SvM should send the reply back.

SvM itself may be a client of some other SvM(s), the SvM client may make both blocking and non-blocking service requests. In addition, an SvM may also pass a client request to another SvM by using a *client-transfer* call [Kim94b]. The latter SvM may again pass the client request to another SvM. This chaining sequence may repeat until the last SvM in the chain returns the results to the client. The main motivation behind such a client transfer call stems from the basic concurrency constraint which requires an execution of an SvM to be made only if a sufficiently large time window between potentially conflicting SpM's opens up. Hence, in certain situations a highly complicated SvM may never be executed due to lack of a wide enough time window. One way to get around this problem is to divide the SvM into multiple smaller SvM's, SvM1, .. SvMx. A client can then call each smaller SvM each time. Calling each smaller SvM incurs communication overhead of transmitting a request to the smaller SvM and obtaining the results. Eliminating most of such communication overhead is the motivation behind an arrangement in which the client calls the first SvM and the latter passes on the client to another SvM and so on until the last SvM of the chain returns the results to the client.

The DREAM library supports the registering of the SvM with the DREAM kernel, receiving service requests from clients, returning results to client methods, blocking and non-blocking service requests to other SvM's, and client transfer calls. The RTO.k programmer can start defining an SvMClass by inheriting the *BasicSvMClass* of the DREAM library. The constructor of the SvMClass should initialize several data items defined in the BasicSvMClass and then register the SvM with the DREAM kernel. The initialized data items include: the name of this method and the owner RTO.k, number and ID's of the ODSS's accessed by this SvM along with the respective access modes, method deadline, whether or not pipelining is allowed, start pointer to the function body of this SvM, etc.

Upon registering an SvM, the name of the SvM and the owner RTO.k along with the service-request DFC ID (assigned by the DREAM kernel) are broadcast to all nodes which may contain potential clients. The potential client RTO.k needs to obtain the capability for accessing the server method before they can request the service. Upon arrival of a service request, the DREAM kernel recognizes the request, binds and dedicates one SvM invocation to an unbound-free MEP, and activates the MEP for execution. It also connects this MEP to the appropriate reply DFC through which the SvM invocation may receive replies from SvM's to which it sent requests.

6. Advantages and Limitations of RTO.k Object Execution using PCD Components

The execution of RTO.k objects using PCD components offers several advantages to the system designer. These include:
(1) The use of DFC's for inter-RTO communication makes the RTO.k objects to be location transparent and allows easy and efficient migration of RTO.k objects. There exists no difference if communicating RTO.k objects are located in the same node or in different nodes.
(2) General purpose kernels (such as DREAM kernel) supporting various types of concurrent and distributed application software can be utilized for the execution of RTO.k objects.
(3) The shared binding of several object methods to an MEP provides for very efficient resource utilization and facilitates the execution of a large number of object methods on the same local node.

Among the limitations of this type of structuring include:
(1) Efficient implementation of DFC is possible only if the underlying communication network supports efficient broadcast capability.
(2) The use of DFC's for communication among RTO's within a local node may not be as efficient as a shared-memory form of communication. However, in the latter case, it is difficult to realize the location transparency of RTO's.

Even though the DREAM library eases the efforts for structuring C++ RTO.k objects, it still imposes burdens on the application designer of dealing with low-level details in redundant forms which could be automatically deduced from a high-level abstract and yet accurate representation of RTO.k objects. An extension of the C++ language to support abstract RTO.k programming is the most natural path to follow in this direction. The authors are currently developing one such extension named the C++T language.

7. Conclusion

The imitation of RTO.k objects using PCD program components facilitates the execution of RTO.k objects on a general-purpose kernel such as the DREAM kernel that views RTO.k objects as PCD program components. The DREAM library considerably reduces the programming efforts through use of the advanced inheritance features of C++. The DREAM kernel offers guaranteed timely services to real-time applications. These suite of tools would go a long way in our efforts to realize the GT design paradigms. One meaningful direction to pursue is to develop the C++T language, the extension of C++ language, that would provide a natural means of accurately representing RTO.k objects.

Acknowledgment: The research work reported here was supported in part by US Navy, NSWC Dahlgren Division under Contract No. N60921-92-C-0204, in part by the California Transportation Department via the UCI Institute for Transportation Studies, in part by Hitachi, Ltd, in part by IBM, in part by Postech, and in part by ETRI.

References

[Att91] Attoui, A. "An Object Oriented Model for Parallel and Reactive Systems", *Proc. IEEE CS 12th Real-Time Systems Symp.*, 1991, pp. 84-93.
[Bri77] Brinch Hansen, P., 'The Architecture of Concurrent Programs', Prentice-Hall, 1977.
[Ish92] Ishikawa, Y., Tokuda, H., and Mercer, C. W., "An Object-Oriented Real-Time Programming Language", *IEEE Computer*, October 1992, pp. 66-73.
[Kim94a] Kim, K.H. and Kopetz, H., "A Real-Time Object Model RTO.k and an Experimental Investigation of Its Potentials", *Proc. 1994 IEEE CS Computer Software and Applications Conf. (COMPSAC)*, Nov. 1994, Taipei, pp.392-402.
[Kim94b] Kim, K.H. et al., "Distinguishing Features and Potential Roles of the RTO.k Object Model", *Proc. 1994 IEEE CS Workshop on Object-oriented Real-time Depen-dable Systems (WORDS)*, Oct. 94, Dana Point, pp.36-45.
[Kim95a] Kim, K.H., Mori, K., and Nakanishi, H., "Realization of Autonomous Decentralized Computing with the RTO.k Object Structuring Scheme and the HU-DF Inter-Process-Group Communication Scheme", *Proc. 1995 IEEE CS Int'l Symp. on Autonomous Decentralized Systems (ISADS)*, April 1995, Phoenix, pp.305-312.
[Kim95b] Kim, K.H. et al., "A Timeliness-Guaranteed Kernel Model - DREAM Kernel and Implementation Techniques", *Proc. 1995 Int'l Workshop on Real-Time Computing Systems and Applications (RTCSA 95)*, Tokyo, Japan, Oct. 1995, pp.80-87.
[Kop89] Kopetz, H. et al., "Distributed Fault-Tolerant Real-Time Systems: The Mars Approach", IEEE Micro, Feb. 1989, pp. 25-39.
[Kop90] Kopetz, H. and Kim, K.H., "Temporal Uncertainties in Interactions among Real-Time Objects",

Proc. IEEE CS 9th Symp. on Reliable Distributed Systems, Huntsville, AL, Oct. 1990, pp.165-174.

[Mor93] Mori, K., "Autonomous Decentralized Systems: Concept, Data Field Architecture, and Future Trends", *Proc. IEEE CS Int'l Symp. on Autonomous Decentralized Systems (ISADS 93)*, Mar.93, Kawasaki, Japan, pp. 28-34.

[Ras89] Richard Rashid et al., "A Foundation for Open Systems", *Proc. the IEEE Second Workshop on Workstation Operating Systems (WWOS2)*, September 1989, pp.109-113.

[Tak92] Takashio, K., and Tokoro, M., "DROL: An Object-Oriented Programming Language for Distributed Real-Time Systems", *Proc. OOPSLA*, 1992, pp. 276-294.

여 백

제 4 편

제3차년도 연차보고서

여 백

목차

여 백

제 3 차년도
연차 보고서

RTS 기술개발

RTS Technology Development

거시적 시뮬레이터 구축 기술 개발

연구기관
포항 공과 대학교

과 학 기 술 처

여 백

제 1 장 거시적 실시간 시뮬레이터 구축 기술 개발

제 1 절 서론

컴퓨터 하드웨어의 성능이 급속하게 발전함에 따라 컴퓨터의 응용분야 역시 그 영역을 빠르게 넓혀 가고 있다. 컴퓨터의 고성능화로 가능해진 응용 분야는 일반적으로 대규모의 행동이 복잡한 실시간 컴퓨터 시스템을 요구하고 있다.

실시간 시스템이란 시스템의 행동의 올바름이 기능적인 측면뿐만 아니라 시간적인 측면에 의해서도 결정되는 시스템으로 원자력 발전소 제어 시스템, 기상 인공위성 제어 시스템, 미사일 제어 시스템, 그리고 교통 정보 시스템 등이 이에 속한다. 이런 실시간 시스템은 그 규모가 방대하고, 행동이 복잡하며, 시간적 제약이 엄격한 특징을 가진다. 이러한 복잡한 행동 양식을 보이는 실시간 시스템의 개발에 있어서, 사용자 요구 사항의 분석은 매우 어려울 뿐만 아니라 시스템 개발 완료 시까지는 충분한 유효성 검사가 힘든 실정이다.

요구 사항의 명세에는 두 일단이 참여하게 된다. 하나는 시스템을 사용하는 단말 사용자이고 다른 하나는 시스템을 분석하는 분석자이다. 일반적으로 단말 사용자는 컴퓨터 시스템에 대한 지식이 없고 분석자는 개발될 특정 시스템에 대한 기반 지식이 없기 때문에 요구 사항 명세(Requirement Specification)는 비정형적이거나 모호하거나 불완

전한 경우가 많다. 이러한 문제점을 해결하기 위해서 정형적인 방법(Formal Method)이 사용되지만 단말 사용자는 정형적인 명세서(Formal Specification)를 이해하기 힘들기 때문에 요구 사항 명세서의 유효성 검사를 하기가 힘들다. 만약 요구 사항 명세서의 오류가 소프트웨어 생명주기 후반기에 발견된다면 이 오류를 초기에 고치는 것에 비해 비용이 수백 배에 이른다는 것이 잘 알려져 있다.

사용자 요구 사항의 유효성 검사를 위한 방법은 크게 3가지 부류로 나뉘어 진다. 즉, 정형적인 방법, 명세의 수행과 시뮬레이션, 프로토타이핑 등이 있다. 이들 각각에 대해 아래에서 언급한다.

정형적인 방법에서는 사용자 요구 사항이 수학적 이론과 방법을 적용하여 정형적으로 명세화된 후에 안전성, 일치성과 완전성과 같은 시스템 성질이 수학적으로 검증된다. 그러므로 정형적인 방법은 시스템의 올바름(Correctness)를 보장할 수 있다.

시스템이 복잡해지고 실시간 성질을 가짐에 따라, 분석되어야 할 시간적 공간적 범위가 극도로 증가하게 된다. 그러므로 이러한 방법을 전체 시스템 명세에 적용하기에는 비실용적인 면이 있다. 또한 이러한 방법은 최종 시스템이 어떠한 모습으로 행동하는지를 보고 느낄 수 있는 방법을 제공하지 못하기 때문에 사용자로 부터 시스템의 유효성을 검증받는 방법이라기 보다는 시스템의 분석에 더 적합한 방법이다.

시스템 명세를 수행시키기 위한 많은 노력이 있어 왔고 많은 명세 수행 방법이 제안되었다. 이 방법은 사용자의 유효성 검사를 위해 시뮬레이터를 이용하여 명세를 해석하고 시스템 행동을 시각적으로 보여 준다. 또 실시간 시스템을 위해 추계적인(Stochastic) 모델을 사용하여

실시간 시스템의 행동을 명세하고 시뮬레이션하고 유효성을 검증하기도 한다.

이러한 방법의 몇 가지 장점은 다음과 같다.

- 시스템의 모습을 보고 느낄 수 있고 시스템의 설계와 구현 전에 사용자로 부터 빠른 검증을 받을 수 있다는 것이다. 이러한 방법을 사용하면 사용자가 요구 사항 명세 단계에 적극적으로 참여하여 요구 사항 추출과 검증 사이의 간격을 극도로 좁힐 수 있고 많은 선택 사항이 경제적으로 평가될 수 있다.

- 정형적인 분석 방법으로는 적합하지 않은 분야를 보완할 수 있다. 예를 들면, 실시간 요구 사항의 행동적 민감성 테스트는 정형적인 방법으로는 불가능하다. 즉 명세를 수행시켜 전체 시스템 행동을 시뮬레이션 함으로써 실시간 요구 사항에 민감한 프로세스를 쉽게 찾아낼 수 있다.

하지만 명세의 수행 방법은 일반적으로 명세에 대해 올바름의 신뢰도를 높여 주는 역할은 하지만 이를 보장하지는 않는다. 시스템의 어떤 성질의 올바름을 보장하기 위해서는 가능한 모든 경우에 대해 검사를 해봐야 한다. 이것은 시스템의 크기가 커다란 경우에는 거의 불가능하다. 그러므로 정형적인 분석 방법과 명세의 수행을 통한 방법 모두가 요구 사항 분석을 위해 필수적이다.

하지만 정형적인 방법과 명세의 수행 방법 모두를 제공하는 방법은 그리 많지 않다. 개발중인 ASADAL(A System Analysis and Design Aid tooL system) CASE 도구는이들 두 가지 방법을 모두 지원한다. 이 도

구의 특징을 요약하면 다음과 같다.

- 위험 요소가 있는 시스템 성질 - 안전성, 생존성, 실시간 반응성 - 의 검증을 위해 정형적인 분석 방법을 제공한다.

- 명세의 수행을 통해 생명 주기 초기 단계에 요구 사항의 유효성 검사 방법을 제공한다.

세부 과제 거시적 실시간 시뮬레이터 구축 기술 개발은 이 ASADAL CASE 도구상에 실시간 시스템의 명세와 그 실시간 시뮬레이션을 돕는 기술을 개발하는 것을 목표로 하고 있다.

1차년도(1994.9-1995.8)에서는 거시적 실시간 시뮬레이션을 위한 시스템의 모델링 방법인 ASADAL 요구 분석 방법론을 개발하였고 그 방법론에 따라 RTET(Real-time Event Trace), TES(Time Enriched Statechart), DFD(Data Flow Diagram)을 그리는 그림 도구를 만들고 시뮬레이터 프로그램으로서의 이 그림들을 분석, 데이터베이스에 의미 있는 형태로 저장할 수 있게 하였다. 또, 이들 명세 언어 RTET, DFD, TES 그림들을 ASADA:L 명세 언어라 한다.를 이용한 시스템의 명세를 시뮬레이션하는 정형적인 방법과 추계적인(Stochastic) 시뮬레이션을 하기 위한 기본 프로세스 명세 언어와 시뮬레이션 드라이버 프로그램도 정의되었다.

2차년도(1995.9-1996.8)에서는 이를 더욱 발전된 형태로 개발하였는데, 우선 DFD의 분할(Decomposition)과 TES명세간의 관계, 분할된 DFD의 시뮬레이션 방법 등에 대한 연구가 진행되었고 사용자 대화형 시뮬레이터를 비롯하여 추계적인 시뮬레이터를 위해 정의된 기본프로세스

명세와 시뮬레이션 드라이버 프로그램을 해석하여 시뮬레이션하는 프로그램을 개발, 기존의 시뮬레이터와 연결하여 추계적인 배치 모드의 시뮬레이션을 가능하게 하였다. 그리고 ASADAL 명세 언어로 만들어진 명세가 여러 실시간 제약 조건을 만족하는지의 여부를 검증하는 기술을 개발하였다. 또한 지능형 생산 공장을 시뮬레이션하기 위한 모델을 구축, ASADAL 방법론을 실제 환경에의 적용성을 높였으며 실제 그 모델을 이용하여 참여 기업(POSCON)에서 제시한 냉간 압연 시스템의 자동 철판 두께 제어기(AGC: Automatic Gauge Controller) 및 장력 제어기(SC: Speed Controller)를 명세, 시뮬레이션 하였다.

이에 이어 당해년도(1996.9-1997.8)에서는 시뮬레이션 결과로 얻어진 자료를 시각화(Visualization)하는 기능을 추가하였으며 명세를 나누어 여러 시뮬레이터에 분산하여 시뮬레이션할 수 있게 하였고 Queueing Model 및 Markov Chain Model에 근거한 자료 분석 및 도달성(Reachability) 검사, 전이(Transition) 사용성 검사, 비결정성(Nondeterminism) 검사 등의 기능을 개발, ASADAL CASE 도구에 구현하였다.

앞으로 이 장 2절의 1 에서는 분산 시뮬레이션 기술 및 그 도구에 관해 정리하였고 2 에서는 데이터 분석기(Data Analyzer)에 의한 분석 방법과 그 도구를, 3 에서는 개발된 여러 검사 방법들에 대해 장리하겠다. 4 에서는 미시적 시뮬레이터와 통합, 운영하는 시뮬레이션 방법을 지능형 공장 시뮬레이션 모델의 예제를 통해 살펴토도록 하고 3절 에서 앞으로 해야할 일을 제시하고 결론을 맺도록 한다.

제 2 절 분산 시뮬레이터와 시뮬레이션 결과 분석기의 개발 및 지능형 공장 시스템에의 적용

1. 분산 시뮬레이션 기술 및 그 도구

분산 시뮬레이션은 명세의 크기가 커질 때, 분산 환경에서 소프트웨어가 동작하게 될 때등의 경우에 효율적이고 빠른 시뮬레이션이나 실제와 분산환경과 같은 시뮬레이션을 위해 필요하다.

이 절에서는 먼저 ASADAL에서 사용하는 분할 상세화 방법 및 그 시뮬레이션이 어떠한 방식으로 아루어지고 있는 지 살펴 보고 그 환경에서 확장된 분산 모델과 그 시뮬레이션 방법에 대해 설명하겠다.

가. 시스템의 분할 상세화

DFD는 예전부터 프로세스(Process; Bubble)를 분할, 상세화 하는 방법을 갖추고 있었다. ASADAL에서는 거대하고 복잡한 시스템의 단계적인 상세화를 DFD로써 제공한다. 시스템의 구조는 DFD의 분할 구조(Decomposition Hierarchy)로 표현되어지고 이제 각각의 DFD에 제어기가 그려지고, 제어기는 하나의 TES로 명세된다.

그림 1은 이러한 ASADAL 명세 언어와 그 모델을 보이고 있다. 먼저 외부적 관점에서 본 시스템 명세를 그림의 (a)와 같이 MSD를 이용해서 하게 된다. 이 그림에는 만들고자 하는 제어 시스템(Target System; TS)외에 시스템 내에 존재하는 관련된 다른 개체들이 표현되고, 제어

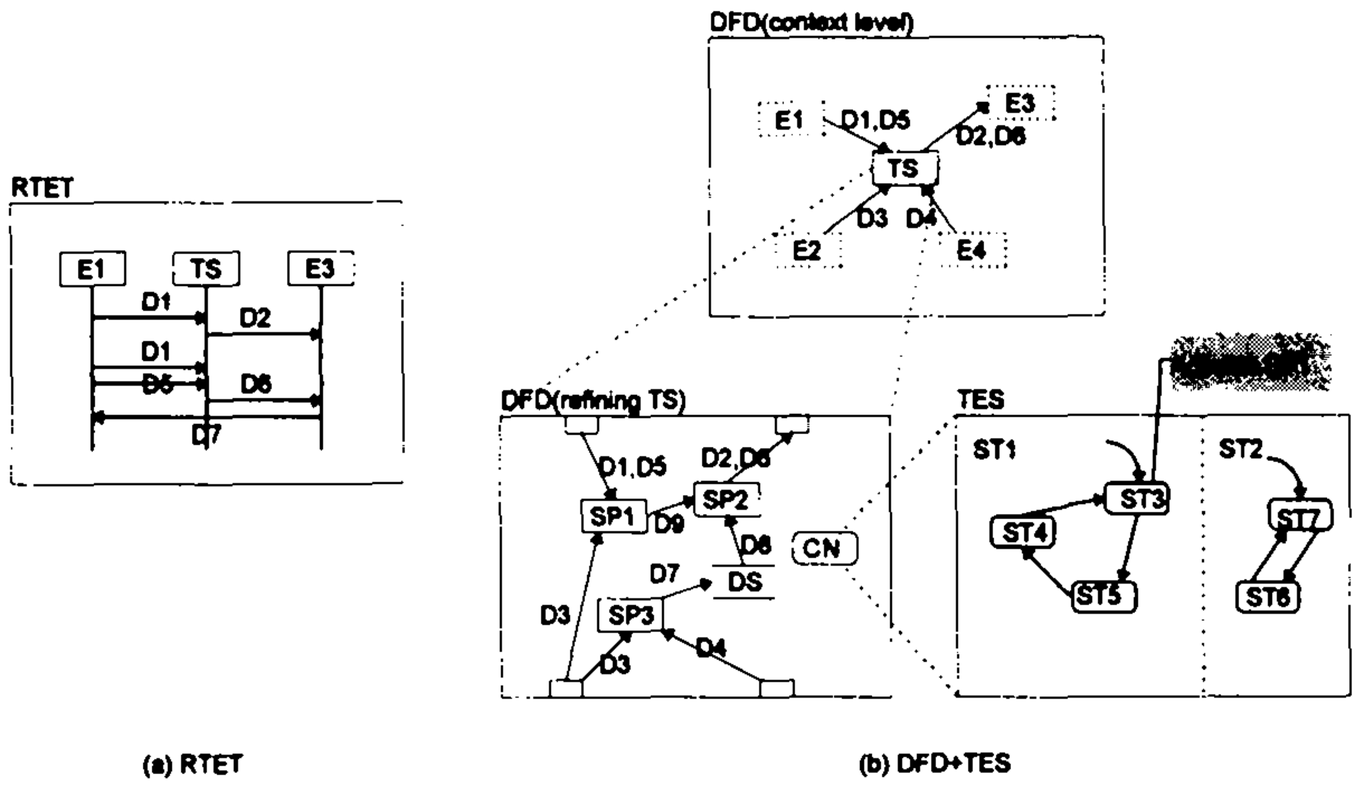


그림 1 ASADAL 시뮬레이션 모델

시스템과의 메세지 교환 외에 시나리오와 관련된 다른 개체들간의 것도 표현되어 시스템 환경 전체의 시나리오가 진행됨에 따른 메세지 흐름을 볼 수 있다.

그림 2는 MSD로 철길 건널목의 차단기 시스템의 간단한 MSD 명세를 보이고 있다. 컴퓨터는 건널목의 차단기와 기차로부터 현 상태에 대한 정보를 얻고, 이 정보를 이용하여 이들에 유용한 정보를 주거나 제어 신호를 보내 기차가 건널목을 지나갈 때는 반드시 차단기가 내려갈 수 있게 해 준다. 이 그림 오른쪽엔 하나의 일반적인 시나리오를 MSD로 보이고 있다. 먼저 기차가 건널목에 접근하면 제어 시스템은 차단기에 건널목을 내리라는 제어 시그널을 보낸다. 그 신호에 의해 차단기는 작동하고, 그 작업이 완료되면 그 사실을 다시 제어 시스템에 알린다. 그럼 제어 시스템은 기차의 건널목 진입을 허용한다. 이 때 그림 오른쪽 아래 나타난 것이 MSD에 포함되어 있는 실시간 요구사항으로서 이 시나리오대로 작동하는 경우 차단기 내림 시그널과 차단기 내림 완료 시

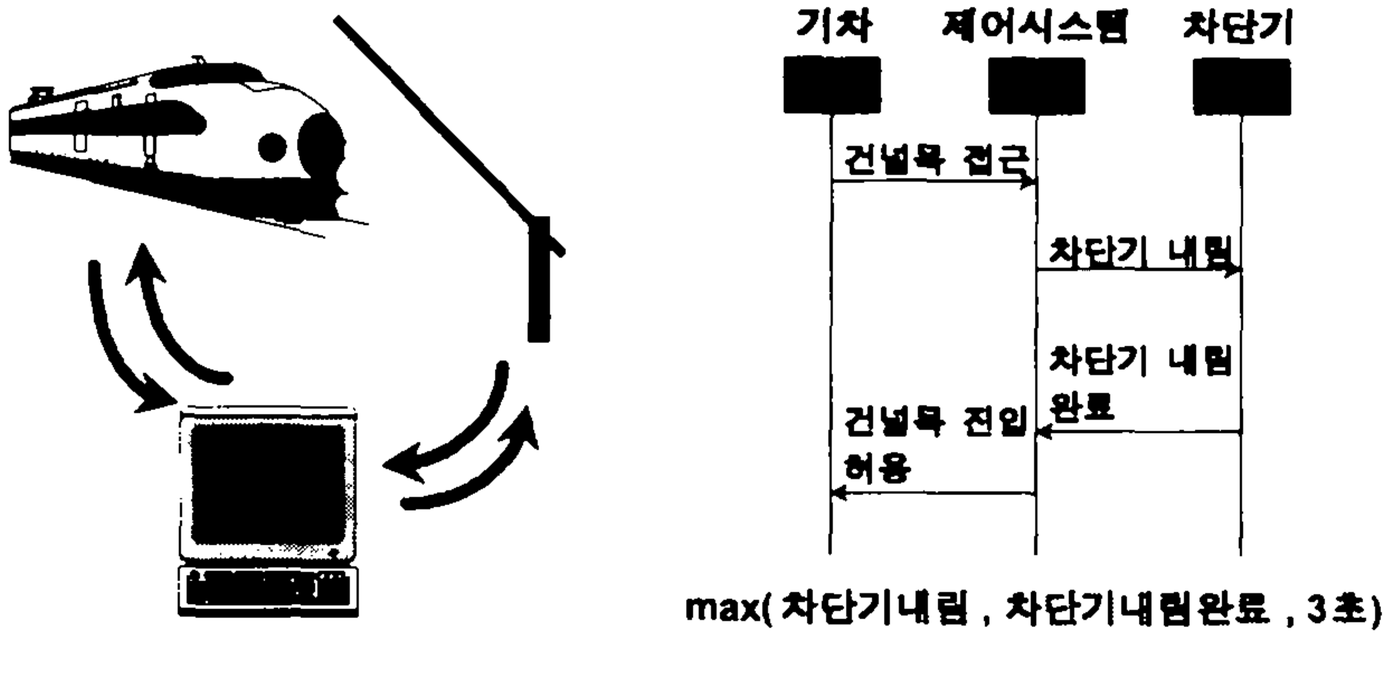


그림 2 차단기 MSD 예제

그널 사이에 걸리는 시간이 3초를 넘어서는 안됨을 말하고 있다.

이렇게 MSD가 명세되면 MSD에서 개체로 표현되었던 제어 시스템 자체에 대한 내부적 관점의 명세를 하게 된다. 이는 DFD를 통해 이루어진다. 최상위 단계(Context Level)의 DFD 그림에서는 먼저 제어 시스템을 하나의 프로세스(Process; Bubble)로 놓고 그것과 데이터나 제어 신호(Control Signal)의 흐름이 있는 개체들을 외부 프로세스(External Process)로 놓고, 제어 시스템으로의 입력과 출력을 명세하게 된다. 이것이 그림 1 (b) 의 최상위 단계의 DFD이다. 이 그림에는 제어 시스템으로의 모든 입력과 출력이 명세되어야 하며 따라서 MSD에서 나타나는 제어 시스템으로의 입력과 출력은 DFD에 나타나 있는 것이어야 한다.

그림 1(b)의 최상위 단계 DFD에는 제어 시스템을 TS라는 이름의 프로세스로 명세하였고, 이것이 분할 상세화된 것이 그 그림의 왼쪽 아래에 나타나 있는 DFD이다. 이 DFD는 TS프로세스가 실은 SP1, SP2, SP3의 세 프로세스들을 가지고 있으며 TS로 들어온 입력이 이들 프로세스

에 어떻게 전해지고 또 이들이 어떻게 TS의 출력을 만들어 내는지를 보인다. 이 과정에서 TS의 입력과 출력이 그것이 상세화된 그림에서 완전히 같은 입력과 출력으로 나타나야 한다는 것은 주지의 사실이다.

분할 상세화 된 DFD(refining TS라고 설명된)에서 모서리 둥근 사각형으로 표현된 이름이 CN인 제어기(Controller)가 있는데, 이 것이 그 오른쪽 그림과 같은 TES로 상세화 되어 시스템의 상태 변화에 있어서 프로세스들이 어떻게 활성화되는지를 보이게 된다. 예를 들어 이 그림에서 초기 상태인 ST3은 보이는 바와 같이 SP1 프로세스를 활성화하게 된다. 그림에서 activate...의 표현은 임의의 것으로 ASADAL 명세의 문법(Syntax)은 아니다. 즉, 시스템이 ST3 상태로 될 때 SP1의 기능을 수행하고 ST 상태에서 다른 상태가 될 때 그 상태는 끝나게 된다.

일반적으로 하나의 DFD에는 하나의 제어기가 있어서 그 DFD에 명세된 프로세스들의 행위(Behavior)들이 시스템 상태 변화에 따라 어떻게 일어나는지 명세하게 된다. 하지만 그림에서와 같이 최상위 단계에서는 프로세스가 하나만 있고 따라서 제어기는 무의미하다. 즉, 이 프로세스는 항상 활성(Active)이다. 최상위 단계뿐 아니라 이와 같이 제어기가 없는 DFD 그림의 프로세스들은 모두 동시에(Concurrent) 활성인 것으로 본다. 왜냐하면 이러한 행위를 나타내는 TES는 쓸 데 없는(Trivial) 것이기 때문이다.

하나의 프로세스는 이를 제어하는 제어기의 TES 그림에서 단 하나의 상태에서만 활성화한다. 그리고 시스템은 항상 어떤 기능을 행하게 되므로 모든 TES 그림에 나타난 상태들은 어떠한 프로세스를 활성화시켜야 한다. 또 여러 프로세스를 활성화시키는 상태가 있을 수 있다. 하지

만 이 경우 그들을 연속적으로(Sequentially) 활성화시키는 것으로 보기 때문에 동시에(Concurrently) 활성화되는 프로세스들의 경우 여러 개의 병렬 상태(Parallel State)에서 활성화하도록 명세되어야 한다. 예를 들어 그림 1(b)의 TES 그림에서 ST3, ST4에서 활성시키는 프로세스들은 동시에 활성화될 수 없지만 ST3과 ST6은 동시에 현재상태로 될 수 있으므로 그들이 활성화하는 프로세스들은 동시에 수행된다.

DFD의 가장 낮은 단계에 존재하는 더 이상 상세화할 필요가 없는 프로세스들은 프로세스 명세(Process Specification)을 가지며 ASADAL에서는 이것을 기술하기 위한 명세 서술 언어(Specification Description Language, SDL)을 정의하고 있다. 다음은 이 언어를 이용한 엘리베이터의 램프를 제어하는 프로세스의 명세 예이다.

```
Process Control_Elevator_Lamp
Input
    floor_info: RECORD {
        arrived floor:integer ;
        previous floor:integer ;
    }
Output
    lamp_command: RECORD {
        lamp_no: integer ;
        command: commandtype;
    }
ComputationSpecification
    On Arrival(floor_info) do
        ProcessTime randExp(5ms) ;
        Begin
            lamp_command.command =  OFF  ;
            Generate(lamp_command) ;
        End
ExecutionControl
    ResumeSuspend ;
```

이 프로그램은 “floor_info” 입력이 들어오면 평균 5ms의 지수(Exponential)분포를 갖는 계산 시간후에 램프 명령(lamp_command)을 출력으로 내보낸다는 것을 보이고 있다.

그럼 이러한 각 그림들간의 관계와 DFD가 상세화 되었을 때 이들의 시뮬레이션이 어떻게 동작하는 지를 보도록 하겠다.

나. 거대, 복잡한 명세의 시뮬레이션

거대하고 복잡한 명세를 하게 되면 앞서 설명했 듯 DFD의 상세화가 필수적이며 한 프로세스와 그것이 상세화된 DFD간의 관계에 대해서도 앞서 언급했다.

그림 3은 DFD와 그 중 한 프로세스 P3을 상세화 한 DFD를 보인 것이다. (a)의 DFD도 어느 한 프로세스를 상세화한 것으로 그 기능들을 세부적으로 P1, P2, P3로 나눈 것인데, 이들은 제어기 CN에 의해 제어되고 P3의 경우는 (c)와 같이 다시 상세화된다. 이 때 그림 (a)가 나타

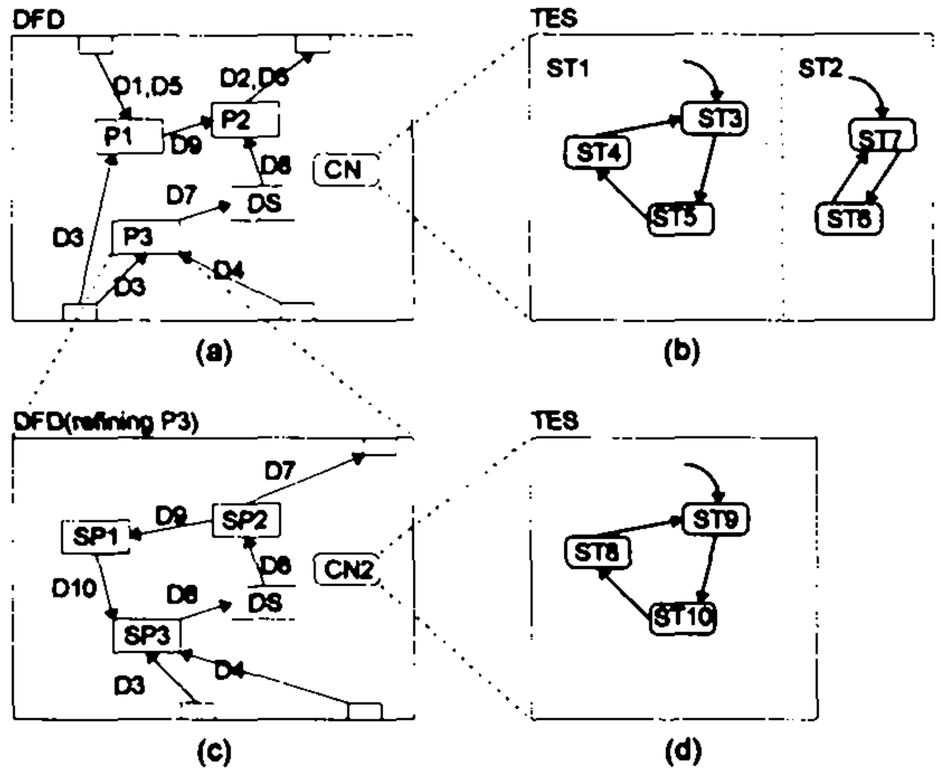


그림 3 DFD 구조와 시뮬레이션

내는 프로세스가 활성화될 때 이 그림에 나타나 있는 기능, 행위들은

작동한다. 즉, 비활성일 때 모든 하부 프로세스(P1과 같은)들도 비 활성이 된다. 또한 제어기 CN도 아무런 행위를 보이지 않는다. 이 프로세스가 활성화되면 제어기 CN의 시뮬레이션이 시작된다. 제어기에 대한 시뮬레이션이 되면 현재의 상태가 바뀜에 따라 활성화되는 프로세스들이 바뀐다. 결국, CN은 프로세스 P1, P2, P3를 활성화, 비활성화 하는 역할을 한다. 이 제어기 상태변화는 프로세스들의 기능 수행에 의한 데이터나 조건(조건) 변수 값의 변화, 또는 데이터의 발생, 프로세스의 중단(Termination), 시간 경과(Timeout)와 같은 사건(Event)의 발생에 의해 이루어진다. 단, 데이터나 조건의 변수값은 전역으로 어느 곳의 제어기에서도 참조할 수 있지만 사건의 발생은 그 발생 원인이 게어할 DFD 그림 내에 있는 것으로 한정한다. 그림이 너무 복잡해지고, 참조 대상을 찾기 어려워지는 현상을 막기 위해서이며 그것으로도 한 그림을 제어하기 위한 정보는 충분하리라고 생각한다.

그림 3의 (b)에서 상태 ST3이 (a)의 프로세스 P3을 활성화하면 (c)의 제어기 명세인 (d)가 행위를 보이기 시작한다. 처음엔 초기 상태 ST9이 현재상태로 될 것이다. 그러면서 그 상태에 활성해야 할 (c)의 프로세스를 다시 활성화할 것이다. 그 후 상황 변화에 따라 TES에서 상태 전이(State Transition)이 발생할 것이며 그에 맞게 (c)의 프로세스들이 동적으로 활성화된다. 그러다가 어느 순간 (b)에서 ST3 상태를 나가면 프로세스 P3는 비활성화되고 따라서 (d)도 비 활성화된다. 이 과정에서 (d)에 상태 기록자(History Connector)가 없으면 다음 활성시엔 ST9가 다시 현재 상태가 활 것이다. 그리고 (c)의 프로세스들도 비활성화되는데, 이들은 하던 일을 멈추고 그 특성에 따라 다음 활성시에 처음부터 다시 기능을 수행하거나 이전에 멈춘 곳으로부터 계속

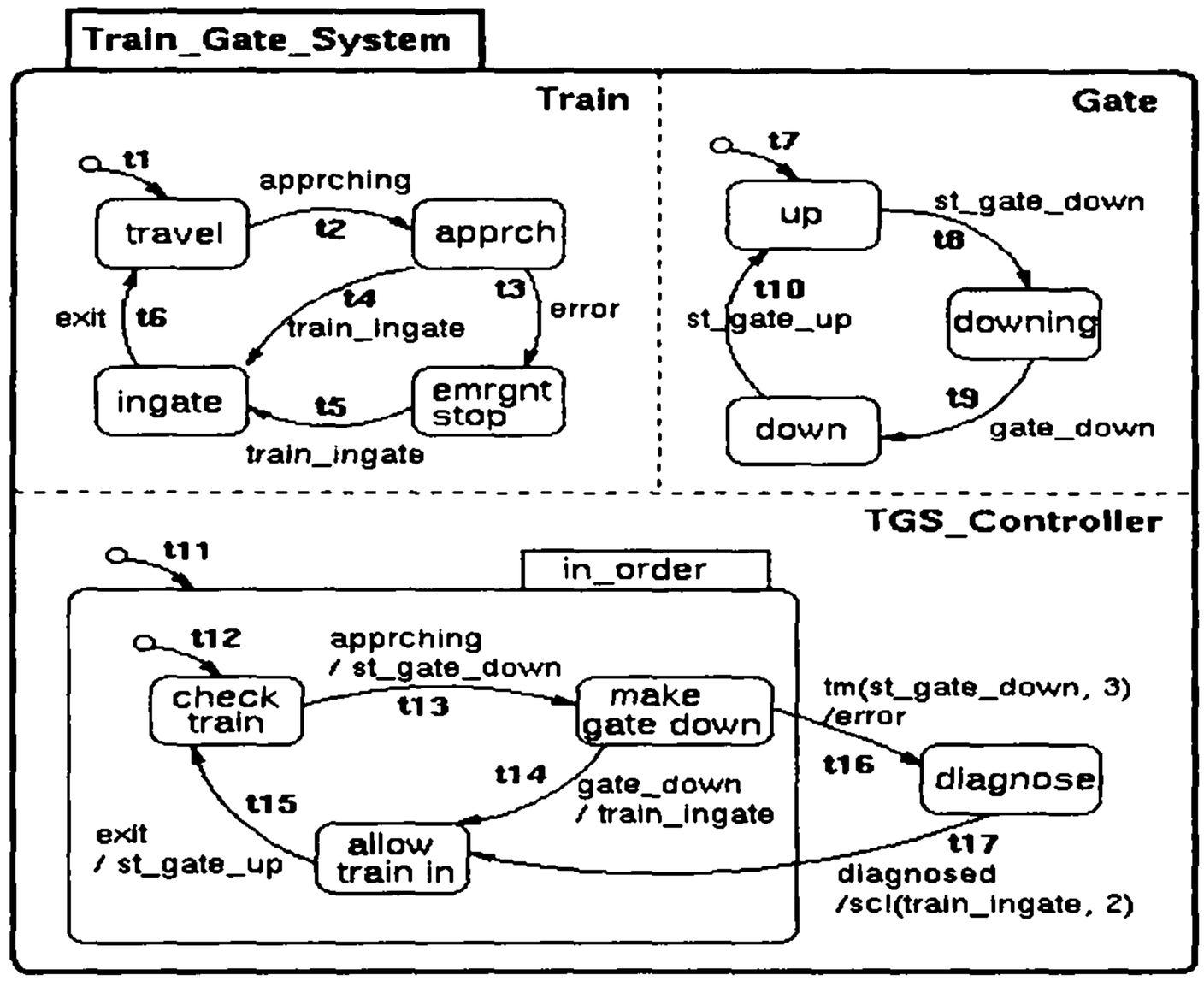
수행해 나가게 된다.

그림 3 (a)의 P3으로 들어가는 D3, D4, 그리고 나오는 D7의 데이터들은 (c)의 같은 이름의 데이터들과 완전히 같은 것이다. 앞서 말했듯 (c)는 (a)의 P3을 상세화한 것이기 때문이다. 예를 들어 P3으로 D3의 데이터가 들어온다는 것은 (c)의 SP3에 D3이 들어온다는 것과 같은 말이다. 따라서 시뮬레이션을 할 때 (a)와 같은 상위 단계의 프로세스에 전해지는 데이터는 그것을 상세화한 그림에 그대로 전해져야 한다. 마찬가지로 상세화된 그림에서 발생한 데이터도 그 상위 단계에 전해져야 한다. ASADAL DFD 모델에서 데이터 흐름(Data Flow)은 각각 하나씩의 큐(Queue; First In First Out)로 설정되어 진다. 즉, 프로세스에서 나오는 데이터는 그 흐름에 저장되고, 그것은 이 큐안에 계속 쌓인다. 그리고 그것은 도착한 순서대로 차례대로 사용하는 측에 전달된다. 일반적으로 프로그래밍 언어에서는 A에서 B로의 데이터 전달이 A가 B에게 전해주거나 B가 A로부터 가져가는 형태를 취한다. ASADAL 모델에서는 이러한 전달을 단순히 큐로 보고 있으므로 이러한 데이터 전달은 A가 큐에 넣고, B가 큐에서 가져가는 형식을 따르게 된다. 이러한 방식은 전달하는 측과 받는 측간의 시간, 공간적인 의존성을 거의 요하지 않으므로 시스템 모델링을 쉽게 만들어준다. 그리고 이러한 모델은 시스템의 큐잉 모델(Queueing Model)에 근거한 분석을 가능하게 해 준다. 이 분석에 대해서는 후에 설명하겠다.

지금까지 설명한 하향식 접근 방법은 실시간 제어 시스템이 복잡한 계산(Computation) 기능과 제어 논리 들을 가지고 있는 경우 그 복잡성을 해결할 수 있는 방향을 제시한다. 하지만, 실제 객체들을 동기화(Synchronization)하는 등 하나의 일을 위해 같이 일할 수 있게 만드

는 생산 시스템(Manufacturing System)같은 응용 분야의 경우는 그 계산이 복잡하지는 않으나 복잡한 제어 논리를 가진다. 이렇게 계산이 TES의 Action에 나타나는 수식적도로 표현이 가능한 경우 TES만으로 명세를 할 수 있다. 예를 들어 그림 2에 있던 철길 건널목 제어 시스템에 대한 명세는 다음과 같이 TES만으로 그려질 수 있다. 참고로, Statechart에서 확장한 TES(Time-enriched Statecharts) 명세는 다음과 같은 시간 관련 요구사항을 명세에 추가할 수 있는데, 이것은 "차단기가 올려져 있는 상태에 있을 때, 기차가 건널목을 지나면 안된다"는 안전성(Safety)를 표현한 것이다.

safety(at(ingate) and at(up))



이제 이러한 명세 구조(Hierarchy)가 어떻게 시뮬레이션 되는지 보

겠다.

다. 하향식 시뮬레이션

ASADAL 명세는 하향식으로 이루어진다. 요구 사항의 명세를 해 나가는 단계에서 아직 덜 상세화된 단계(Gross Level)의 명세를 시뮬레이션하는 것은 조금이라도 빨리 오류나 문제점을 찾으려는 노력을 뒷받침하는 것으로 꼭 필요하다.

ASADAL 명세를 시뮬레이션하면서 완전히 명세되지 않은 요소들이나 외부 개체들의 시뮬레이션은 사용자 대화형의 경우 사용자에 의해 이루어지고 배치의 경우 시뮬레이션 드라이버에 의해 이루어진다.

프로세스가 DFD 그림이나 프로세스 명세로 아직 상세화되지 않았을 때 시 프로세스를 이용한 시뮬레이션의 결과는 아무런 의미있는 시간적 행위(Timing Behavior)나 기능(Action)을 보이지 못한다. 따라서 적어도 이러한 프로세스에 대해서 시뮬레이션 드라이버 프로그램이나 프로세스 명세로 수행 시간이 명세되어야 합리적인 시간적 행위를 보일 수 있다. 예를 들어 우리는 간단히 프로세스에 그것이 얼마의 시간 동안 수행될 것인지를 추계적인 방법으로 명세할 수 있다. 또는 프로세스에 그 프로세스를 시뮬레이션할 때 데이터가 들어오면 얼마만큼의 수행 시간이 지난 후에 어떤 데이터가 나가고 또 어느 정도 지난 후에 어떤 데이터가 나가도록 명세할 수도 있다. 이런 식으로 프로세스 명세에 더 자세한 것을 집어 넣을 수록 더욱 자세한 시뮬레이션이 가능해진다.

이렇게 일단 프로세스 명세를 통해 시뮬레이션해 보았던 프로세스에 대해 충분히 상세화할 준비가 되면 그것을 다른 DFD 그림으로 상세화

하거나 더 자세한 프로세스 명세로 확장하거나 하면 된다.

이상으로 DFD의 상세화에 의한 복잡한 모델의 시뮬레이션에 대한 설명은 마치기로 하고, 이제 이 명세를 어떻게 분산시킬 수 있는 지 알아보도록 하겠다.

라. 분산 명세 방법

분산 시뮬레이션을 하기 위해서는 명세를 분산시킬 수 있어야 한다. 이 분산을 아무렇게나 해서는 소기의 목적을 달성할 수 없기 때문에 분산 방법도 잘 개발하지 않아서는 안된다. 예를 들어 DFD의 어느 정도 수준(Level)에서 프로세스 별로 분산에서 시뮬레이션한다고 하자. 프로세스는 그 기능성을 중심으로 나눈 것이기 때문에 그 통신량이 고려된 것이 아니고 TES는 여러 곳에 분산되어 있는 프로세스들을 제어해야 하기 때문에 그들간 통신에 과부하(Overhead)가 걸려 느려질 수 있고 실제 분산 환경이 그런 식으로 만들어 질 것도 아니기 때문에 시뮬레이션으로 분산환경의 효율이나 성능(Performance)같은 것도 평가하기 어려울 것이다. 따라서,ASADAL에서는 의미 있는 단위로 DFD, TES를 분산하는 방법을 개발하였다.

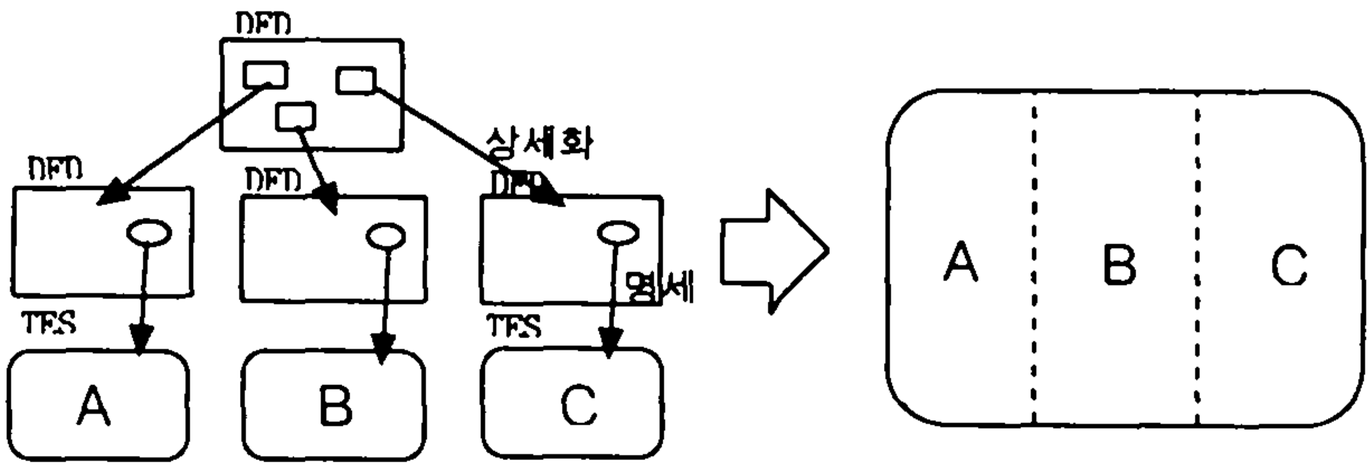
실시간 내장형 제어 소프트웨어(Real-time Embedded Control System)는 외부 환경과의 상호 작용(Interaction)이 아주 많고 복잡하며 궁극적으로 외부 환경을 제어하는 것이 목적이기 때문에 외부 환경에 존재하는 객체들에 대한 이해와 그들과의 상호 작용에 대한 이해가 아주 중요하다. 또한, 설계 과정에 있어서 각 외부 객체와의 상호작용에 정보 은닉(Information Hiding), 책임 중심 설계(Responsibility Driven Design)등의 원칙에 의하여 외부 객체들을 다루는 객체들이 그

소프트웨어 내에 만들어지기 나름이다. 그리고 객체 지향적으로 분석할 경우 다른 객체들과의 상호 연동에 그다지 신경쓰지 않고 그 객체가 언제, 어떤 일을 해야 하며 어떤 서비스를 제공해야 하는지만 생각하면 되기 때문에 명세가 쉬워지는 경향이 있다. 또한 객체 지향적인 방법은 다음 개발 단계인 설계로 넘어갈 때 자연스럽게 소프트웨어 구조등의 설계 모델로 넘어갈 수 있다는 장점도 가지고 있다. 따라서, ASADAL에서는 만들어진 DFD, TES 명세를 객체 지향적인 모델로 바꾼다.

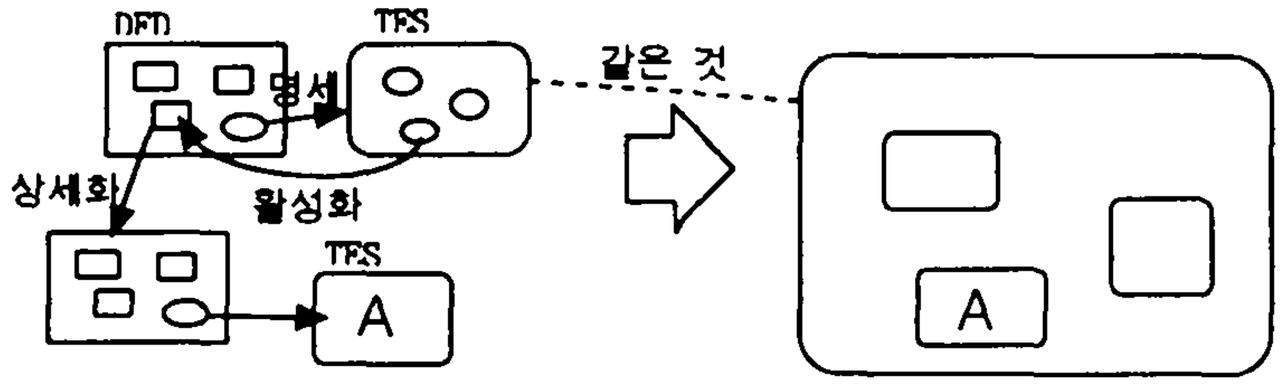
각 DFD에 있는 TES들은 하나의 TES로 쉽게 합쳐진다. 즉, 각 DFD를 제어하는 행위 명세들이 합쳐져서 하나의 행위명세로 될 수 있다. TES의 한 상태가 다른 상태 안에 존재할 수 있게 하는 Higraph 표현방법은 이것을 가능하게 한다.

다음과 같은 지침들에 따라 하나의 TES로 만들어진다.

- TES가 없는 DFD안의 프로세스를 상세화한 DFD를 제어하는 TES들은 병렬로 합친다.



- 어떤 프로세스의 DFD 안에 있는 프로세스들을 제어하는 TES 명세는 그 프로세스를 활성화하는 상태의 안으로 넣는다.



- 순차적으로 활성되는 프로세스들의 TES 명세들은 순차적으로 연결하되 앞선 프로세스 TES의 종결자(Terminator)와 다음 프로세스의 TES 초기 상태(Initial State)를 상태 전이로 연결한다.

이제 객체를 찾는다. 객체가 찾아지면 각 객체에 상태 변화를 할당하고 필요한 기능을 넣을 것이며 데이터를 찾을 것이다. 객체는 만들어진 TES의 병렬 상태들을 적당히 나눔으로써 가능하다. 실시간 제어 시스템의 경우는 외부 환경과 밀접하게 연결되기 때문에 외부 객체들을 다루는 소프트웨어 객체들을 만드는 것이 좋다. 객체로 나누어지지 않아도 시뮬레이션의 속도 상승을 위해 병렬 상태들을 적당히 여러 개로 나눌 수 있다.

이렇게 나누어진 명세는 여러 시뮬레이터에 나뉘어 시뮬레이션된다. 경우에 따라 시뮬레이터는 한 컴퓨터에서 또는 여러 컴퓨터에서 동시에 사용가능하다.

그럼 이제 나누어진 명세가 각각의 데이터베이스에 들어있는 상태에서 분산 시뮬레이션 실행 방법을 보겠다.

1. 동기 제어기 "ASDsynchronizer(ASADAL 동기 제어기의 실행 화일)"를 동작시킨다.

2. 시뮬레이터들을 동작시키고 명세를 각각 읽어들인다. 시뮬레이터들은 기동될 때 동기 제어기 존재 여부를 확인, 있는 경우 분산 시뮬레이션 모드로 들어간다. 동기 제어기가 있는 경우 하나의 제어판으로 모든 시뮬레이터들을 동시에 제어할 수 있다.

동기 제어기와 시뮬레이터들은 네트웍으로 연결된다. 동기 제어기는 잘 알려진 포트(Well-known Port)에 시뮬레이터들이 연결하기를 기다리고 있으며 일단 시뮬레이터들이 연결되면 그들에게 분산 시뮬레이션을 할 것이라는 것을 알린다.

동기 제어기에 의해 분산 시뮬레이션이 일어나는 방식에 대해 알아보자.

동기 제어기는 먼저 시간 및 단계(Step)을 동기화한다. 각 시뮬레이터에서 일어나는 시간의 증가 및 단계의 증가는 모두 동기화되지 않으면 안되며 각각의 시뮬레이터에서 진행시키던 시뮬레이션을 이제는 모두 동기 제어기가 맡는다. 즉, 각각의 시뮬레이터에서는 이제 임의로 시간이나 단계를 증가시키지 않고 동기 제어기가 이들을 동시에 증가시킨다.

시뮬레이터들은 시간이나 단계의 증가를 동기 제어기가 요구할 때 그 증가에 맞게 시뮬레이션하고 각 단계별로 자신이 가지고 있는 모든 시뮬레이션 정보를 동기 제어기에 보낸다. 동기 제어기로 보내진 정보들은 보낸 시뮬레이터를 뺀 다른 시뮬레이터들로 보내진다. 동기 제어

기로부터 이렇게 넘어 온 정보를 이용하여 시뮬레이터는 자신이 가지고 있는 정보를 갱신한다.

동기 제어기가 전달하는 시뮬레이션 정보에는 다음과 같은 것들이 있다.

- 사건의 발생: 사건에는 어떤 상태로 들어감, 어떤 상태에서 나옴, 데이터 흐름의 발생, 어떤 프로세스의 끝남(Termination), 데이터나 조건 값의 변화등 ASADAL에서 정의하고 있는 자체(Built-in) 사건들과 사용자가 마음대로 정의하여 사용하는 사건들이 있다.

- 변한 데이터나 조건의 값: 데이터나 조건의 값이 변했을 때 그것은 동기 제어기를 통해 다른 시뮬레이터로 전달된다. 이 때 동기 제어기는 한 단계에서 데이터 값이 두 번 이상 변화하는 경우 Racing 오류를 발생시킨다. 이러한 Racing은 다음과 같이 동시에 발생한 상태 전이가 Action으로 같은 데이터나 조건의 값을 변화시켰을 때 발생하며 어떤 것이 먼저 처리되었으냐에 따라 그림의 amount의 값을 3으로도, 5로도 만들게 되므로 이러한 경우는 없어야 한다.

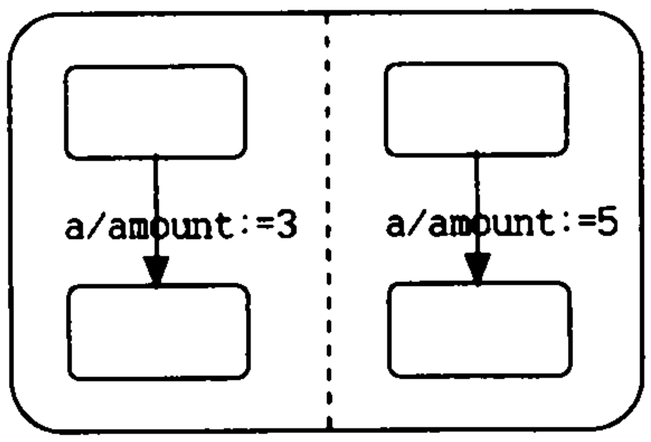


그림 8은 두 가지 음료를 파는 자판기 시스템을 시뮬레이션하는

ASADAL 시뮬레이터의 실행모습이다. 하나의 시뮬레이터엔 그림과 같이 DFD, TES등이 동적으로 시뮬레이션되는 모습을 볼 수 있는 화면과 제어판, 그리고 데이터 값을 볼 수 있는 창과 사건을 발생시키거나 데이터 값을 변화시킬 수 있는 창이 있다.

이것과 같은 TES를 분산해서 시뮬레이션한 예가 그림 9에 있다. 이 그림은 그림 8의 TES에서 사용자가 돈을 넣고, 버튼에 불(Lamp)이 들어오며 음료수가 나오는 부분, 즉 사용자에게 보이는 것을 다루는 부분과 내부적으로 음료수를 내보낼 것인가 안내보낼 것인가를 결정하고 운

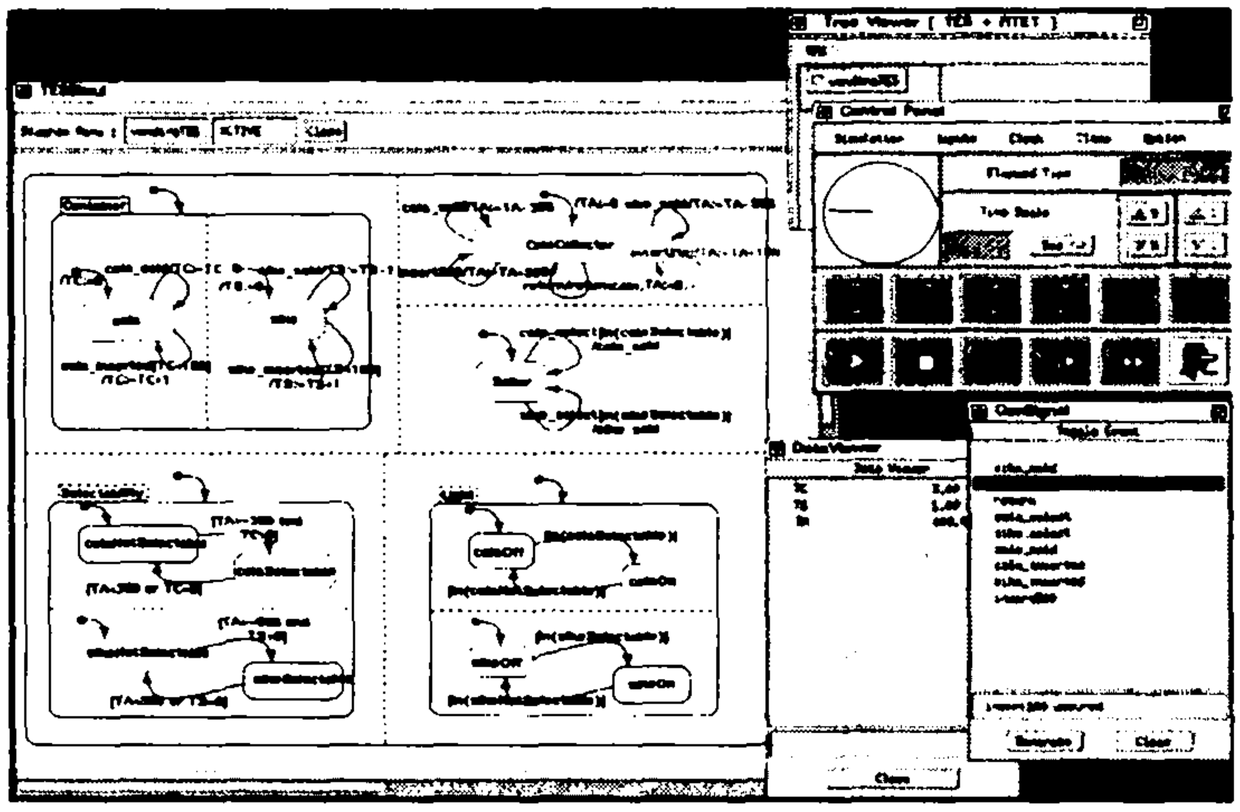


그림 8 자판기 시스템 시뮬레이션 예

영자가 음료수를 더 집어 넣는 부분의 두 가지로 크게 나누었을 때 사용자에 가까운 부분의 명세를 시뮬레이션하는 그림이다.

나머지 부분의 명세 시뮬레이션은 그림 10과 같으며 이 두 시뮬레이션은 하나의 컴퓨터에서도, 두 개의 컴퓨터에서도 실행가능하다. 두 시뮬레이션이 모두 제어기(Control Panel)을 가지고 있으며 둘 중 아무 것으로도 제어가 가능하다. 즉, 제어기를 사용하면 그 정보가 동기

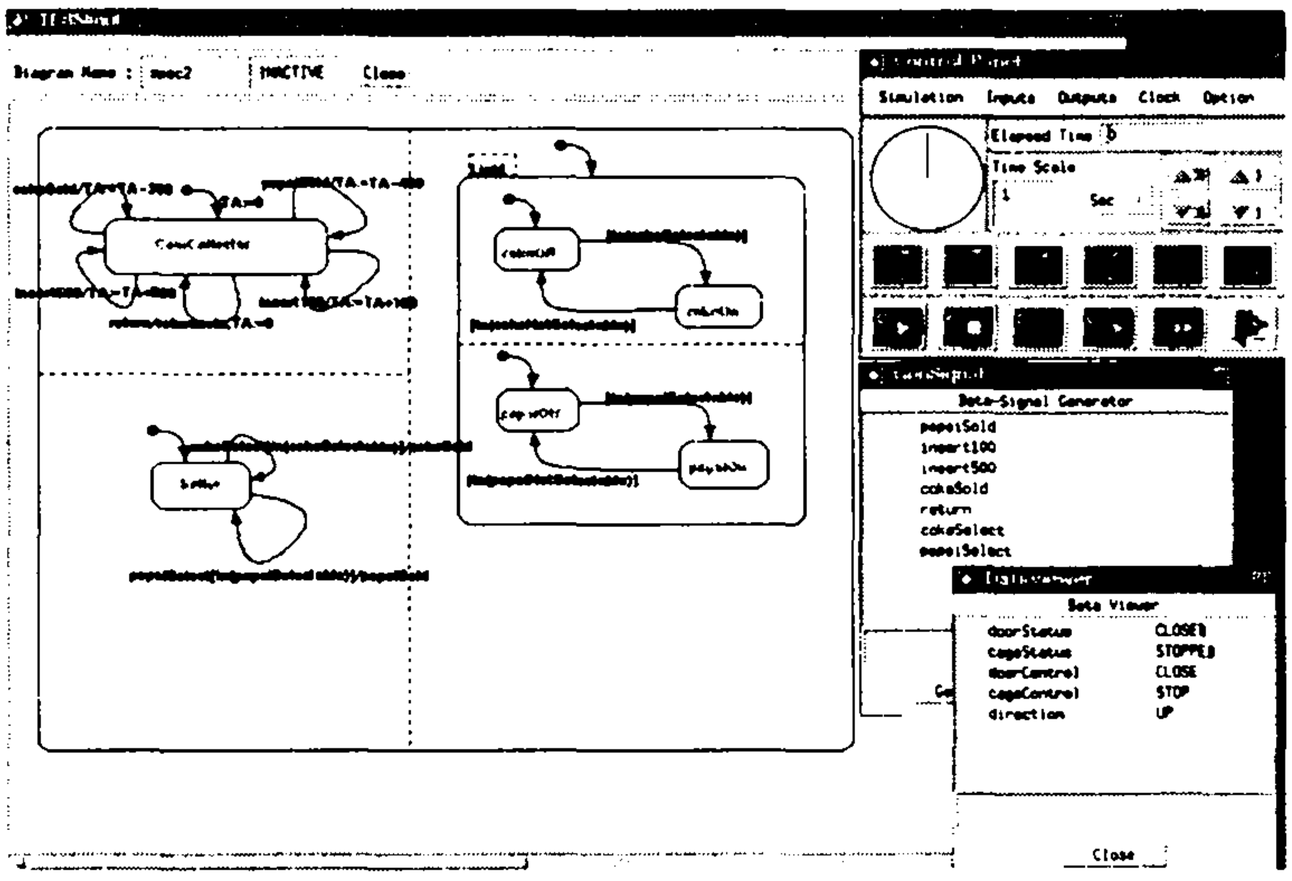


그림 9 자판기 시스템 분산 시뮬레이션 예 중의 일부

제어기로 전달된다. 분산 시뮬레이션에서는 단계와 시간 제어를 모두 동기 제어기가 하므로 이 정보로 모든 시뮬레이터에 제어 정보를 전함으로써 모두 동시에 제어된다.

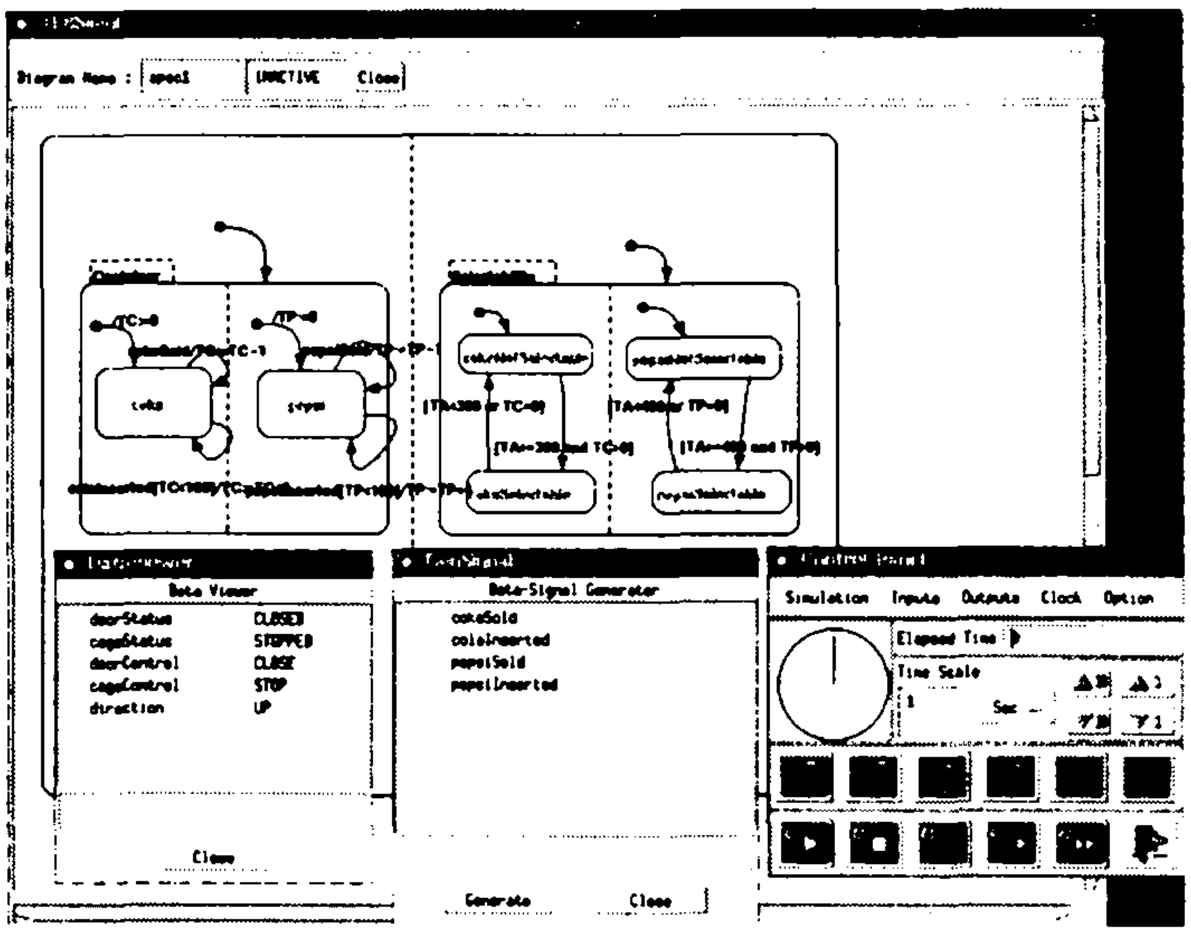


그림 10 자판기 시스템 분산 명세 - 그림 9의 나머지부분

분산 시뮬레이션의 또 하나 장점은 시뮬레이션시 시스템 사용자들이 실제 상황처럼 움직여 볼 수 있다는 것이다. 즉, 이 경우 두 사람이 각각 다음 컴퓨터(시뮬레이터)에서 운영자, 사용자 역할을 하면서 시스템이 시뮬레이션되는 보습을 볼 수 있다. 이러한 기능은 시스템이 복잡해지고 많은 사람들이 연관되어 있을 때 필수적이다.

2. 데이터 분석기(Data Analyzer)

가. 시뮬레이션 결과물

배치 시뮬레이션을 하면 그 결과로 먼저 완전한 시간에 따른 시스템 상황 변화에 대한 기록 파일(Log File)을 얻을 수 있다. 이 파일엔 다음과 같은 내용들이 발생 시각과 함께 저장되어 있다.

· 사건의 발생

· 프로세스의 활성화, 비활성화

· 데이터, 제어 시그널의 발생, 그 값

· 데이터의 사용

· 조건 값의 변화

· 어떠한 상태에 들어간 것, 나간 것

기록 화일은 세 가지가 동시에 만들어지며 그들은 각각 데이터 파일, 프로세스 파일, 상태 파일이다.

데이터 파일에는 데이터가 큐(데이터 흐름)에 들어가거나 나갈 때, 또는 데이터의 값이 변할 때 더해지며 다음과 같은 ASCII File로 되어 있다.

```
destinationFloor reachedFloor
destinationFloor    En  1   8
reachedFloor        En  1   4
reachedFloor        Ex  2   3
reachedFloor        En  3   2
reachedFloor        En  4   1
reachedFloor        Ex  5   0
destinationFloor    Ex  5   8
```

첫 줄은 명세에 존재하는 데이터들을 나열하고 그 다음 줄부터는 데이터가 큐에 들어갈 때는 둘 째 열을 “En”으로 한 데이터 이름과 발생 시각, 그 때의 값이 더해진다. 큐에서 나갈때는 둘 째 열이 "Ex"가 된다. TES의 Action 영역에서와 같이 큐에 들어가거나 나가는 것이 아닌

값이 바뀌는 경우는 둘 째 열이 "Ch"가 된다.

프로세스 파일에는 프로세스가 활성화, 비활성화되는 정보가 다음과 같이 저장된다.

```
contol_door open_cage close_cage control_lamp
control_door        Ac  1
open_cage           Ac  1
close_cage          Ac  2
open_cage           De  4
```

이 파일의 첫 줄에는 프로세스들이 나열되고 그 다음 줄부터는 프로세스가 활성화(Activate), 비 활성화(Deactivate)될 때 두번째 열을 각각 "Ac", "De"로 하여 기록한다. 세 번째 열은 그 시각이 기록된다.

상태 파일에는 상태가 현재상태로 들어오고(Enter), 나가는(Exit) 것을 다음과 같이 기록한다.

```
light_on light_off door_opening door_closing door_closed
    door_opened cage_stopped cage_moving_up
    cage_moving_down
light_off door_closed cage_stopped
light_off     Ex   3
light_on      En   3
door_closed   Ex   3
door_opening  En   3
door_opening  Ex   5
door_opened   En   5
```

상태 파일에는 먼저 모든 상태들이 나열된다. 그 다음엔 이들중에서

초기 상태들이 나열된다. 그 다음부터는 상태로 들어가거나 나갈 때 그 시각과 함께 기록된다.

나. 시뮬레이션 결과 분석

시뮬레이션이란 일정 시간 동안 시스템의 모형을 실행시켜 보고 그 결과들을 바탕으로 시스템의 궁극적 행동(Long-Run Behavior)을 예측해 보는 것이다. 그러므로, 시뮬레이션 결과를 분석하여 의미를 추출해 내지 않고 끝낸다면 그 시뮬레이션은 별로 유용하지 않을 것이다. 그래서, ASADAL 시뮬레이터는 시뮬레이션 결과를 통계적이고 추계적으로 분석해 주고 그 의미를 그래픽컬하게 보여 주는 도구인 데이터 분석기(Data Analyzer)를 제공한다.

이 분석 도구는 시뮬레이션 도중에는 데이터 값, 프로세스의 서비스 시간, 데이터가 큐에서 기다리는 시간 등의 평균, 분산의 변화를 그래프를 통해 동적으로 보여 준다. 그리고, 시뮬레이션이 끝나면 수행된 동안 나타난 시스템의 결과를 바탕으로 예측된 최종의 결과를 보여 준다.

다. 분석 내용

ASADAL 시뮬레이터에서 TES명세와 DFD명세의 시뮬레이션은 TES명세에 의해 시스템의 상태(State)의 전이가 일어나며 그 때 각 상태에 명시되어 있는 DFD명세의 프로세스가 활성화 되어 정해진 일을 하고 비활성화 되면서 진행된다.

시뮬레이션 도중 각 데이터나 제어 신호는 특정 확률 분포 함수에

따라 프로세스가 가지고 있는 큐로 들어오며, 큐에 쌓여 있던 데이터는 기본 프로세스가 수행될 때마다 하나씩 소비된다. 큐는 기본 프로세스의 특성에 따라 길이가 다르며, First-Come First-Served방식이다. 기본 프로세스는 서비스를 받으러 들어 오는 데이터에 대해서 서비스를 제공한 후 서비스가 끝나면 내보내며 프로세스가 데이터를 서비스하는 시간은 일정하거나 특정 확률 분포를 따르게 한다.

이렇게 시뮬레이션이 진행되는 동안 어떤 데이터가 언제 어떤 프로세스의 큐에 들어갔고 언제 프로세스에 의해 서비스 받았으며 언제 프로세스 밖으로 나왔는지 그 값이 무엇인지를 앞서 언급한 기록 파일에 적는다.

이 기록 파일로부터 다음과 같은 것들을 분석할 수 있다.

· 데이터 값 분석

각 데이터나 제어 신호에 대해서 현재 어떤 값이 들어오고 있는지를 보여 준다. 그리고, 시뮬레이션 도중 현재까지의 값의 평균과 분산의 동적인 변화를 그래프를 통해 보여 준다.

· 큐 길이 분석

큐에서 서비스를 기다리는 데이터의 개수의 평균과 분산을 구하여 그래프로 보여 준다.

· 서비스 시간 분석

각 데이터에 대해서 프로세스에 들어가서 서비스를 받고 프로세스를

나가는데 걸리는 시간의 평균과 분산을 계산하여 그래프로 동적인 변화를 보여 준다.

· 기다리는 시간 분석

각 데이터에 대해서 큐에 들어와서 프로세스에 들어가기 전까지 큐에서 머무는 시간의 평균과 분산을 계산하여 그래프로 동적인 변화를 보여 준다.

· 상태의 활성 비율

시뮬레이션 동안 각 상태에 있는 비율이 얼마나 되는지를 계산하여 보여 준다. 그리고, 시뮬레이션 결과를 바탕으로 상태의 전이 확률을 계산하고 Markov Chain을 이용하여 궁극적으로 어떤 상태에 있을 확률이 얼마나 되는가를 계산한다.

· 프로세스의 Busy/Idle Proportion

프로세스가 시뮬레이션 도중 얼마나 일을 하고 얼마나 하지 않았는지를 계산하여 보여 준다. 이 결과로부터 프로세스가 대부분의 시간을 서비스했다면 프로세스를 하나 늘리는 것을 고려해야 할 것이다.

· 시간 제약 검사(Time Constraint Checking)

MSD 명세에는 어떤 사건이 일어나고 그 다음에 어떤 사건은 얼마의 시간 안에 일어나야 한다는 시간 제약 등을 명세할 수 있다. 그래서, 분석 도구는 MSD 명세에 나타나는 두 사건 사이의 시간 제약들이 시뮬레이션 도중에 잘 맞는지를 계산해 준다. 그리고, 시뮬레이션이 끝나

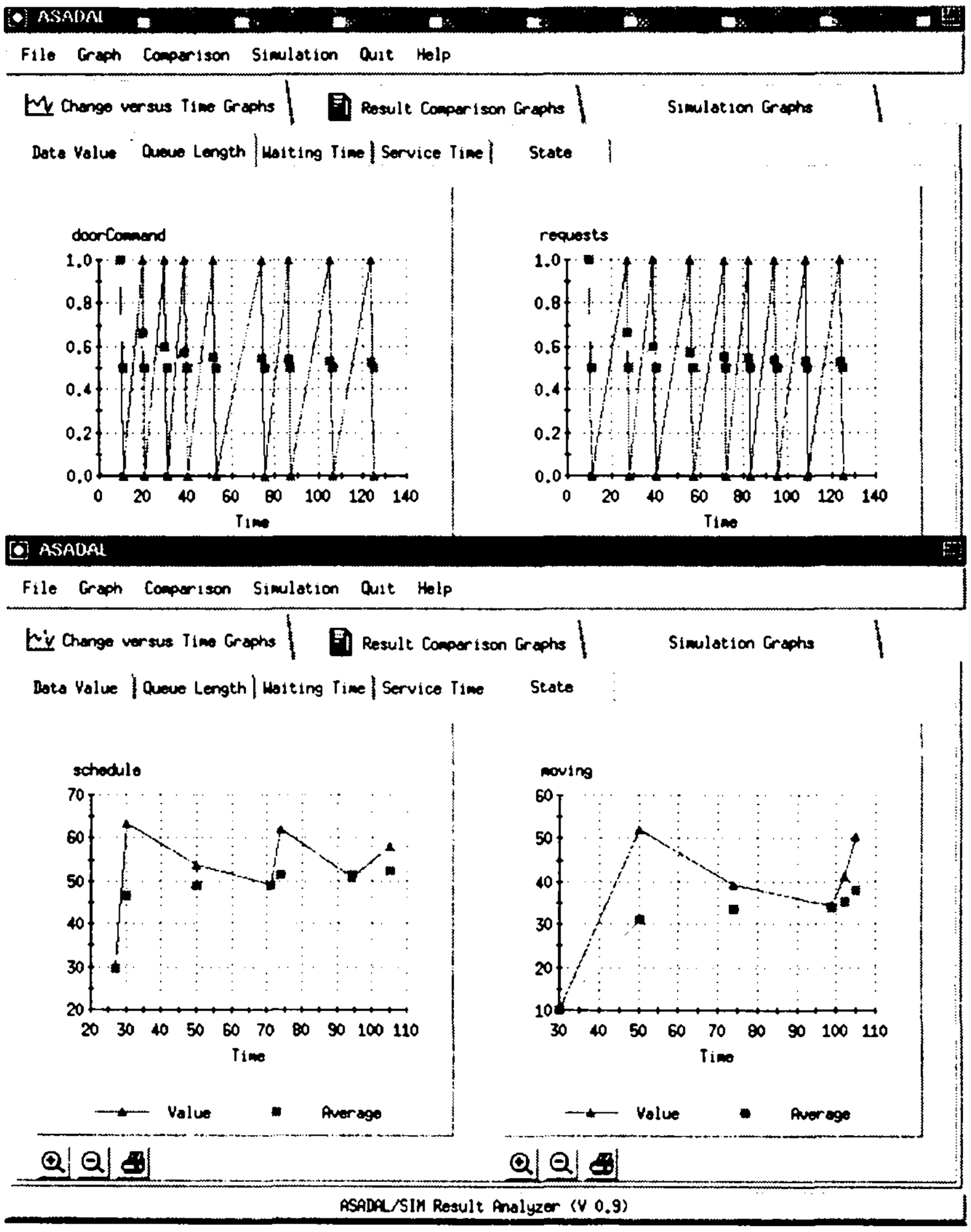


그림 11 데이터 분석기 - 큐 길이 및 상태 도달 분석

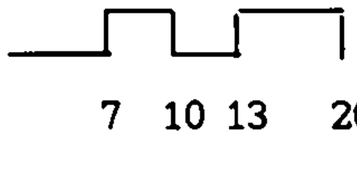
고 나서는 시간 제약을 지키는 데 대한 신뢰도를 계산해서 보여 준다.

그림 11은 시간의 변화에 따른 큐 길이와 어떤 상태에 머무른 비율을 표시가 그래프이다.

그림의 위쪽의 두 개의 그래프는 “door_command”와 ”request” 두 개

의 DFD 데이터흐름에 대하여 데이터가 큐에 들어오고 나감으로써 발생하는 큐 길이의 변화를 보이고 있다. 선은 두 개씩 나타나는데 하나는 값의 변화로서 예를 들어 왼쪽 위 그래프에서 큐의 길이가 1과 0 사이에서 반복되는 것을 보이고 있고 또 다른 하나는 평균값의 변화로서 대략 평균적으로 0.45정도의 큐 길이고 수렴하는 것을 볼 수 있다.

아래쪽의 두 그래프는 어떤 상태에 들어 있는 시간의 비율(Proportion of Time in a State)을 나타낸 것이다. 이 그림에서는 각각 "Schedule"과 "moving" 상태에 들어 있는 비율을 보이고 있다. 이 값은 어떤 상태에서 나가고 다시 나가는 시간 중 들어있었던 시간의 비율이다. 예를 들어 다음과 같이 어떤 상태에 들어 가고 나갔다고 하자.



위로 올라간 것이 어떤 상태에 있는 것, 내려간 것이 그 상태를 나온 것이라면 10의 시각에 0.3의 비율로 그 상태에 있었으며 데이터 분석기 그래프에 10의 시각에 0.3이 찍힌다. 마찬가지로 20에 0.7이 찍힌다. 평균은 10초에 10*0.3/10=0.3, 20초에 (0.3*10+0.7*(20-10))/20 =10/20=0.5와 같이 계산된다.

그림 13은 데이터 값의 변화를 보인 그래프이다. 사용자는 이 그래프에서 여러 개의 데이터들을 한 그래프에 보일 수도 있고 이들의 통계값을 같이 볼 수도 있다. 이 그래프에서 목적층(DestinationFloor)으로 엘리베이터가 움직여 가는 것을 도달층(reachedFloor)의 변화로 볼 수 있다.

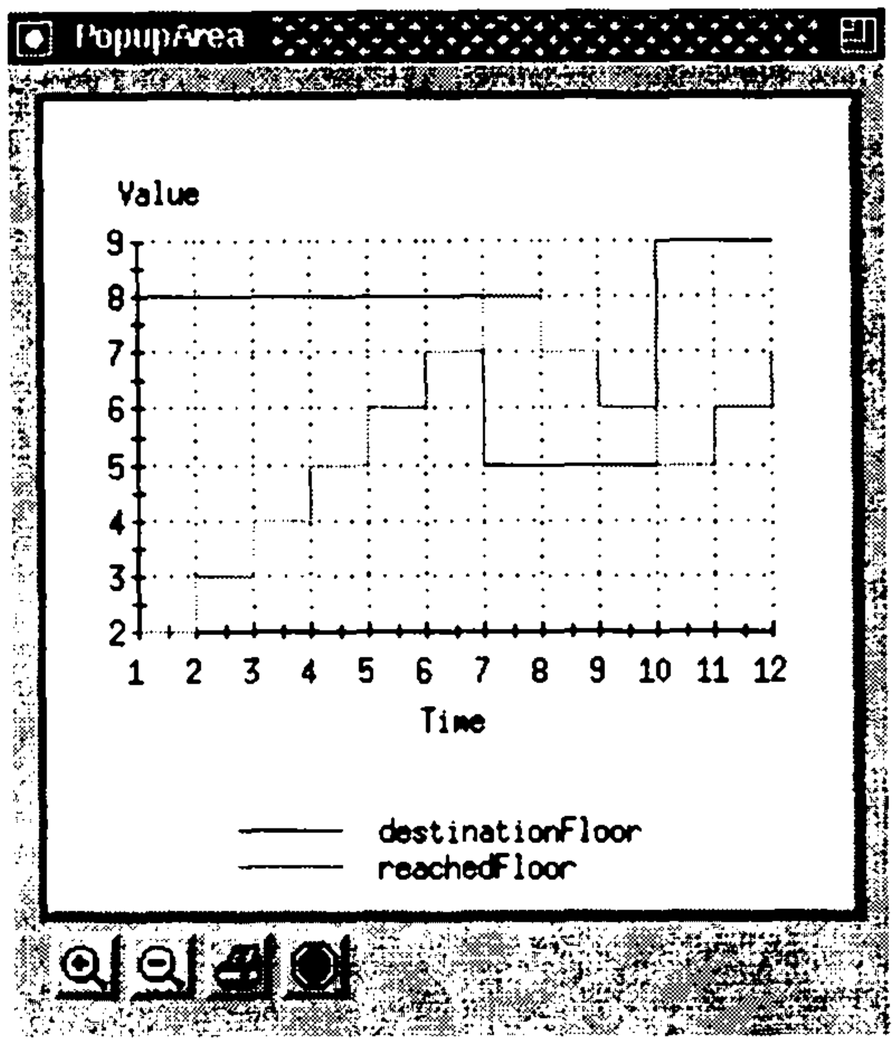


그림 13 데이터 분석기- 데이터 값의 변화 분석

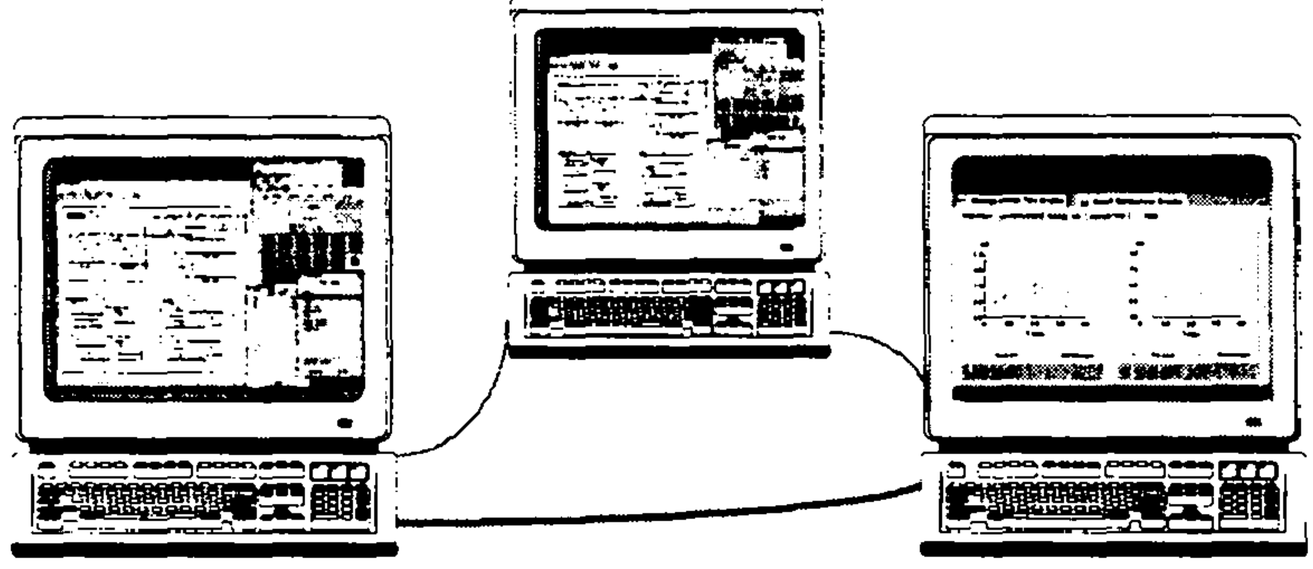


그림 14 분산 시뮬레이션 및 데이터 분석기

데이터 분석기는 시뮬레이터와는 서로 다른 실행 파일로 만들어져 있으며 언제든 시뮬레이터가 만들어 낸 기록 파일을 분석해낼 수 있다. 데이터 분석기는 한 가지 모드를 더 가지고 있는데 그것은 온라인 분석이다. 이것은 시뮬레이션을 사용자 대화형으로 하면서 바로 그 출력을 데이터 분석기에 연결하여 동적으로 그래프의 변화를 보여주는 방식을

말한다. 이 경우 데이터 분석기와 시뮬레이터는 TCP/IP 네트웍으로 연결된다. 까라서, 결과적으로 일반적인 ASADAL 시뮬레이션은 그림 14와 같은 형태로 이루어진다.

3. 테스트, 분석 방법

ASADAL 시뮬레이터가 지원하는 여러 테스팅 방법들에 대해 알아보겠다. 모든 검사는 시뮬레이션 후에 시뮬레이견 기록 파일에서 이루어질 수도, 시뮬레이션과 동시에(On-line) 이루어 질 수도 있다.

가. 도달성 검사(Reachability Test)

도달성 검사는 어떤 상태에 시스템이 도달했는지를 검사한다. 이 검사는 단순히 어떤 상태 하나에 도달했는지 뿐 아니라 이들이 섞인 형태로, 즉 이떤 상태와 어떤 상태에 동시에 들어간 것이 있었느냐 등을 테스트 할 수 있다. 이 기능은 발생해서는 안되는 상태를 시스템이 들어간 적이 있었느냐를 테스팅하는 데 쓰일 수 있다.

나. 전이 사용성 검사(Use-Transition Test)

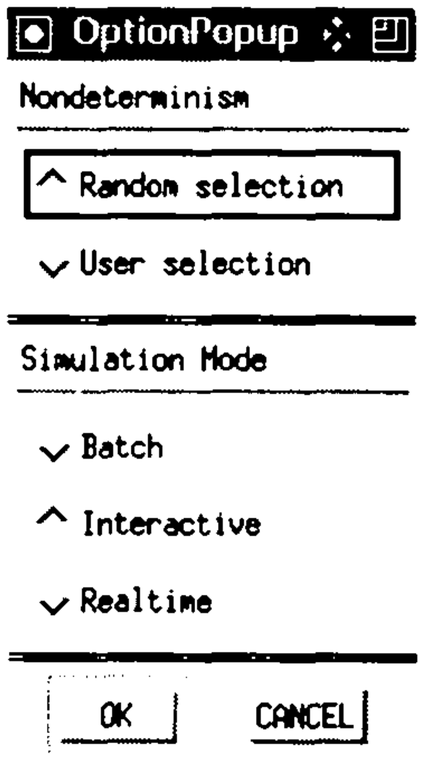
TES에 있는 전이들이 시스템 동작시 실제로 사용이 되었는지에 대한 검사로서 사용되지 않은 전이가 분석됨은 물론 전이의 사용 빈도가 분석됨으로써 주(Major) 상태 전이 경향이 나타난다. 사용되지 않은 전이는 명세의 잘못이나 시스템 오류를 찾는데 도움이 된다.

다. 비결정성 검사(Nondeterminism)

사용자 대화형 시뮬레이션 도중 비결정성이 발생하면 시뮬레이터는

두 가지 방법중 하나로 그것을 해결한다. 하나는 시뮬레이션을 중단하고 사용자에세 그 비결정성을 어떻게 해결할 것인지를 묻는 것이고 다른 하나는 임의로 아무 것이나 선택하여 진행하는 것이다.

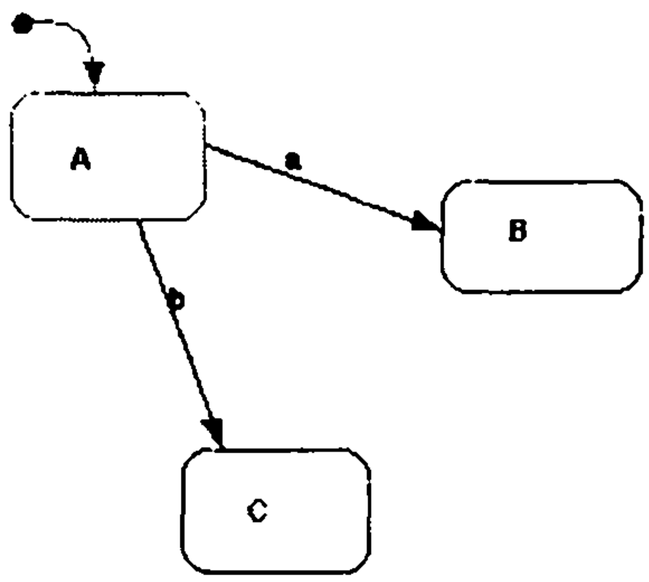
이 두가지중 하나를 선택하는 것은 다음과 같은 제어판(Control Panel) "Option" 메뉴를 선택할 때 나타는 창에서 선택 가능하다.



이 그림의 위쪽엔 비 결정성을 어떤 방식으로 해결하겠는지가 있다. 위의 것을 선택하면 시뮬레이터가 알아서 아무 것이나 Random하게 선택하고 그 사실을 기록 파일에 남기고 시뮬레이션을 계속 진행한다. 하지만 아래의 것을 선택하면 비결정성이 발생했을 때 다음과 같은 창이 나온다.

이 것은 명세가 다음과 같은 경우 나오는 창이다.

이 명세에서 현재 상태가 A이고 사건 "a"와 "b"가 동시에 발생했을 경우 비결정성이 생기며 이전의 창이 보이는 것이다. 사용자는 그 창에서 두 개의 상태 전이중 하나를 선택함으로써 계속 시뮬레이션을 진행



시킬 수 있다. 물론, 창의 아래 오른쪽 버튼으로 임의로 하나가 선택되게 할 수도 있다.

시뮬레이션이 사용자 대화형이 아니고 배치모드로 진행중일 때는 사용자에게 선택권을 주지 않고 자동으로 임의 산택하여 진행된다. 물론 사용자는 시뮬레이션이 끝난 후 그러한 일이 있었다는 것을 알게 된다.

4. 자동 두께 조절 시스템 및 장력 제어 시스템 - 지능형 생산공장 예제

자동 두께 조절 시스템(Automatic Gauge Control System)은 두 개의 좁은 롤 사이를 철판이 통과하게 함으로써 두께를 얇게 만드는 압연 공정에 있어서 두 롤의 간격을 실시간으로 효과적으로 조절함으로써 원하는 두께의 철판이 나오게 하는 시스템을 말한다. 그리고 장력 제어 시스템(Speed Controller)은 양쪽 릴(Reel)과 가운데의 롤 사이를 움직이는 철판이 정해진 장력을 유지하도록 양쪽 릴의 속도를 조종함으로써 철판이 끊어지지 않게 해한다.

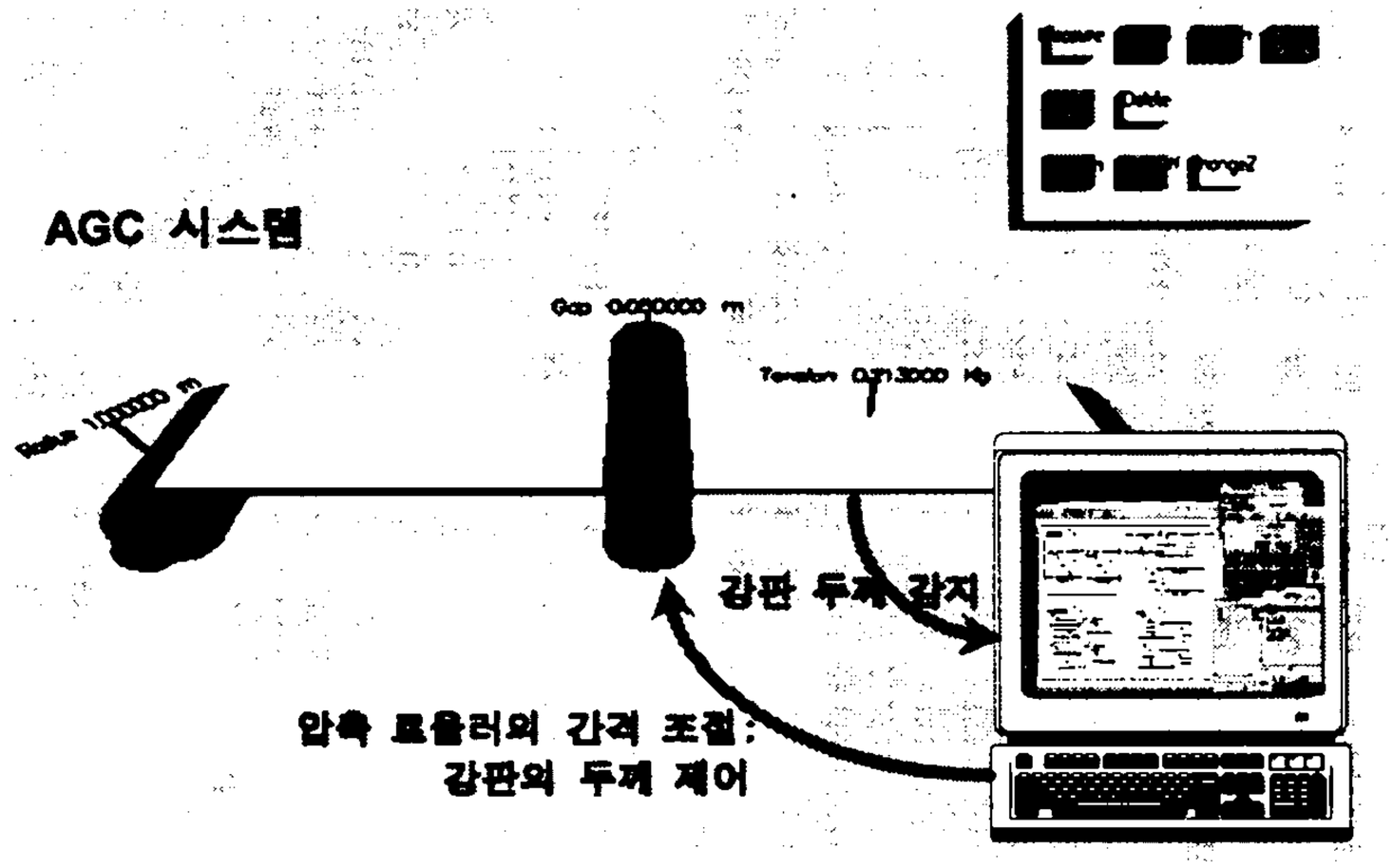


그림 18 AGC 시뮬레이션 환경

그림 18은 AGC의 시뮬레이션 환경이다. 시뮬레이터는 만들고자 하는 제어 시스템인 AGC뿐 아니라 물리적인 환경도 시뮬레이션해야 하며 이들이 맞물려 잘 돌아가는지가 테스팅 가능해야 한다.

이러한 AGC를 시뮬레이션함으로써 얻을 수 있는 효과는 다음과 같다.

- 여러 AGC 알고리즘들을 시뮬레이션을 통해 비교, 분석함으로써 최적의 것을 선택

  알고리즘 프로세스 명세를 바꿈으로서 쉽게 여러 알고리즘들을 적용했을 경우의 성능이나 생산된 제품의 질등을 분석, 최적의 것을 찾을 수 있다.

- 시스템 성능에 대한 알고리즘 인자들의 영향 조사

  알고리즘뿐 아니라 알고리즘에서 사용하는 여러 인자(Paramter)

들을 변화시켜가며 그들이 주는 영향을 분석, 최적의 인자들을 찾을 수 있다.

- 시간 제약의 검사

시뮬레이션을 통해 시간 제약이 지켜지고 있는 지를 검사할 수 있다. 이것은 시스템의 성능(Performance)에 대한 것 뿐 아니라 여러 가지 발생 가능한 에러 상황에 대한 반응 속도같은 것에 대한 정밀한 검사가 가능하다.

- 시스템 개발 초기에 시스템의 행동 검사

시스템이 어떤 방식으로 동작할 것인지를 보는 것은 아주 중요하다. 대부분의 경우 실시간 제어 시스템은 실세계에 대한 깊은 이해를 필요로하기 때문에 개발 초기에 시스템 동작 모습을 미리 볼 수 있게 해주는 시뮬레이션은 시스템에 대한 개발자의 이해를 도울뿐 아니라 개발자, 사용자들의 이해를 일관성있게 해 준다.

- 시스템의 구성 방법(Configuration)을 다양하게 쉽게 바꿀 수 있다. 시뮬레이션이나 테스트 등의 결과에 의해 시스템 구성은 다양한 형태로 바뀌게 된다. 따라서 이러한 변화를 쉽게 수용하고 다시 시뮬레이션할 수 있는 환경이 절실하며 ASADAL 시뮬레이터사 그것을 지원하고 있는 것이다.

그림 19는 AGC 시뮬레이터의 구조를 보이고 있다. 이것은 단일 스탠드 AGC, SC로서 실제 환경(Real World)을 시뮬레이션함과 동시에 AGC와 SC를 시뮬레이션하고 이들간의 상호작용을 네트웍을 통해 교환함으

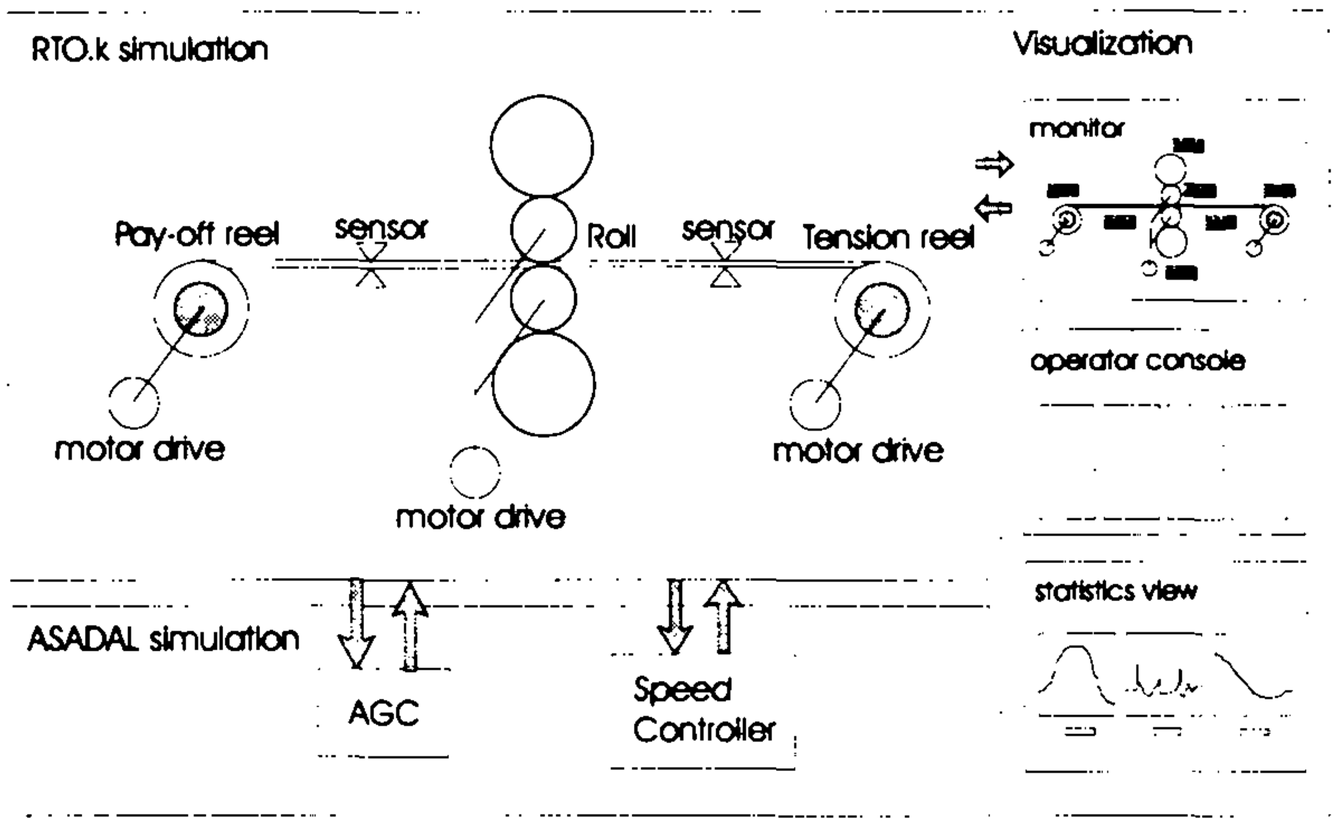


그림 19 AGC 시뮬레이터의 구조

로써 실제와 같이 시뮬레이션한다. 이 때 Visualization 환경은 압연 공장이 돌아가는 모습을 보여주고 작업자들에게 제공되는 제어 파넬 들을 실제와 같게 제공, 실제와 같은 상황에서 시뮬레이션할 수 있게 하고 데이터들의 값이나 통계 값 등을 분석하고 그래프화하여 보여준다.

그림 20은 시뮬레이션 도중 실제 환경을 볼 수 있는 Visualization 도구를 보이고 있다. 이 도구는 실제 환경을 3차원으로 모델링하여 사용자로 하여금 가상 현실 도구를 이용하여 그 환경을 돌아다닐(Navigate)할 수 있게 해 주고 환경에 관계된 여러 값들, 예를 들어 압연기같은 경우 롤에 들어가고 나가는 철판의 두께 같은 것들이 시뮬레이션 도중 변하는 모습을 계속적으로 보여준다. 물론, 그 값에 따라 실제 모양도 계속 변화한다.

3차년도에 만들어진 지능형 생산 공장 모델은 이 AGC 및 SC를 다중 스탠드로 확장한 것이었다. 즉, 롤이 하나가 아닌 여러 개일 경우 이들을 제어하는 AGC 및 SC도 여러 개가 있게 되고 이렇게 될 경우 제어기

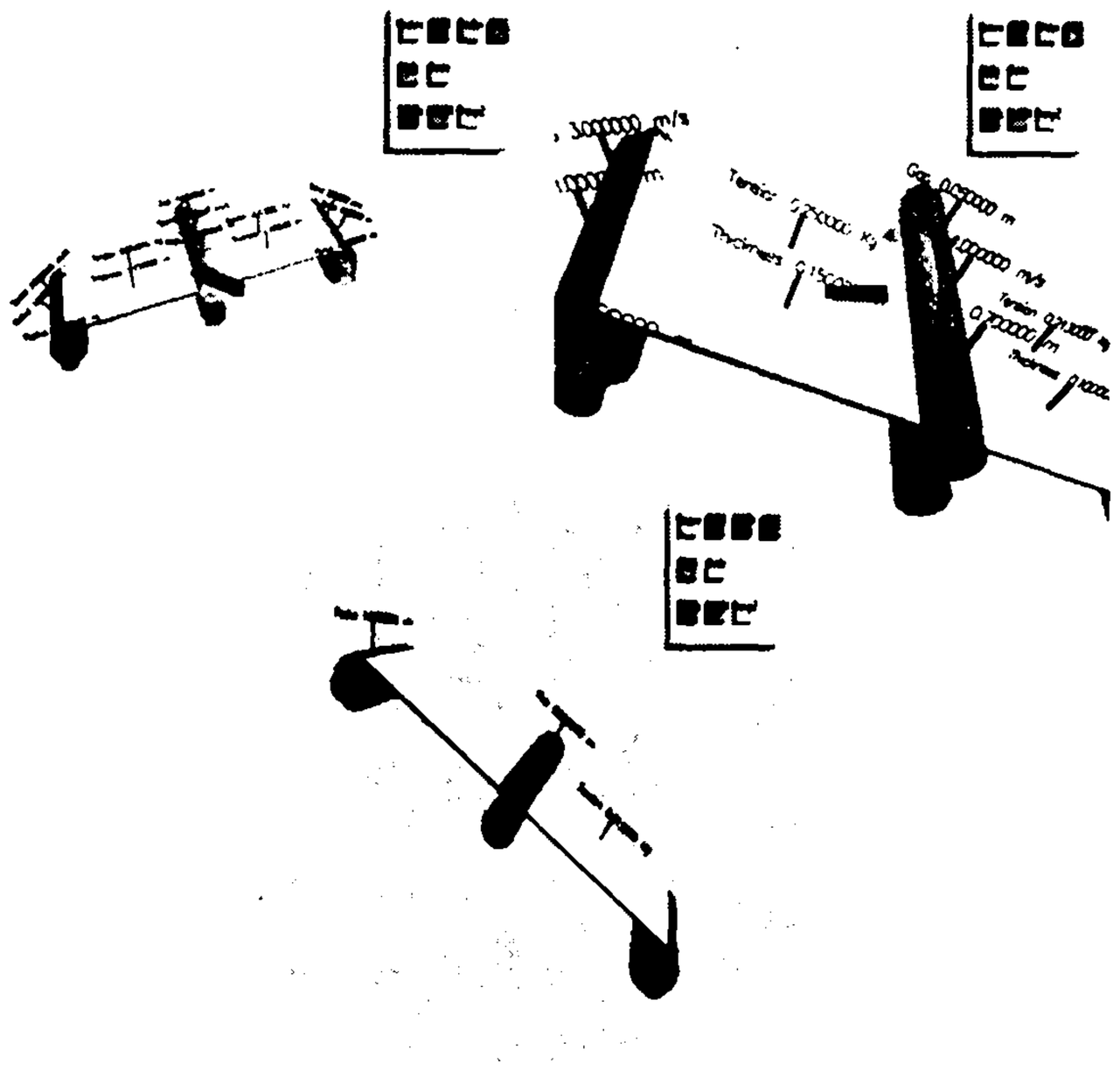


그림 20 압연기의 동적 보여주기 화면

들이 병렬적으로 분산되어 각각의 롤 및 릴을 제어하고 그 영향으로 실제 환경이 변화하며 그 환경 변화가 다시 이들에게 영향을 주는 식으로 시스템이 이루어진다.

그림 21 및 22는 AGC의 최상위 단계 및 그 하위 단계의 DFD이다. 이번 구현에서는 제어기를 두 개로 나누어 명세를 만들고 시뮬레이션하였다. 즉, 그림 21과 22에 있는 명세는 두 개의 SC와 한 개의 AGC를 명세한 것이고 나머지 한 개의 AGC는 따로 명세하였다.

그림 21에서는 제어기가 없는 것이 당연하다. 왜냐하면 제어 가능한 프로세스가 단 하나밖에 없기 때문이다. 그리고 그림 22에서는 모든 프로세스들이 동시에(Concurrent) 활성화된 것으로 보고 있기 때문에

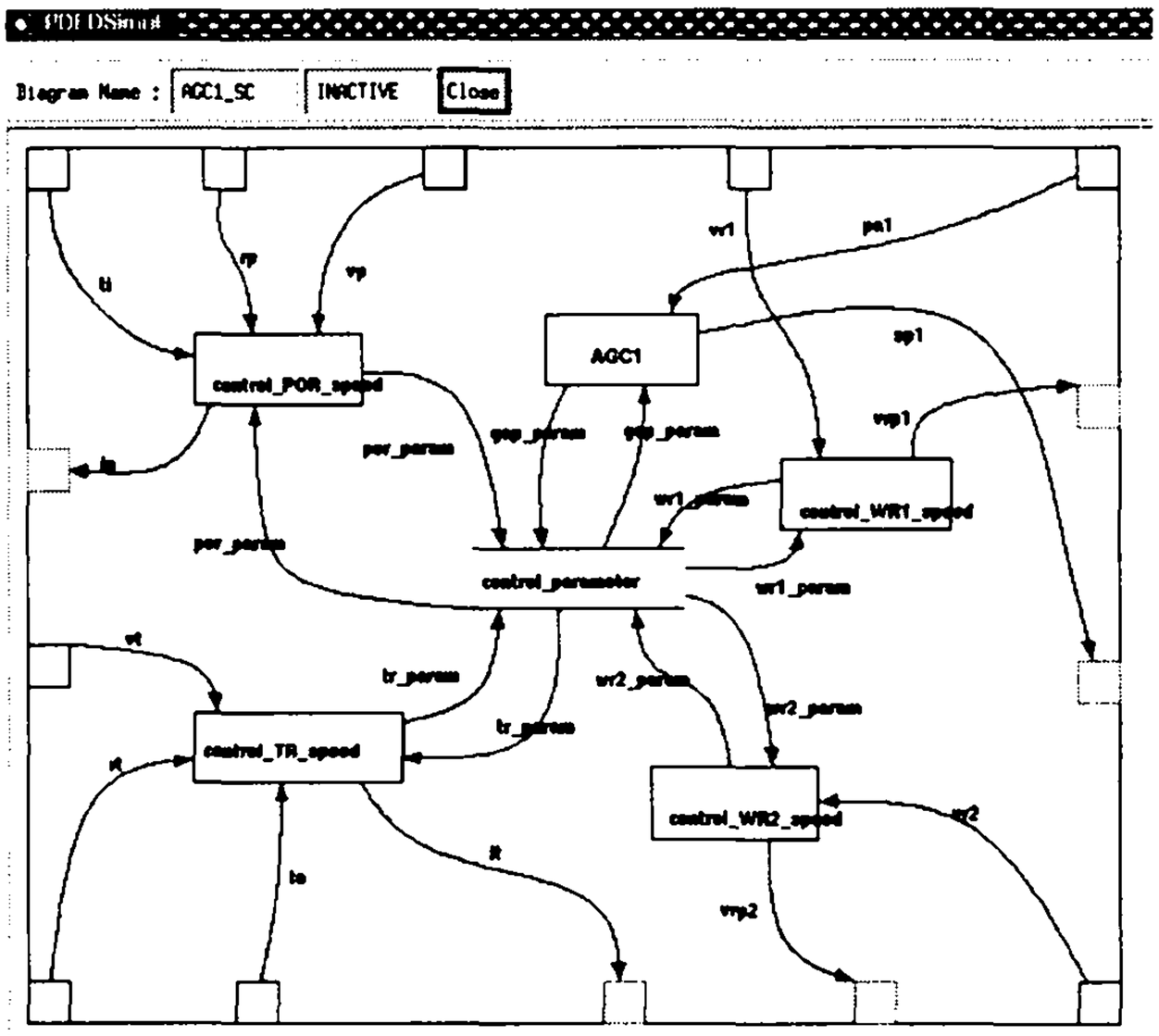


그림 22 AGC 및 SC 단계 1 DFD

제어기가 필요 없었다.

그럼 이제 그림 22에 있는 프로세스들의 프로세스 명세에 대해 살펴보겠다.

```
Process AGC1
Input
    pa1 : float ;
    gap_param : COMPOUND ;
Output
    sp1 : float ;
    gap_param : COMPOUND ;
Computational Specification
On Arrival (pa1)
ProcessTime (randExp(5ms)) {
    Store (pa1) ;
    sp1 := 100 - pa1 - 10 * 1 * 3 - 5 / 3 ;
```

```
        Generate (pa1) ;
    }
ExecutionControl
    StartStop
```

여기서 AGC1은 pa1과 gap_param을 입력으로 받아들이고 sp1과 gap_param을 출력으로 내보내며 pa1이 입력으로 들어왔을 때 평균 5ms의 지수분포를 갖는 시간 정도시간에 pa1을 데이터 저장소(Data Store)에 저장하고 sp1을 주어진 식에 의해 계산하고, pa1 출력을 내보내는 일을 하며 비활성화후 활성화될 때 처음부터 다시 한다는 내용을 담고 있다.

```
Process control_POR_speed
Input
    ti : float ;
    rp : float ;
    vp : float ;
    por_param : COMPOUND ;
Output
    ip : float ;
    por_param : COMPOUND ;
Computational Specification
    On Arrival (ti)
    ProcessTime (randExp(5ms)) {
        Store (ti) ;
        retrieve (rp) ;
        ip := -1 * rp * ti / 1 / 1 ;
        Generate (ip) ;
    }
    On Arrival (rp)
    ProcessTime (randExp(5ms)) {
        Store (rp) ;
```

```
        retrieve (ti) ;
        ip := -1 * rp * ti / 1 / 1 ;
        Generate (ip) ;
    }
    On Arrival (vp)
    ProcessTime (randExp(5ms)) {
        Store (vp) ;
        retrieve (rp) ;
        retrieve (ti) ;
        ip := -1 * rp * ti / 1 / 1 ;
        Generate (ip) ;
    }

ExecutionControl
    StartStop

Process control_POR_speed
Input
    to : float ;
    rt : float ;
    vt : float ;
    por_param : COMPOUND ;
Output
    it : float ;
    por_param : COMPOUND ;
Computational Specification
    On Arrival (to)
    ProcessTime (randExp(5ms)) {
        Store (to) ;
        retrieve (rt) ;
        it := rt * to / 1 / 1 ;
        Generate (it) ;
    }
    On Arrival (rt)
```

```
ProcessTime (randExp(5ms)) {
    Store (rt) ;
    retrieve (to) ;
    it := rt * to / 1 / 1 ;
    Generate (it) ;
}
On Arrival (vt)
    ProcessTime (randExp(5ms)) {
        Store (vt) ;
        retrieve (rt) ;
        retrieve (to) ;
        it := rt * to / 1 / 1 ;
        Generate (it) ;
    }

ExecutoonControl
    StartStop

Process control_WR1_speed
Input
    vr1 : float ;
    wr1_param : COMPOUND ;
Output
    vrp1 : float ;
    wr1_param : COMPOUND ;
Computational Specification
    On Arrival (vr1)
    ProcessTime (randExp(5ms)) {
        Store (vr1) ;
        vrp1 := vr1 ;
        Generate (vrp1) ;
    }
ExecutionControl
    StartStop
```

```
Process control_WR2_speed
Input
    vr2 : float ;
    wr2_param : COMPOUND ;
Output
    vrp2 : float ;
    wr2_param : COMPOUND ;
Computational Specification
    On Arrival (vr2)
    ProcessTime (randExp(5ms)) {
        Store (vr2) ;
        vrp2 := vr2 ;
        Generate (vrp2) ;
    }
ExecutionControl
    StartStop
```

이상으로 3차년도 프로젝트의 결과인 분산 시뮬레이션, 시뮬레이션 결과 데이터 분석기 등에 대하여 설명을 마치고 다음엔 2단계로 계속할 연구 내용에 대해 정리하겠다.

제 3절 앞으로의 개발 방향

빠른 실시간 시스템의 프로토타입(Prototype) 제작과 그 유효성의 검사(Validation)는 소프트웨어 공학의 주 연구 분야들중 하나이다. 실시간 시스템은 엄격한 시간적 요구사항들을 가지고 있고 이 요구사항들을 만족시키기 위한 Temporal Logic, 시뮬레이션, 스케쥴링, 태스크 할당(Allocation)등의 여러 접근방법들이 고안되었다. 실시간 시스템은 주로 컴퓨터로 만들어지는 제어시스템(Control System)과 센서와

엑츄에이터등으로 구성된 그것에 의해 제어되는 시스템 등으로 구성된다. 일반적으로, 실시간 시스템의 시간적 행위에 대한 유효성 검사는 제어 시스템에 의해 제어시스템이 어떻게 반응, 동작하는지에 대한 추상화된 예상과 가정에 근거한다.

하지만, 대부분의 실시간 시스템들은 그것이 제어하는 동적으로 변화하고 예측하기 어려운 환경에 실재하는 객체들과 물리적으로 복잡하게 상호작용한다. 예를 들어 항공기 제어 시스템은 여러 기상 상황등의 주변 환경에 맞춰 작동해야 하고 아주 엄격한 안전 요구를 고려할 때 간단한 환경 모델은 의미있는 행위 명세와 그것의 실제간은 시뮬레이션에는 충분하지 않을 수 있다. 따라서, 실시간 시스템의 완전한 명세는 적어도 주어진 중요한 요구사항을 검증할 수 있을 정도로 제어 시스템의 작동 환경과 그것에 의해 실재 객체들이 어떨게 반응하는지를 설명해야 한다. 게다가, 명세의 시뮬레이션은 제어 시스템뿐 아니라 주어진 그 외부 환경에 대한 시뮬레이션을 포함하거나 적어도 외부환경과 함께 동작할 수 있어야 한다.

많은 경우 "환경"은 우리가 살고 있는 물리적 환경이고, 따라서 환경의 시뮬레이션은 중력, 마찰 등의 물리 현상을 시뮬레이션할 수 있어야 한다. 물리적은 현상의 시뮬레이션은 다년간 "물리에 근거한 모델링(Physcally-based Modeling)", "질적인 설명(Qualitative Reasoning)", "계산적인 물리(Computational Physics)" 등의 이름으로 여러 분야에서 연구되어 왔다.

의미있는 물리적 시뮬레이션을 위해, 제어 시스템과 관계된 실재 객체들은 어떤 모양을 가지고 어느 정도의 공간을 차지하게 된다. 사실,

물리적 세계에서 여러 다른 모양과 설정(Configuration)에서의 시뮬레이션은 서로 다른 결과를 만들어 낸다. 예를 들어 외형상 전투기는 여객기와는 공기중에서 다른 유체 역학적(Aerodynamic) 특성을 보인다. 우리는 이러한 무게, 모양, 물질등의 물리적 성질과 위치, 방향, 속도 등의 설정을 "형태(Form)"라 부른다.

형태는 기능에도 영향을 준다. 예를 들어 크기가 다른 두 개의 로봇 운반기계는 서로 다른 작업 영역과 능력을 가진다. 따라서 형태 데이터가 없이는 시스템의 행위(Behavior)와 기능(Function)을 옳게 명세하기 어렵다. 사실, 시뮬레이션을 통한 가시화(Visualization)는 시스템의 유효성 검증의 좋은 방법인데, 형태 데이터를 이용해 시뮬레이션과 동시에 실제 환경을 보여줌으로써 비행기들이 아주 가까이 다가간다거나 두 로봇이 작업하다가 부딪히는 등 발생하지 않아야 할 상황이나 위험 상황이 한 눈에 드러날 수 있다.

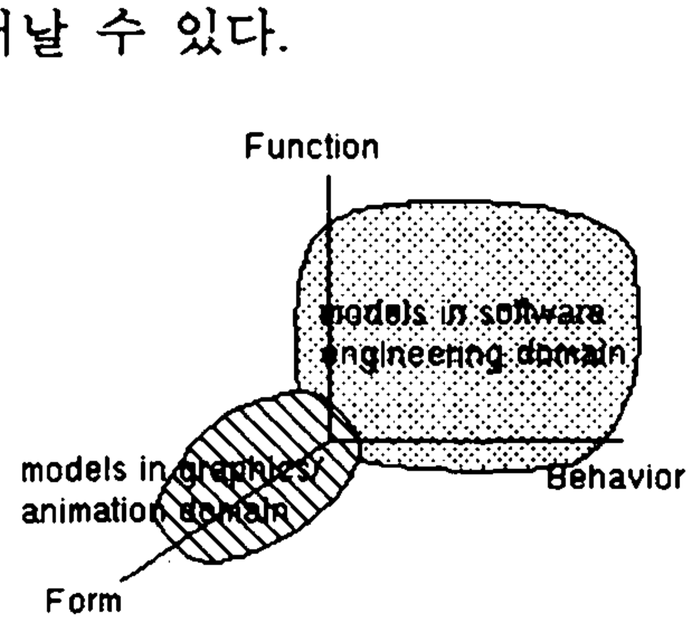


그림 23 분석의 세 영역

그림 23과 같이 전통적으로 소프트웨어 공학 분야에서는 내장형 제어 시스템(Embeded Control System)의 명세와 그 시뮬레이션에 대한 연구를 행위와 기능에 촛점을 맞춰 왔고, 그래픽스와 가상 현실 분야에서는 형태 데이터의 실제와 가까운 가시화에 중점을 두어 왔다. 이들 중 네 가지의 독특한 관련 연구들에 대해 살펴보겠다. Gaskell과

Philips의 EGS(Executable Graphical Specification)는 DeMarco의 Data Flow Diagram(DFD) 표현 방법을 사용하여 기능 모델을 명세하고 프로세스를 자체 정의한 코드로 기술하고 그것을 1대1 해석(Interprete) 방식으로 시뮬레이션함으로써 시스템의 기능을 살펴볼 수 있게 한다. 하지만, EGS는 데이터 처리 중심 소프트웨어를 목표로 하여 만들어졌고 실시간 성질들을 명세할 수 있는 요소가 없다. Harel은 그래피컬한 Computer-Aided Software Engineering(CASE) 도구인 STATEMATE를 만들었는데, 이 도구는 대형의 복잡한 실시간 시스템에 대한 명세, 분석, 설계, 다큐먼트 생성, 코드 생성등을 한다. ASADAL은 실시간 시스템을 위한 역시 그래피컬한 CASE 도구이다. 이 도구는 쉽게 단계적으로 시스템에 접근할 수 있는 방법을 제공하고 있으며 시뮬레이션 및 증명을 통해 시스템에 대한 유효성 검사, 검증 등을 한다.

이 모든 접근 방법들이 의미있는 제어 시스템 운영 환경 객체들의 행위엔 관심을 기울이고 있지 않았으나 최근 Friesen 등이 실시간 제어 시스템과 그 작동 환경까지를 혼합적(Hybrid)으로 명세하는 객체 지향적인 방법을 제시하였다. 그러나, 아직 형태에 대한 것은 아직 대부분의 방식에서 간과되고 있는 현실이다.

반면, 그래픽스와 Computer-Aided Design(CAD) 등의 분야에서는 명확하게 형태, 공간적 행위와 기능들을 표현하고, 이들을 모두 일관된 객체로 표현하는 것등의 방법으로 "움직이는" 세계를 만드는 방법이 연구되어 왔고, 다년간 여러 도구들이 개발되었다. 그러나 이 분야에서는 그러한 것을 개발하는 데 대한 체계적인 기본환경(Systematic Framework)이 마련되어 있지 않다. 예를 들어 CAD로 모양을 만들고 일반적인 프로그래밍 언어 수준의 언어를 이용한 프로그래밍을 통해 행위

를 넣고 있다. 이러한 구조화되지 않은 개발 방법은 개발물들을 유지, 보수하고 최적화(Tuning)를 어렵게 만들고 있다.

2단계 프로젝트는 첫 째 효과적인 유효성 검사 및 실제와 같은 실시간 시스템 명세의 시뮬레이션이 목표이고, 이 목표를 달성하기 위하여 기존의 ASADAL이 가지고 있는 행위 및 기능 명세 및 시뮬레이션 기능을 확장하여 형태 정보도 명세하여 행위, 기능, 형태의 명세를 가시화와 함께 시뮬레이션할 수 있게 하는 방법 및 이를 구현한 도구인 ASADAL/PROTO를 개발하였다. 이 방법은 ASADAL의 강력한 행위, 기능의 정형적인 명세 방법(DFD와 Time-Enriched Statecharts(TES)이용)과 객체 지향적인 형태 명세 언어인 *visual object specification(VOS)*을 이용하고 있다.

형태를 가지고 있는 객체들을 위해 VOS는 이들 객체의 모양, 크기, 무게 등의 물리적인 성질들과 객체들 간의 공간적인(Spatial) 제약(Constraint), 객체들의 자발적인(Spontaneous) 행위등을 기술하게 되어 있다. 실시간 시스템의 바람직한 행위는 제어 시스템의 명세에 기술되지만 최종적으로 그것이 개발된 후의 행위는 제어 시스템의 인자(Parameter)들과 그것의 형태, 작동 환경에 의해 영향받는다. 따라서, 실시간 시스템의 완전한(Complete) 명세를 위해 ASADAL/PROTO 명세 방법은 DFD, TES와 VOS를 합친 새로운 통합 실시간 객체 모델(Unified Real-Time Object Model)을 제시한다.

1. 시스템 모델링

형태로의 확장을 하지 않은 ASADAL의 주요 시스템 모델링 철학은 단계적인(Incremental) 개발 방법에 기반하며 그에 적합한 명세와 시뮬

레이션 도구를 가지고 있다. 우리는 형태 디자인에도 이러한 개념을 확장한다. 단계적인 개발은 프로그램을 단계적으로 개발하고 유효성 검사하고 배포할 수 있게 하며, Spiral, Evolutionary-prototyping, Staged-delivery 등의 많은 생명 주기(Lifecycle) 모델에 의해 지원을 받고 있다. 단계적인 개발의 장점은 많다. 예를 들면, 그것은 프로젝트를 작은 부분프로젝트들로 나눔으로써 위험성을 줄이고 완전한 시스템이 작동되기 오래 전부터 시스템의 동작 모습을 제공함으로써 눈에 보이는 진보를 증가시킨다. 단계적인 개발 접근 방법은 시스템의 구현을 시작하기 전에 그것에 대한 완전한 요구 사항을 가질 필요가 없음을 의미한다.

시장이나 연구 사회에 나타나기 시작하고 있는 객체 중심적인 그래픽스 패키지들조차도 아직 그래픽스 애니매이션 응용 프로그램의 단계적인 개발을 완전히 지원하지 못하고 있다. 그래픽스 객체들은 기하학적 모델링이나 CAD(computer-aided design)을 통해 그것들이 사용되거나 제어될 때 어떻게 될지를 미리 추측하고 기억해야 하는 사용자에 의해 만들어지고 있다. 그리고 나서 시뮬레이션 프로그램들이 어느 정도 자세한 단계까지 쓰여지고 컴파일되고 실행되고 수정된다. 때때로 그래픽스 객체를 수정하는 일도 발생한다.

물리적 성질들, 설정, 상호 작용하는 행위, 다른 형태에 관련된 속성들은 그것의 기능, 시간에 따른 행위들과 함께 단계적으로 전개해야 한다. 형태는 모양과 제어되는 시스템의 반응하는 행위들 뿐만 아니라 환경 객체들도 포함한다. 제어되는 시스템은 행위와 기능 명세가 만들어질 때 점차적으로 만들어지며 환경 객체의 명세는 대부분 미리 주어진다. 기본적인 주제는 각 차원의 명세들이 서로서로에게 영향을 준다

는 것이다. 예를 들면, 인간 운영자가 존재하는 환경 명세를 가진 제조 제어 시스템에서 안전을 보장해야 한다. 요구되는 행위에 따라 어떤 종류의 로봇은 제외된다. 하나의 로봇이 있을 때 우리는 로봇 간의 부딪힘(Collision)은 가능성이 없음을 가정하고 로봇에 대한 차원 인자들을 상세하게 명세하는 데 신경쓸 필요가 없음을 가정한다.

우리의 목표는 다른 상세화 단계에서 가시적으로 애니매이션되는 제어되는 객체와 환경 객체들을 가지고 실시간 제어 시스템의 명세들에 관련되는 분석을 하는 것이다. 그래서, 제어 시스템 명세의 시뮬레이션은 외부 환경에 대한 시뮬레이션과 함께 수행되며, 그 결과 제어되는 시스템과 상호작용 행위를 관찰하고 분석할 수 있다.

그림 24는 세 가지 모델링 차원을 다루는 실시간 시스템에 대한 ASADAL/PROTO의 새로운 모델링 방법을 보여 준다.

• 행위 명세는 사건에 대응하여 시스템의 상태 변화를 보여주며, 적절한 기능들을 활성화시키고 제어한다.

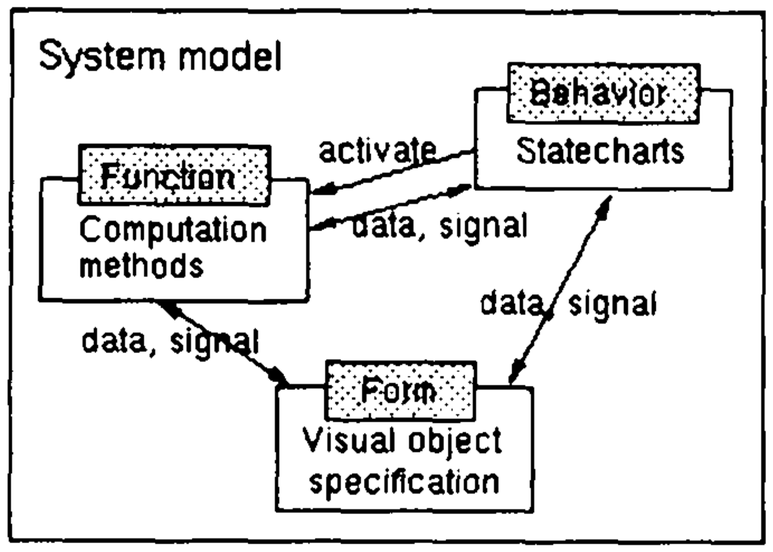


그림 24 ASADAL/PROTO의
시스템 모델

·기능 명세는 다른 entity로부터의 입력에 기반하여 제어 신호와 출력 데이타를 계산하는 함수들을 가진다.

·형태 명세는 객체들의 여러 가지 공간적 성질들을 정의한다. 행위와 기능 명세가 모든 객체에게 주어지는 반면, 형태 명세는 물리적 성질을 갖는 제어되는 시스템이나 환경 객체에게 주어진다.

계산의 분리와 추상화의 단계에 따라서 제어 논리는 행위/기능 명세 속에 있을 수도 있고 형태 명세 속에 숨겨질 수도 있다. 예를 들면, 처음에는 로봇 제어 시스템의 기능 명세가 단순히 목표 위치를 가지는 "move"명령을 보내고, 형태 명세가 Inverse Kinematics를 수행하고 joint angle 에러 행렬을 계산하고 그것의 Preset Gain에 곱하고 각 시뮬레이션 Tick마다 적절히 각 Joint를 Actuate한다. 하지만, 좀 더 상세한 단계로 가면 같은 제어 논리가 기능 명세로 들어갈 것이다. 이 경우에 형태 모델은 각 시뮬레이션 Tick마다 단순히 센서로 들어온 현재의 Joint Angle을 제공하고 Joint Actuator에 대한 신호들을 받는다. 전자의 경우는 형태가 스스로 처리하는 능력을 가지는 로봇 시스템

으로 추상화되고, 후자의 경우는 처리능력이 없는 기계적 장치로서 추상화된다.

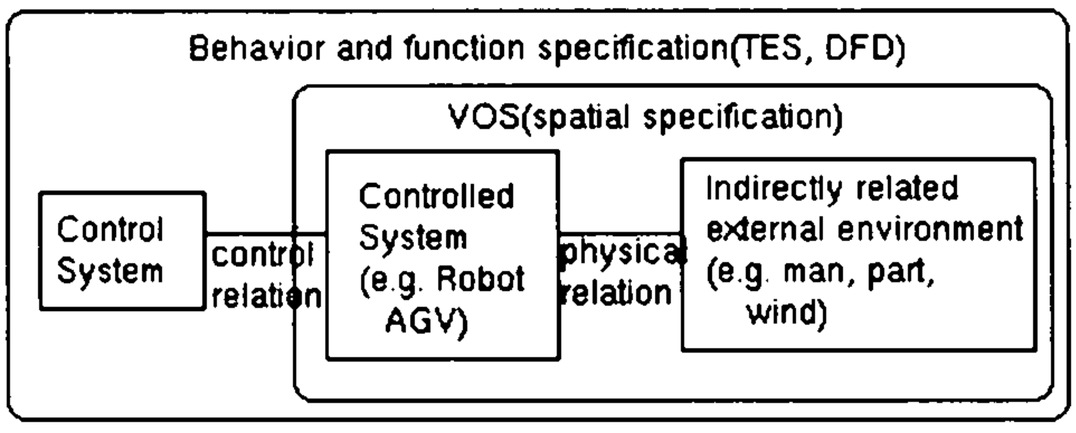


그림 25 제어 시스템 소프트웨어와 물리 환경 사이의 상호 작용의결과로서 제어되는 시스템의 행위

제어 시스템 소프트웨어와 물리 환경 사이의 상호 작용의 결과로서 제어되는 시스템의 행위행위와 기능 모델들은 각각 DFD와 TES의 정형적 모델을 사용하여 만들어지는데,이 둘은 ASADAL의 명세 방법에서 제공된다. DFD는 하나의 프로세스나 기능을 명세하는데, 어떻게 그것을 여러 개의 부분프로세스와 그들의 입력/출력관계로 분할할 수 있는지의 관점이다. DFD는 대응되는 TES를 가질 수 있는데, TES는 어떻게 프로세스가 행동하는지, 시스템의 상태에 따라서 그 프로세스를 언제 실행할지에 대해서 명세한다.

형태는 가시적 객체 명세(VOS: Visual Object Specification)을 사용하여 표현된다. 그것의 주요 목적은 물리적 개체의 물리적 성질들과 설정을 기술하는 것이다(그림 26 참조). VOS는 실시간으로 제어되는 시스템(예를 들어 비행 제어 시스템에 의해 제어되는 비행기)과 외부 환경 객체(예를 들어 새들, 구름, 지형)들의 형태를 모두 표현할 수 있다는 점에서 일반적이다. VOS는 제어되는 시스템이나 실세계 환경 객체들 사이의 명백한 물리적 관계나 제약 조건들도 명세한다.(예를

들면, Block의 위치는 일단 End-effector에 잡히면 그것의 위치를 따른다) 이상적으로 그러한 관계들은 물리적 시뮬레이션 계산에 의해 자동적으로 추론되어야 한다. 그러나, 실제로는 환경 시뮬레이션이 물리적 상세화에 의해 수행될 수 없기 때문에 추상화의 합리적인 단계가 설정되어야 하며 여전히 어느 정도의 합리적인 추측이 필요하다. 제어되는 시스템의 동적인 행위는 대개의 경우 물리적 법칙들로 그 외의 실재 객체들과 연결되어 있으므로 이는 의도되는 행위와 관찰된 행위가 혼합되는 곳이다.

2. 모델링 과정과 시뮬레이션에 의한 유효성 검사

ASADAL에서 행위와 기능 명세들은 실제로 몇 장의 MSD를 그려 봄으로써 시작한다. MSD는 시스템의 객체들 사이에 교환되는 데이타나 제어 신호들을 시간에 따라 나열함으로써 실시간 시스템의 외부 행동에 대한 전형적인 시나리오들을 기술한다.

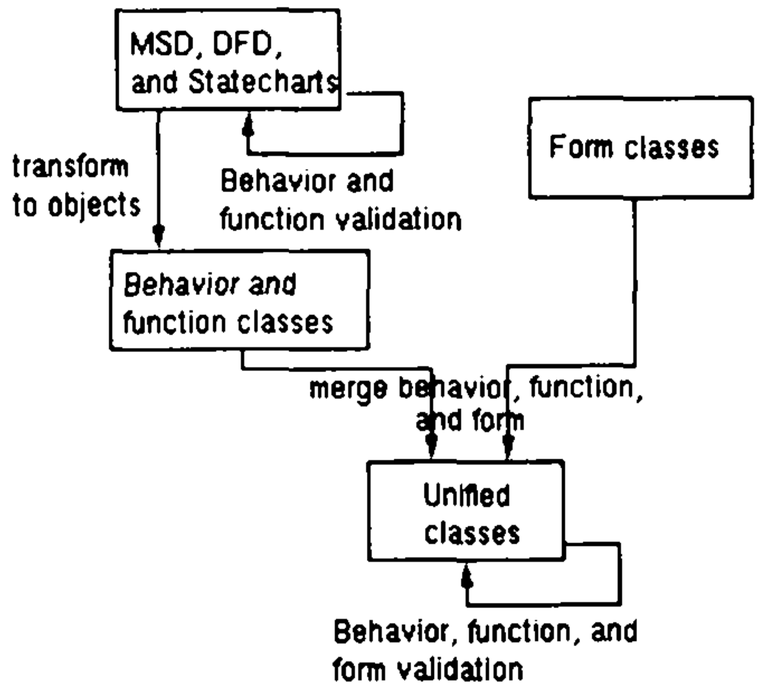


그림 26 단계적인 명세와 유효성 검사 과정

형태에 대해 고려와 함께 이것은 전체 모델링 과정의 객체 중심성을 지원한다. 다시 말하면, MSD의 사용과 형태에 대한 고려는 실세계 객

체들에 대응하는 객체들이 쉽게 identify되고 상세화될 수 있게 한다. 행위/기능 명세와 형태 디자인이 처음에는 각각 진행되다가 나중에 결합되는데 객체 중심성에 대한 일치된 견해를 가지고 전체 시스템 모델링을 시작하는 것이 중요하다.

그림~\ref{specProcess}는 단계적인 방식으로 시스템을 명세하고 유효성 검사를 하는 모델링 과정을 보여 준다. 제어 시스템과 실세계 환경을 포함한 전체 시스템의 명세는 그래피컬한 도구인 TES와 DFD를 가지고 수행한다. 실세계에 존재하는 객체 클래스들에 대한 형태의 명세는 VOS를 사용하여 병렬적으로 수행한다[1]. 좋은 객체 중심적 framework을 만들기 위해서 TES를 클래스들의 병렬적인 행위들과 그와 연결된 DFD의 기능들로 나누어서 객체 중심적 표현을 만든다. 그 후에 로봇과 같은 클래스들은 형태를 가질 것이고 제어 소프트웨어와 같은 내부 객체들은 형태를 갖지 않을 것이다. 형태를 가진 새로운 클래스들은 행위와 기능 클래스들과 그것의 VOS 클래스들로부터 상속함으로써 만들어진다. 실세계 객체들은 이러한 형태, 행위, 기능 클래스들로부터 설정 인자값(위치, 색깔)들을 새로 할당하고 객체화함으로써 만들어지고, 제어 소프트웨어와 같은 내부 객체들은 행위와 기능 클래스들로부터 형태 클래스 없이 객체화된다.

두 개의 시뮬레이션 루프(Loop)가 수행되는데, 하나는 실시간 시스템의 명세의 행위와 기능을 시뮬레이션하고, 다른 하나는 제어되는 시스템과 물리 환경을 시뮬레이션하고 가시화한다. 상호 작용하는 시스템의 행위가 ASADAL을 사용하여 관찰되고 분석될 수 있다. 이러한 명

---

1 모양 자체에 대해서는 기하학적 디자인 도구를 사용하거나 주어진 도메인에 대한 라이브러리에서 정의되어 있는 모양 모델을 선택함으로써 모델링한다

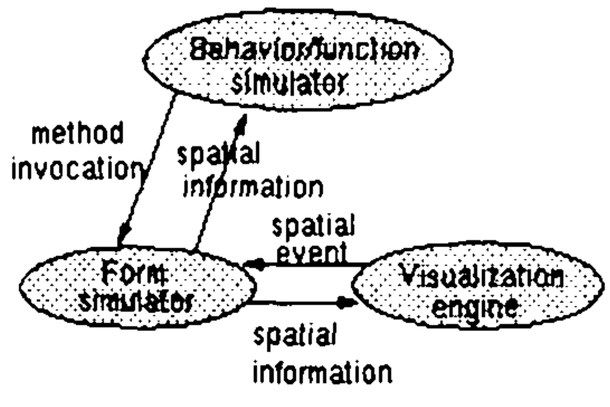


그림 27 시뮬레이션 및
가시화 시스템의 구조

세와 시뮬레이션에 의한 유효성 검사 과정은 단계적인 방식으로 계속될 수 있다. 예를 들면, 형태 객체는 그것을 구성하는 부분 객체들의 모임(Set)으로 상세화되고, 그것에 대응하여 행위와 기능들도 상세화된다.

3. 프로토타입

행위와 기능 명세와 시뮬레이션 환경에 대한 구현은 완성되었다. 전체 시스템을 ASADAL이라고 부르고 "http://selab.postech.ac.kr/~realtime"을 통하여 얻을 수 있다. ASADAL/PROTO는 새로운 형태 표현과 물리적 시뮬레이션을 위한 도구들을 포함한다. 그림 27은 행위와 기능 시뮬레이터, 형태 시뮬레이터, 가시화 엔진을 가지는 ASADAL/PROTO의 구조를 보인다. 전체 구현이 1997년 말에 유효하게 될 Java3D에 기반하고 있지만 그것의 초기 프로토타입은 VRML(Virtual Reality Markup Language)과 자바 기술에 기반하여 구현되었다. 제조 공장 예제의 가시화 시뮬레이션의 프로토타입이 그림 28에 있다. 한 Manipulator가 두 개의 다른 Pallet으로부터 부품들을 가져와서 조립하고 다른 한 Manipulator가 완성된 물건을 세번째 Pallet에 옮겨 놓는다. 사람이 시뮬레이션의 행위에 영향을 줄 수 있는데, 한 가지 예로 테이블에 부딪히면 Pickup Manipulator가 조립을 실패하게 된다.

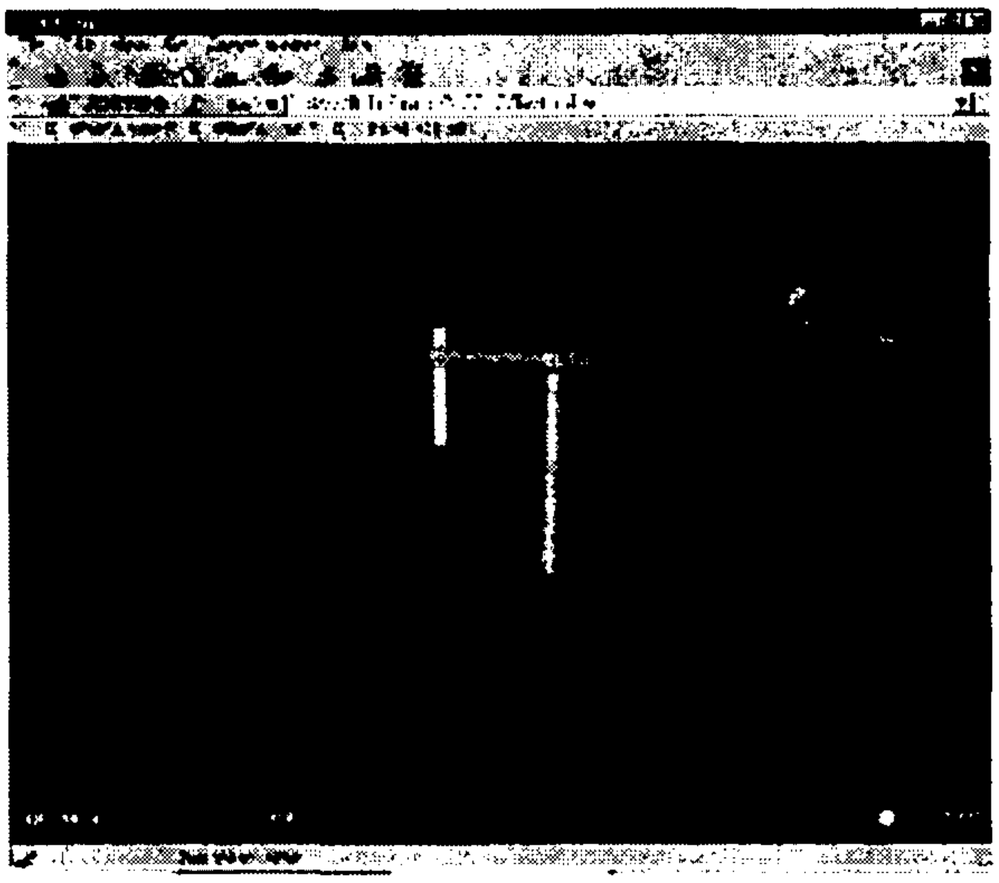


그림 28 ASADAL/PROTO의 가시화 환경
프로토타입

4. 맺음말 및 전망

명세와 그에 대한 분석과 시뮬레이션의 목적은 개발 초기 단계에서 시간과 안전이 중요한 실시간 시스템의 신용도를 증가시키기 위한 것이다. 이 목적을 달성하려면 명세 방법들은 처음에는 관계있고 중요한 환경의 행위들을 Identify할 수 있는 방법들을 제공해야 하고, 상세화 단계에서는 대응되는 기능적, 행위적 특성들을 정의하는 방법들을 제공해야 한다. "형태는 기능을 따른다"는 말이 있는데, 이것은 시스템의 기하학적, 재료적 성질들은 어떤 Utility를 가진다는 것을 보여줄 수 있음을 의미한다. 그것들은 서로 관계가 없는 사건들이 아니다. 형태의 이러한 측면은 소프트웨어 공학에서 실시간 시스템 명세의 중요한 요소로 고려되고 있지 않다. 우리는 이 Paper에서 올바른 전체 명세와 실제적인 시뮬레이션 데이타를 만들 때 형태와 환경 객체들의 모델링이 중요하다는 것을 보였다. 우리의 명세 모델은 Unified, 객체 중심적인 Framework 내에서 세 가지 차원을 모두 다루며 방법론은 제어 소프트

웨어뿐만 아니라 외부 환경 객체와 제어되는 객체들의 모델링도 포함한다. ASADAL의 단계적이고(Incremental) 계층적인(Hierarchical) 명세와 시뮬레이션 방법 및 도구들은 들은 객체 중심적인 그래픽스, 시뮬레이션 접근과 결합되고 있다. 우리는 이 일들이 실시간 그래픽스와 가상세계 생성 분야에 또한 영향을 줄 것이라고 믿는다.

제 3 차년도
연차 보고서

RTS 기술개발

RTS Technology Development

미시적 시뮬레이터 구축 기술 개발

연구기관
시스템공학연구소

과 학 기 술 처

여 백

제 2 장 미시적 시뮬레이터 구축 기술 개발

제 1 절 서론

실시간 분산 시뮬레이션 응용을 개발할 때 시뮬레이션 대상의 시간적 행동 및 분산 노드간 상호작용의 복잡성 때문에 모델 개발에 어려움이 있다. 그러나 실시간 분산 객체(RTO)를 기반으로 하여 시뮬레이션 모델을 설계할 때 모델의 시간적 행동의 표현이 자연스러워지고 설계의 명확성을 가져다 준다.

본 연구에서는 1,2 차년도에 개발한 실시간 객체지향 모델링 방법을 이용한 압연공정 제어시스템 실시간 시뮬레이터를 확장하고 그래픽 시뮬레이션 접속 기능을 개발 하였다.

시뮬레이터의 설계 구현 과정에서 RTO 접근 방법이 설계의 자연스러움, 설계 명세의 단순 명확화, 시간적 행동 표현의 복잡성 제거, 객체의 노드 분산 용이성등 많은 장점을 가지고 있음을 확인 할 수 있었고, 삼차원 그래픽을 실시간 시뮬레이터에 적용함으로써 사용자에게 시뮬레이션 결과를 효과적으로 전달할 수 있는 방법을 개발하였다. 이 기술은 보다 복잡하고 규모가 큰 실시간 분산 시뮬레이션에 효과적으로 적용할 수 있다고 본다

제 2 절 그래픽 시뮬레이션 접속 기능

1. 개요

실시간 시뮬레이션은 시뮬레이션 객체의 시간적 행위를 시뮬레이션 대상의 시간적 행위와 동일하게 모의함을 그 특징으로 한다. 따라서 시뮬레이션의 행

위를 그와 동일한 속도로 보여주는 것이 필요하며 이를 시뮬레이션의 진행과 함께 그래픽으로 표현하는 것이 효과적이다.

본 시스템은 시뮬레이션의 각종 입력변수를 그래픽 사용자 인터페이스(GUI)를 통해 입력하여 시뮬레이션 초기 파일을 구축하며, 시뮬레이션을 가동하여 시뮬레이션의 진행상황 및 출력자료를 실시간으로 제공 받고 진행 중 임의 시점에 정지하여 그 때의 데이터를 받아보는 등의 시뮬레이션 진행 제어를 할 수 있고, 3 차원 시뮬레이션 객체를 항해하여 사용자에게 필요한 정보를 쉽게 획득할 수 있게 하는 것을 특징으로 한다.

본 시스템은 세부분으로 구성되는데, 첫번째는 사용자 입력부 이다. Pay-Off Reel(POR), Work Roll(WR's)들 및 Tension Reel 로 구성된 압연공정의 각 부의 초기치 설정을 입력하는 부분, 소재의 특성치들을 입력하는 부분 및 수행시키기 위한 공정의 설정치를 입력하는 부분들로 이루어 진다. 사용자로부터 입력받은 값들은 시뮬레이터의 환경파일을 갱신하여 재구축한다.

다음은 시뮬레이션 모니터링(Monitoring)부 이다. 여기서는 압연공정 시뮬레이션 환경을 삼차원 객체로 구성하여 시뮬레이션의 수행에 따른 모의행위를 사용자에게 보여주며 시뮬레이션의 각종 변수를 삼차원 객체상의 그래프로 표현하여 사용자에 대한 시각적 효과를 증진시킨다. 또한 여기에 시뮬레이터의 시동 정지등의 사용자 제어를 구현하여 시뮬레이션의 수행을 관찰하면서 사용자에 의한 시뮬레이터 제어가 가능하게 한다.

세번째는 시뮬레이터와의 통신 부분이다. 시뮬레이션 그래픽 노드는 시뮬레이터와는 별도의 프로세서에 할당되어 있는 분산 시스템이므로 시뮬레이터 노드와 그래픽 노드간에 통신이 필요하다. 그래픽 노드는 Windows 95 상의 TCP/IP 를 이용하는데, 그래픽 노드와 연결되는 압연공정 시뮬레이터는 Dream Net 상의 분산 프로토콜을 이용하므로 양 프로토콜을 서로 변환해 주어야 한다. 따라서 양 노드간에 프로토콜 변환을 담당하는 브리지 노드를 도입하여 이를 해결한다.

2. 그래픽 노드의 설계

그래픽 노드는 사용자와 시뮬레이터 사이를 연결하여주는 역할을 한다. 그래픽 노드의 기능은 첫째 시뮬레이션 데이터를 받아서 삼차원으로 표출하고 사용자에게 현재의 시뮬레이션 결과를 알기 쉽게 보여주며, 둘째 사용자의 입력을 받아서 시뮬레이션을 제어 하는 기능이 있다. 또한 사용자는 시스템과 소재, 공정의 초기값을 설정할 수 있다.

시뮬레이터와 데이터를 주고 받는 방법으로는 우선 그래픽 노드는 TCP/IP Ethernet 네트워크에서 동작하고, 실데이터는 모두 브리지를 통해서 받는다.

가. 그래픽 노드의 구조

그래픽 노드는 다음과 같은 부분으로 이루어져 있다.

- 시뮬레이션 모니터링 부분
  현재의 시뮬레이터의 상황을 삼차원으로 사용자에게 보여 준다. 시뮬레이터의 객체에 대응하는 프로그램내의 객체를 생성하여야 하고 시뮬레이터의 상황변화에 따라 프로그램의 객체의 상태를 바꾸어 준다.

- 사용자 입력 부분
  사용자의 입력을 받아들여 처리하는 부분이다. 사용자는 시뮬레이터를 제어하는 입력을 하거나, 그래픽 노드의 표출 상태를 바꾸는 입력을 하는 것이 가능하다. 시뮬레이터를 제어하는 입력은 통신부분을 통해 처리하여야 하고, 표출 상태를 바꾸는 입력은 모니터링 부분을 통해 처리하여야 한다.

- 통신 부분

시뮬레이터에서 데이터를 받아 들이거나 시뮬레이터로 제어 메시지를 받아 들이는 부분이다. 외부와의 통신은 이 부분을 통해 이루어 진다.

나. 모니터링 부

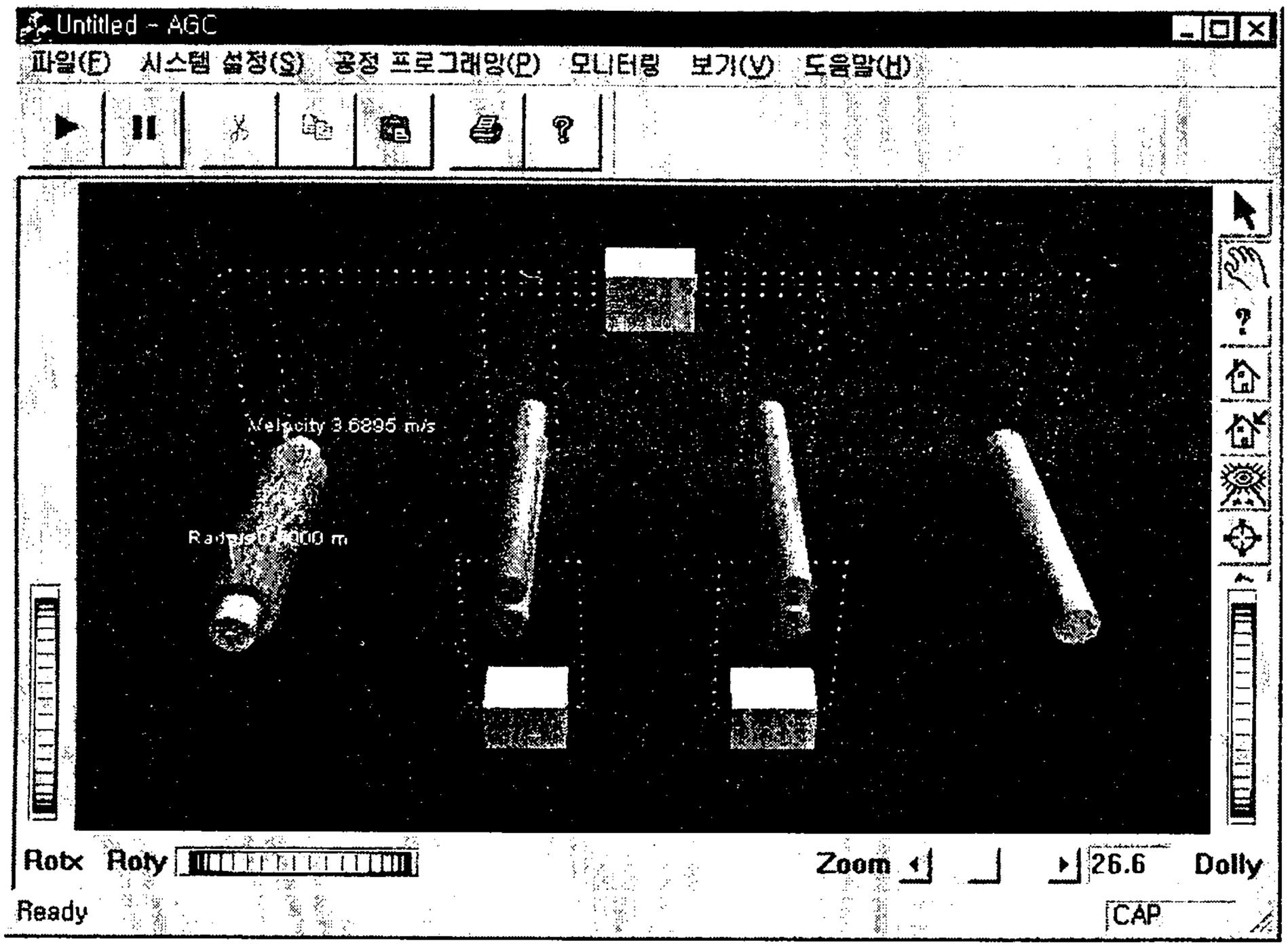


그림 1 모니터링 부 초기화면

시뮬레이터의 각각의 객체에 해당하는 프로그램의 객체들은 다음과 같다.

- Class CRollingMill

시뮬레이터 전체를 나타내는 객체로 Material, Pay-off Reel, Tension Reel, Work Roll 을 포함하는 객체이다. WarmupDelaySecs, 시스템 LifeHours, SpmInterval, 시뮬레이션 Interval, 제어 Interval, RatioOfCircumference 와 같

은 속성들도 여기에 포함된다.

- Class CMaterial

  가공되는 소재를 나타내는 객체로 소재이름, 길이, 폭, 작업롤과의 마찰계수, 평균변형저항, 영 계수, 밀도의 속성을 정의할 수 있다.

- Class CReel

  가공하는 기계의 부분 중 Reel 들을 나타내는 객체이다. POR 과 TR 의 기본적으로 같은 형태이므로 이 CReel 클래스로 POR 과 TR 변수를 만든다. CReel 클래스에는 POR 과 TR 이 가질 수 있는 설정값인 모터 토크 상수, 모터 관성 모멘트, 기어비, 맨드렐 반경, 초기 반경, WR 까지의 거리, 정규 구동 전류, 현재 반경, 현재 속도 등이 있다. 이 중에서 WR 까지의 거리의 경우 POR 은 WR1 까지의 거리를 나타내고, TR 은 WR2 까지의 거리를 나타낸다. 시뮬레이션을 위해 그림을 그려줄 때 필요한 값은 클래스에 포함되어 있지 않다.

- Class CWorkRoll

  가공하는 기계의 부분 중 Work Roll 들을 나타내는 객체이다. 현재 두 개의 Work Roll 이 있으며 작업롤 반경, 탄성 계수, 소재 입측 두께, 소재 출측 두께, 소개 입측 속도, 소재 출측 속도, 목표 단위 면적당 입측 장력, 목표 단위 면적당 출측 장력, 초기 압하력, 초기 소재와의 거리와 같은 속성들을 가지고 있다. WorkRoll 이 두개이므로 WR1 의 출측값과 WR2 의 입측값을 같게 설정했다.

위의 객체들을 표출하기 위해서는 각각의 객체들을 Open Inventor 상의 객체로 나타내야 한다. 다음과 같은 객체들을 이용하였다.

- **Steel Plate**

  철판을 실제로는 POR과 TR에 나선형으로 감겨 있는 것이지만 이것은 표현하기 힘들어서 시뮬레이션에 적합하지 않으므로 Reel에 감겨 있는 철판은 실린더로 표현하고 둘 사이의 철판은 그냥 판으로 만든다. Reel이 회전함에 따라 감겨 있는 철판이 같이 회전하면서 반경을 줄이거나 늘여서 실제로 감기고 풀리는 것과 비슷한 효과를 낸다. 철판은 Reel에 감겨 있는 것이므로 릴보다 반경이 큰 실린더로 만들면 된다. 철판이 다 풀린 경우 철판의 반경이 릴의 반경과 같으면 철판의 색이 릴에 겹쳐져 보이므로 철판의 반경을 릴의 반경보다 좀 작게 한다.

  중앙의 이동하는 철판은 무늬를 넣어서 움직임을 시각적으로 더 잘 나타내도록 했다. 또한 양 옆에 감긴 철판이 회전하는 것으로 철판이 풀리는 것 같은 효과를 줄 수 있다. 철판은 Work Roll에 들어가기 이전과 들어가기 이후의 상태가 다르므로 Work Roll에 따라서 나누어야 한다. Work Roll이 두 개 이므로 철판을 세 부분으로 나누면 된다.

- **Pay-Off Reel 과 Tension Reel**

  POR과 TR은 실린더가 회전하는 형태이다. 회전을 더 잘 보이게 하기 위하여 실린더의 끝면에 네모난 모양의 막대를 덧붙였다. POR과 TR 사이의 거리와 반경은 사용자가 설정하거나 디폴트 값을 사용하고 회전 속도는 목표 속도 값을 유지하도록 했다. Reel 하나를 그릴 때 필요한 값은 실린더의 반경과 길이, 색깔이 있다. 반경은 사용자가 설정을 하거나 디폴트 값을 사용하고 길이는 철판의 길이에서 얼만큼 더한 값을 사용하고 색깔은 녹색을 사용했다.

- **Work Roll**

  Work Roll은 철판을 얇게 펴는 역할을 하므로 실린더 두 개를 철판의 아래 위로 위치시키고 이를 회전시킨다. Work Roll의 위치는 디폴트 값을

사용하거나 사용자의 설정값을 사용한다.

- **Status Bar와 Status Text**

  사용자가 현재의 값을 알아보기 위한 방법으로 현재의 상태를 나타내주는 바를 실린더를 사용하여 그리고 현재 값을 글자로 표시했다. 이 값들은 보이지 않게 할 수도 있다.

다. 제어 메시지

그래픽 노드에서 사용자가 시뮬레이터를 제어하려면 그래픽 노드 상의 제어 도구를 사용하게 되는데 이 도구를 사용한 입력은 제어 메시지를 생성하여 브리지를 통해 시뮬레이터로 전달된다

- Start 메시지를 보내는 기능

  그래픽 노드가 시작될 때 그래픽 노드가 Start 메시지를 보내며 시뮬레이터가 Start 메시지를 받으면, 시뮬레이터는 멈춤상태가 된다.

- 멈춤 기능

  멈춤 기능은 사용자가 시뮬레이션하는 도중에 시뮬레이션을 잠시동안 멈추고 싶을 때 사용할 수 있는 기능이다. 사용자가 멈춤 기능을 사용하면, 시뮬레이터와 컨트롤러는 현재의 상태를 그대로 유지한 채 더 이상 시뮬레이션 상의 시간은 가지 않는다.

- 계속 기능

  시뮬레이터가 멈춤 상태에 있을 때 사용하는 기능으로 시뮬레이터가 계속 구동하도록 하는 기능이다.

라. 통신

그래픽 노드와 시뮬레이터 사이의 통신은 브리지 노드가 밀접하게 관련되어 있다. 실제로 그래픽 노드의 통신은 브리지 노드하고만 이루어 진다. 이를 자세히 설명하면 다음과 같다.

- 그래픽 노드와 브리지 노드는 Socket 의 TCP/IP protocol 중 UDP protocol 을 사용한다.

- UDP protocol 의 Visual C++에서의 구현

  Microsoft Visual C++을 이용한 UDP 의 구현

Microsoft Visual C++을 이용한 TCP/IP 통신을 하는 방법은 여러 가지가 있다. 이중에서 CSocket 을 이용하는 방법을 사용하였다. 통신을 하는 방법을 설명해 보면 다음과 같다.

1) Windows Socket API

Windows Socket API 는 Windows Version 의 Socket 으로 Unix 용의 Socket API 와 거의 같다. 지원되는 함수도 거의 같고 함수의 사용방법도 거의 같다.

2) MFC CasyncSocket

Microsoft Visual C++의 MFC 에 포함된 class 로, Windows Socket API 의 MFC class Version 이라고 생각하면 된다. 단지 Windows Socket API 를 MFC class 로 나타낸 정도로 비슷한 기능을 가진 Method 들을 제공한다. Windows Socket API 와 다른 점은 OnReceive 나 OnSend 같은 메시지처리

routine 을 정의 하는 것이 가능하므로 Socket 을 Polling 하지 않아도 된다는 것이다.

3) MFC Csocket

위의 CAsyncSocket 에서 상속받는 class 로 Blocking 이나 synchronous operation 을 지원하므로 사용자가 사용하기 더 쉽다는 장점이 있다. CSocket 의 객체에 Create Method 를 사용할 때 인수로 SOCK_STREAM 이나 SOCK_DGRAM 을 주어서 각각 TCP 방식이나 UDP 방식을 사용할 수 있다. TCP 방식에는 Receive 와 Send Method 를 UDP 방식에는 ReceiveFrom 과 SendTo Method 에 Blocking 기능이 있으므로 이들을 사용하면 된다.

- 실제 사용방법

  CSocket 에서 상속받는 CMySocket class 를 정의 하였다. Create Method 를 사용할 때 SOCK_DGRAM 인수를 사용하여 UDP 방식을 사용하였다. 이때 OnReceive 메시지 처리 함수를 재정의 하였다. OnReceive 처리 함수는 외부에서 보내는 데이터있으면 불리므로 이를 처리하는 부분을 넣었고, 외부에 보낼 데이터가 있을 때에는 SendTo 함수를 사용하였다.

마. 사용자 접속 부

1) 메뉴

프로그램의 메뉴 바는 다음과 같은 메뉴에 의해서 구성된다.

- 파일

  파일 메뉴는 설정 저장이 있어서 Reel 설정, Work Roll 설정, 소재 설정, 공정 설정 등을 저장할 수 있다.

● 시스템 설정

시스템 설정은 시뮬레이터에 설정되어 있는 기계에 대한 초기치를 입력하는데 사용한다. 다음과 같은 세부 메뉴가 있다.

Reel 설정

Pay-Off Reel 설정

Tension Reel 설정

Work Roll 설정

첫 번째 Work Roll 설정

두 번째 Work Roll 설정

● 공정 프로그래밍

◆ 소재 설정

시뮬레이터에 설정되어 있는 소재, 즉 사용되는 철판에 대한 정보를 입력하는데 사용된다.

◆ 공정 설정

시뮬레이터에 설정되어 있는 공정, 즉 철판을 가공하는 방법에 대한 정보를 입력하는데 사용된다.

● 모니터링

시뮬레이터의 조작에 관련된 메뉴가 있다.

◆ 시작 (Start) - ID_MONITOR_START

시뮬레이션을 시작한다. 각각의 부분이 동작을 시작한다. 이전에 일시 중지를 한 경우에는 중지한 부분부터 이어서 동작한다.

◆ 일시 중지 (Pause) - ID_MONITOR_PAUSE

시뮬레이션을 일시 중지한다. 각각의 부분이 동작을 중지하고 현재의

상태를 유지한다.

- 보기

  각 개체의 상태를 나타내는 데이터 바와 텍스트를 표시하거나 없앨 수 있다.

2) 도구바 (Tool Bar)

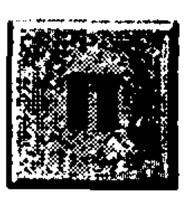
도구바는 모두 세 가지가 있다. 첫번째는 어플리케이션의 도구바로 화면 상단에 위치해 있다. 두번째는 Open Inventor 가 만드는 도구바로 화면 우측에 있다. 세번째는 슬라이드바로 확대, 축소해서 보기 기능을 지원하며 화면의 하단에 있다. 어플리케이션 도구바의 모양은 다음과 같다.

그림 2 어플리케이션 도구바

각각의 버튼을 설명하면 다음과 같다.

- 시작 버튼 (Start Button) 

  모니터링 메뉴의 시작과 같은 역할을 한다.

- 일시 중지 버튼 (Pause Button) 

  모니터링 메뉴의 일시 중지와 같은 역할을 한다.

- Open Inventor 도구바의 모양은 다음과 같다.

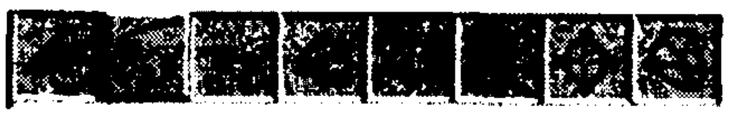


그림 3 오픈인벤터 도구바

몇 개의 버튼을 설명하면 다음과 같다.

- 선택 버튼 (Selection Button) 

  물체를 선택하고자 할 때 사용한다.

- 네비게이션 버튼 (Navigation Button) 

  물체를 여러 각도에서 보고자 할 때 사용한다.

- 슬라이드바의 모양은 다음과 같다.

그림 4 슬라이드바

오른쪽으로 움직이면 확대되고, 왼쪽으로 움직이면 축소된다.

3. 그래픽 노드의 구현

시뮬레이션 데이터를 삼차원으로 보여주기 위해서 OpenGL 을 기반으로 한 Open Inventor 2.2.1 을 사용하여 Microsoft Windows 95 상에서 Microsoft Visual C++ 4.2 를 사용하여 구현하였다. Open Inventor 는 삼차원 물체를 쉽게 생성할 수 있도록 해주며, Microsoft Window 상에서 쉽게 삼차원 Display 프로그램을 생성할 수 있도록 해준다.

- 철판 움직임 만들기

  Work Roll 이 두 개가 있으므로 철판은 3 부분으로 나눈다. 철판의 움직임

을 시각적으로 나타내기 위해 첫째 텍스처 매핑 방법을 사용하여 철판에 무늬를 넣었고 둘째 3 부분으로 나뉜 철판 각각을 다시 두 조각으로 나누어서 다음과 같이 움직인다.

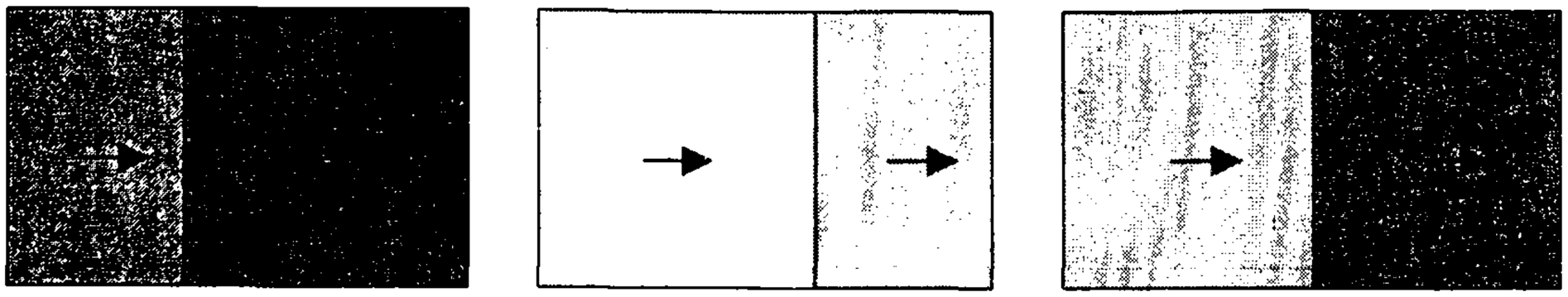


그림 5 철판 움직임

1-1 판은 증가하고 1-2 판은 감소한다. 1-1 판의 길이가 최대가 되면 1-2 판의 길이는 0 이 되고 1-1 판의 길이는 줄어들기 시작하고 1-2 판은 다시 증가를 시작한다.

위와 같은 작업을 12 판과 2 판에 대해서도 똑같이 적용하고, 시뮬레이션이 끝날 때까지 계속한다. 처음 시작해서 아무 것도 없을 경우에는 -1 판이 증가해서 최대 크기가 될 때까지 -2 판의 길이는 0 이다.

여기서 증가치와 감소치는 시뮬레이터에서 나오는 데이터인 Entry Point 값을 사용한다. Entry Point 는 세 개가 있는데 각각의 판에 따라서 현재 판의 위치를 나타내는 값이다. 1 번 판이 풀리기 시작하면 Entry Point 는 시뮬레이션이 끝날 때까지 계속 증가한다. 두 번째 Entry Point 는 두 번째 판이 풀리기 시작하면 증가하기 시작하고 세 번째 Entry Point 는 세 번째 판이 풀리기 시작하면 증가한다. 이 값들을 가지고 임의의 순간에 각 철판의 크기를 함수로 계산할 수 있다.

계산된 함수들은 다음의 그래프와 같이 나타난다.

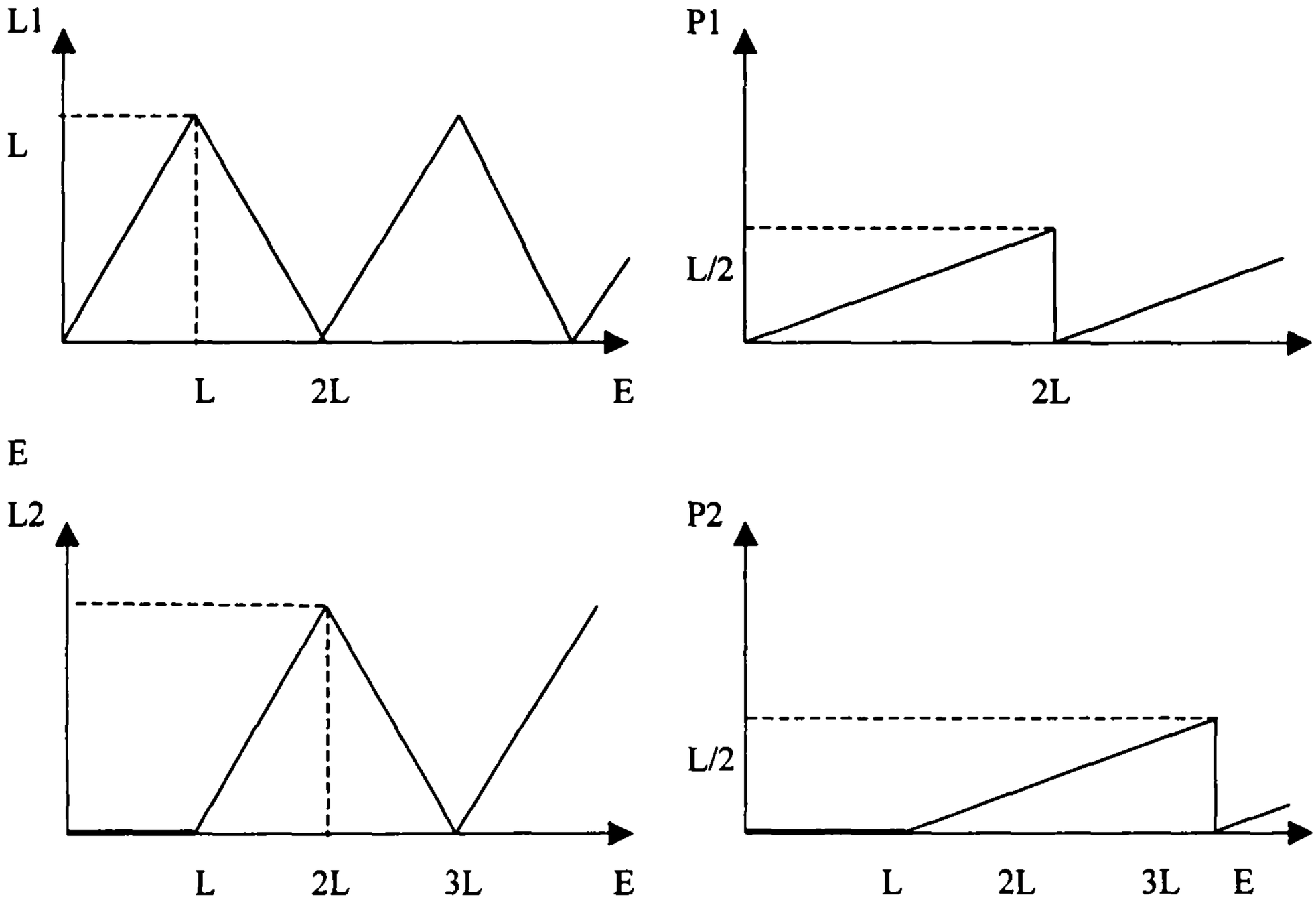


그림 6 철판 움직임의 함수 그래프

위의 그래프는 -1 번 판과 -2 번 판의 움직임을 그래프로 나타낸 것이다. 이 그래프는 1, 12, 2 번 철판 모두에 공통으로 적용할 수 있다. 엔트리 포인트 값으로 판의 위치와 길이를 결정할 수 있다.

- 데이터 Format 과 Conversion

그래픽 노드는 브리지 노드와 통신을 하는데, 통신을 하기 위해서는 데이터 Format 과 Endian 이 결정되어야 한다. 여기서는 이러한 데이터 Format 과 Endian 을 설명한다.

- 데이터 Format

  브리지 노드에서 오는 데이터는 두 가지 종류가 있다. 컨트롤러에서 오는 제어 메시지와, 시뮬레이터에서 오는 시뮬레이션 Status 메시지가 있다.

  각각의 메시지의 종류는 다음과 같다.

표 1 Sun 쪽의 컨트롤러에서 Dream Net 쪽의 시뮬레이터로 전달되는 메시지들

DFC ID	Name	설명
1	POR_SvM_from_SC	Speed 컨트롤러에서 POR 로 가는 제어 메시지
2	WR1_SvM_from_SC	Speed 컨트롤러에서 WR1 으로 가는 제어 메시지
3	WR2_SvM_from_SC	Speed 컨트롤러에서 WR2 로 가는 제어 메시지
4	TR_SvM_from_SC	Speed 컨트롤러에서 TR 로 가는 제어 메시지
5	WR1_SvM_from_AGC1	AGC 에서 WR1 으로 가는 제어 메시지
6	WR2_SvM_from_AGC2	AGC 에서 WR2 로 가는 제어 메시지

표 2. DreamNet 쪽의 시뮬레이터에서 Sun 쪽의 컨트롤러로 전달되는 메시지들

메시지 ID	Name	설명
1	SC_SvM_from_POR	POR 에서 SC 로 가는 시뮬레이션 상태 정보
2	SC_SvM_from_WR1	WR1 에서 SC 로 가는 시뮬레이션 상태 정보
3	SC_SvM_from_WR2	WR2 에서 SC 로 가는 시뮬레이션 상태 정보
4	SC_SvM_from_TR	TR 에서 SC 로 가는 시뮬레이션 상태 정보
5	AGC1_SvM_from_WR1	WR1 에서 AGC1 로 가는 시뮬레이션 상태 정보
6	AGC2_SvM_from_WR2	WR2 에서 AGC2 로 가는 시뮬레이션 상태 정보
7	WR1_SvM_from_POR	POR 에서 WR1 으로 가는 시뮬레이션 상태 정보
8	WR2_SvM_from_WR1	WR1 에서 WR2 로 가는 시뮬레이션 상태 정보
9	TR_SvM_from_WR2	WR2 에서 TR 로 가는 시뮬레이션 상태 정보

두 가지 종류의 데이터 Format 은 다음과 같다.

표 3 제어 메시지

Contents	Length	Notes
DFCID	4 bytes	보내는메시지의 DFCID
Size	2 bytes	실제 데이터의 Size
데이터 Format	2 bytes	데이터 의 유효성 여부를 나타내는 Flag
데이터	4 bytes	Float 형의 실제 제어 데이터

표 4 시뮬레이션 Status 메시지

Contents	Length	Notes
Size	2 bytes	실제 데이터의 Size
메시지 ID	2 bytes	보내는메시지의 메시지 ID
데이터 1	4 bytes	Float 형의 실제 시뮬레이션 Status 데이터 중 첫 번째
데이터 2	4 bytes	Float 형의 실제 시뮬레이션 Status 데이터 중 두 번째
데이터 3	4 bytes	Float 형의 실제 시뮬레이션 Status 데이터 중 세 번째
데이터 4	4 bytes	Float 형의 실제 시뮬레이션 Status 데이터 중 네 번째

각각의 메시지에 따라 전달되는 데이터들의 종류와 의미는 다음과 같다.

표 5 Sun 쪽의 컨트롤러에서 Dream Net 쪽의 시뮬레이터로 전달되는 메시지들

DFC ID	Name	변수명	설명
1	POR_SvM_from_SC	Ip	Ip 는 Speed 컨트롤러에서 POR 로 가는 전류의 양을 나타낸다
2	WR1_SvM_from_SC	Vrp1	Vrp1 는 Speed 컨트롤러에서 WR1 으로 가는 속도조정을 나타낸다.
3	WR2_SvM_from_SC	Vrp2	Vrp1 는 Speed 컨트롤러에서 WR2 으로 가는 속도조정을 나타낸다.
4	TR_SvM_from_SC	It	It 는 Speed 컨트롤러에서 TR 로 가는 전류의 양을 나타낸다
5	WR1_SvM_from_AGC1	Sp1	Sp1 은 AGC1 에서 WR1 으로 가는 Speed Parameter 의 값이다.
6	WR2_SvM_from_AGC2	Sp2	Sp1 은 AGC2 에서 WR2 으로 가는 Speed Parameter 의 값이다.

표 6 DreamNet 쪽의 시뮬레이터에서 Sun 쪽의 컨트롤러로 전달되는 메시지들

메시지 ID	Name	변수명	설명
1	SC_SvM_from_POR	Rp, Ti, Vp	Rp, Ti, Vp 는 각각 POR 의 Radius, Tension, Velocity 를 나타내는 시뮬레이션 상태 정보
2	SC_SvM_from_WR1	Vr1	Vr1 은 WR1 의 속도를 나타내는 시뮬레이션 상태 정보
3	SC_SvM_from_WR2	Vr2	Vr1 은 WR1 의 속도를 나타내는 시뮬레이션 상태 정보
4	SC_SvM_from_TR	Rt, To, Vt	Rt, To, Vt 는 각각 POR 의 Radius, Tension, Velocity 를 나타내는 시뮬레이션 상태 정보
5	AGC1_SvM_from_WR 1	Pa1	Pa1 은 WR1 의 Roll Force 를 나타내는시뮬레이

			션 상태 정보
6	AGC2_SvM_from_WR2	Pa2	Pa2 은 WR1 의 Roll Force 를 나타내는시뮬레이션 상태 정보
7	WR1_SvM_from_POR	Hi, Pe	Hi, Pe 는 각각 POR 과 WR1 사이의 철판의 Thickness 와 Entry_point 를 나타내는 시뮬레이션 상태 정보
8	WR2_SvM_from_WR1	V12, H12, T12, W1e	V12, H12, T12, W1e 는 각각 WR1 과 WR2 사이의 철판의 Velocity, Thickness, Tension, Entry_point 를 나타내는 시뮬레이션 상태 정보
9	TR_SvM_from_WR2	Vo, Ho, W2e	Vo, Ho, W2e 는 각각 WR2 과 TR 사이의 철판의 Velocity,

			Thickness, Entry_point 를 나타내는 시뮬레이션 상태 정보

- Conversion

  시뮬레이션 시스템에 Intel PC 와 Sun Sparc 이 공존하므로 Endian 문제가 발생한다. 여기서 Endian 이란 여러 바이트로 된 데이터에 대해서 상위 바이트와 하위 바이트를 어떤 순서로 저장하는가를 뜻하는 말이다. Intel x86 PC 와 Sun Sparc 은 서로 Endian 의 순서가 반대이다. 따라서 서로간에 데이터를 주고 받을때는 데이터를 바꾸어 주어야 한다.

  현재 브리지 노드에서 TCP/IP 쪽으로 데이터를 보낼 때는 모두 Unix Endian 으로 바꾸도록 브리지가 프로그램 되어 있다. 그리고 DreamNet 으로 가는 데이터는 모두 PC Endian 으로 바꾸도록 되어 있다. 이러한 이유로 그래픽 노드로 오는 모든 데이터들은 Unix Endian 으로 되어 있다. 따라서 데이터를 얻어내기 전에 바이트의 순서를 바꾸어 주어야 한다.

4. 브리지 노드의 구현

가. 개요

Dream Kernel 은 Dream Net 이라는 Data-Field Ethernet 네트워크위에 구현되어 있다. Dream Net 은 Ethernet 위의 Packet 드라이버로 구동되므로 TCP/IP 같은 네트워크와의 연계하기 위해서 프로토콜을 변경하여 주는 브리지 프로그램이 필요하다.

브리지 프로그램은 Dream Net 과 TCP/IP 를 서로 연결 시켜주는 역할로, 구

체적으로 설명하면 한쪽의 정보를 다른 쪽의 네트워크에 맞는 형태로 바꾸어 전달해 주는 역할을 한다. 현재 구현되어 있는 것은 두 가지 형태의 시뮬레이터 구조에 대해서 브리지를 구현하였는데 이를 그림으로 나타내면 다음과 같다.

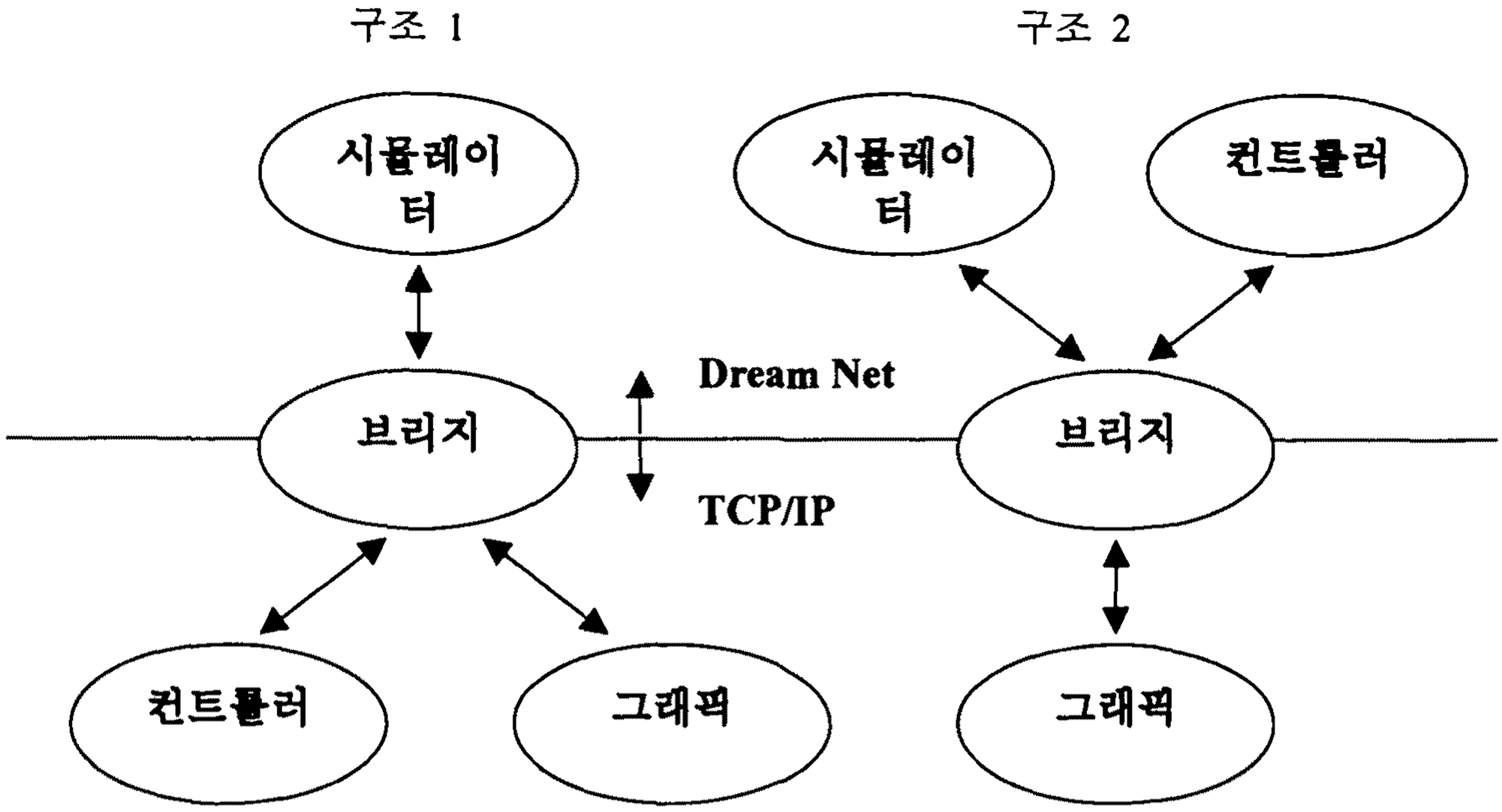


그림 7 두가지의 시뮬레이터 구조

첫 번째 구조는 브리지를 중심으로 Dream Net 쪽에 시뮬레이터가 있고, TCP/IP 쪽에 컨트롤러와 그래픽 노드 부분이 있는 구조이고, 두 번째 구조는 Dream Net 쪽에 시뮬레이터와 컨트롤러의 부분이 있고 그래픽 노드 부분만 TCP/IP 쪽에 있는 구조이다.

나. 배경 지식

브리지 Program 을 이해하는데 필요한 배경 지식은 여러가지가 있는데 크게 나누어서 TCP/IP 에 대한 이해, Unix 와 PC 의 Endian 에 대한 이해, 시뮬레이션 시스템에 대한 이해가 필요하다.

- TCP/IP 에 대한 이해

우선 Socket Interface 를 설명하면 Socket Interface 는 1981 년도에 Berkely 4.2 Software Distribution 에 소개된 Interprocess Communication 으로 여러 가지 protocol 에 대한 지원을 하는데, 이중 TCP/IP protocol 은 Internet domain(AF_INET)을 통해서 지원한다.

Internet Domain 에는 TCP 방식과 UDP 방식의 두 가지 통신방법이 존재하는데 이 중 UDP 방식을 사용한다. 각각을 설명하면, TCP 방식은 서로간에 가상의 통신선로를 만든 후에 이를 이용하여 데이터를 주고 받는 방식이고, UDP 방식은 데이터에 받는 곳과 보낸 곳을 명시하여 보내는 방식이다.

현재 그래픽 노드의 통신을 위해서 UDP 를 사용한 이유는 Dream Net 이 Packet 네트워크상에서 구현되어 UDP 와의 유사성이 많으므로 브리지가 UDP 로 구현되었고, 따라서 그래픽 노드도 UDP 로 구현하는 것이 자연스럽기 때문이다. 또한 실제의 네트워크가 가변적이므로 UDP 로 구현하는 것이 네트워크의 구성이 변할 경우 수정이 용이 하기 때문이다. 그러나 단점으로는 UDP 는 데이터의 전달이 보장되지 않기 때문에 시뮬레이션 데이터를 일부분 잃어버릴 수 있다. 그러나 현재의 그래픽 노드에서는 문제가 되지 않는다.

- Unix 와 PC 의 Endian 에 대한 이해

시뮬레이션 시스템에 Intel PC 와 Sun Sparc 이 공존하므로 Endian 문제가 발생한다. 여기서 Endian 이란 여러 바이트로 된 데이터에 대해서 상위 바이트와 하위 바이트를 어떤 순서로 저장하는가를 뜻하는 말이다. Intel x86 PC 는 Little

Endian 을 사용하고 Sun Sparc 은 Big Endian 을 사용한다.. 따라서 서로간에 데이터를 주고 받을때는 데이터를 바꾸어 주어야 한다.

현재 브리지 노드에서 Dream Net 에서 TCP/IP 쪽으로 데이터를 보낼 때는 모두 Endian 을 바꾸도록 브리지가 프로그램 되어 있다. 그리고 TCP/IP 쪽에서 Dream Net 으로 가는 데이터는 모두 Endian 을 바꾸도록 되어 있다. 이러한 이유로 그래픽 노드로 오는 모든 데이터들은 Unix Endian 으로 되어 있다. 그러나 그래픽 노드는 PC 에서 구현되어 있으므로 따라서 데이터를 얻어내기 전에 바이트의 순서를 바꾸어 주어야 한다.

주의해야 할 점은 Endian 의 문제는 네트워크 Protocol 과 상관 있는 것이 아니라 데이터를 보낸 쪽과 사용하는 쪽의 기계의 종류에 상관 있다는 것이다.

● 시뮬레이션 시스템에 대한 이해

시뮬레이션 시스템은 여러 개의 노드로 구성되어 있는데 각각의 노드는 별개로 실행되므로 처음 시스템이 기동하기 위해서는 시작을 동기화 하기 위한 메시지가 필요하다. 현재는 그래픽 노드가 가장 늦게 실행되면서 Start 메시지를 보내도록 되어 있다. 따라서 브리지에서는 이에 대한 처리가 필요하다.

시스템이 시작되고 나서는 Dream Net 에서 실행되는 노드는 자신이 받아서 처리하는 SvM 메시지에 대한 실제 DFC_ID(동적으로 할당된다)를 Broadcast 하게 된다. 따라서 브리지는 이 메시지들을 받아 어떤 SvM 메시지가 어떤 DFC_ID 를 가지게 되었는지 기록해 두어야 한다.

초기화가 끝난 후에는 데이터의 교환이 필요한데, 노드간에는 아까 주고 받았던 DFC_ID 를 이용하여 메시지를 주고 받게 된다. 브리지는 처음에 알아뒀던 DFC_ID 를 이용해서 적절한 변환을 해주어야 한다. 메시지의 전달의 구현은

뒤에 설명되어 있다.

다. 기능과 구조

시뮬레이션 시스템의 구조에 따라 브리지의 구현이 달라지는데 다음과 같다.

● 구조 1

이 경우 브리지의 기본적인 역할은 시뮬레이터와 컨트롤러사이의 데이터를 전달해 주어 시뮬레이션 시스템이 정상적으로 작동하도록 하고, 또한 전달되는 데이터를 그래픽 노드에도 전달해 주어 그래픽 노드가 현재의 시뮬레이션 상태를 표시하도록 하는 것이다. 이를 위해서는 시뮬레이터와 컨트롤러사이의 데이터를 복사하여 그래픽으로 전달하는 작업을 해야 한다. 또한 그래픽의 제어 메시지를 시뮬레이터와 컨트롤러에 각각 복사하여 전달해야 한다.

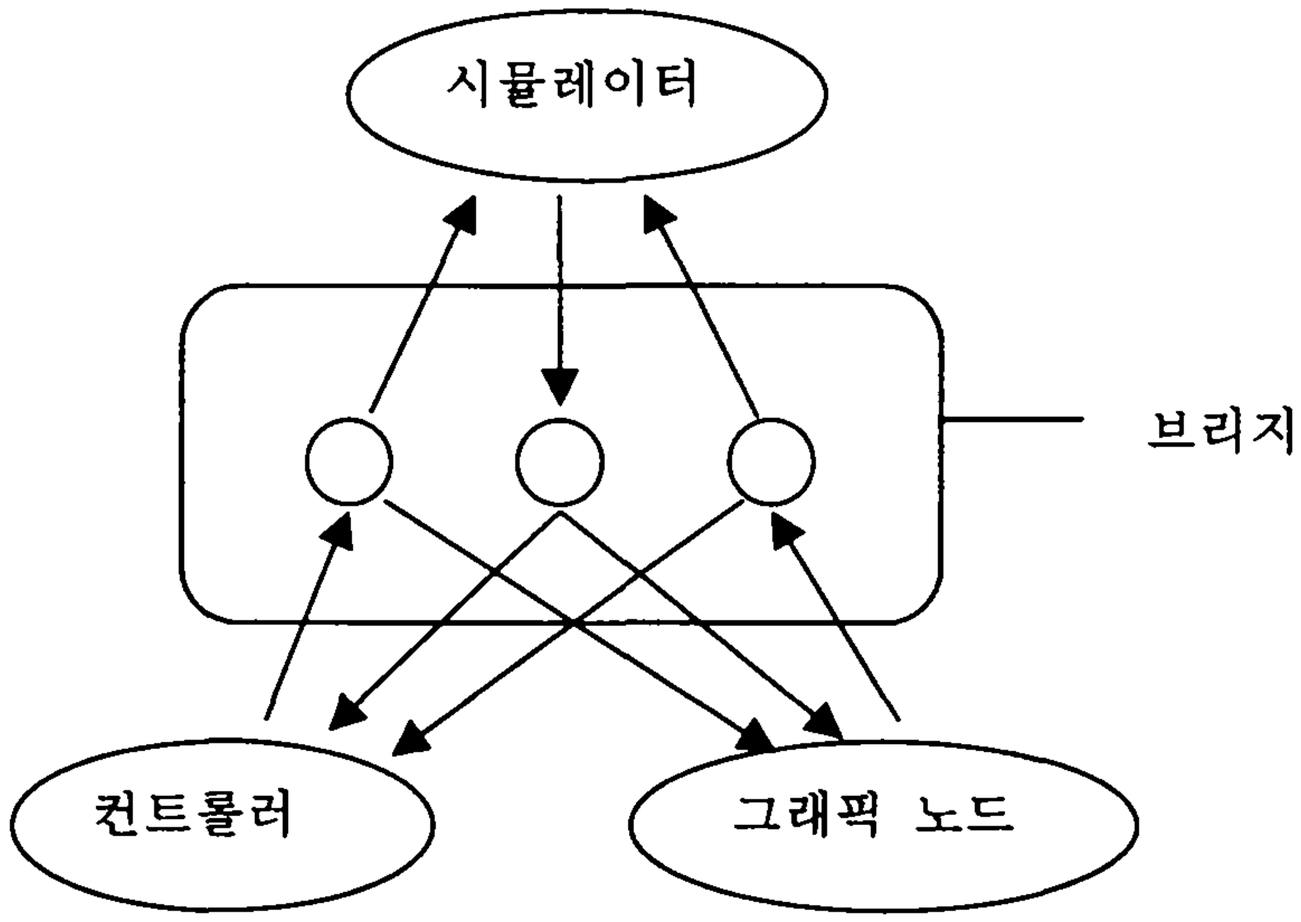


그림 8 구조 1의 시뮬레이터 구조

● 구조 2

이 경우는 시뮬레이터와 컨트롤러와의 통신은 Dream Net 을 통해서 이루어 지므로 브리지는 관계하지 않아도 된다. 대신 Dream Net 을 통하는 데이터를 보고 필요한 시뮬레이션 데이터만을 그래픽 노드로 보내주면 된다. 그리고 그래픽의 제어 메시지를 Dream Net 에 Broadcast 하면 된다.

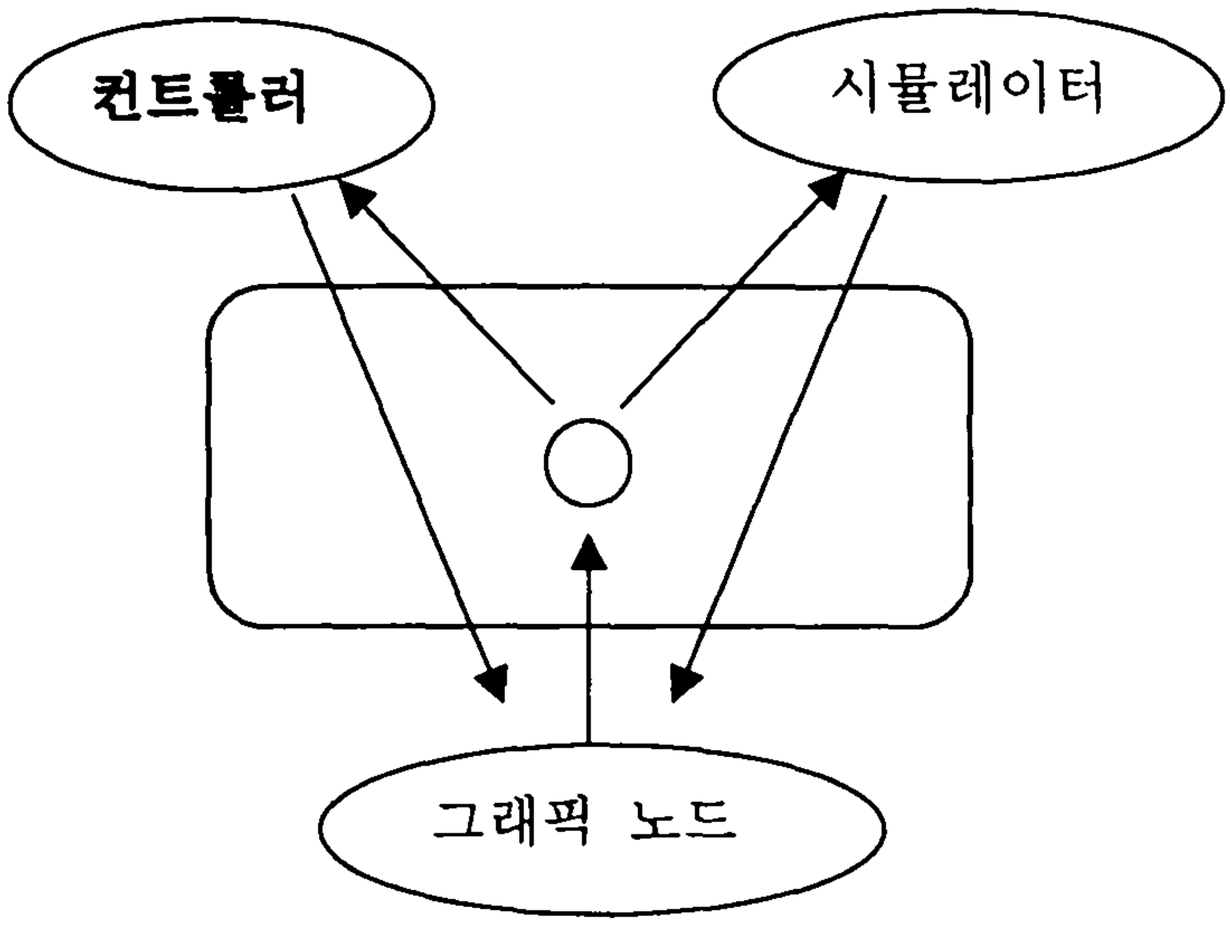


그림 9 구조 2 의 시뮬레이터 구조

라. 구현 방법

● 통신

TCP/IP 의 구현은 Socket 을 생성하는 부분과 메시지를 주고 받는 부분으로 나눌 수 있다. 각각을 설명하면

Socket 을 생성하는 방법

```
int sock_from_sun, sock_to_sun, length;
struct sockaddr_in server;

/* socket 의 생성 */
```

```
sock_from_sun = socket(AF_INET, SOCK_DGRAM, 0);
server.sin_family = AF_INET;
/* 받는 socket */
server.sin_addr.s_addr = INADDR_ANY;
/* port 를 지정. */
server.sin_port = htons(1005);
/* Socket 을 실제적인 기계의 port 에 지정 */
bind(sock_from_sun, (struct sockaddr *)&server, sizeof server);
```

메시지를 보내는 방법

```
struct sockaddr_in server;
struct hostent *hp;

/* 받는 곳의 host 이름을 이용 실제 IP address 를 얻는다 */
hp = gethostbyname(hostname);
/* 받는 곳의 주소와 port 를 지정 */
memcpy((char *)&server.sin_addr, (char *)hp->h_addr, hp->h_length);
server.sin_family = AF_INET;
server.sin_port = htons(host[index].port);
/* 실제 데이터를 보낸다 */
sendto(sock_to_sun, (unsigned char *)&buf_w,
sizeof(buf_w), 0, (struct sockaddr *)&server,
sizeof server);
```

메시지를 받는 방법

```
/* 실제의 메시지를 받는다. */
read(sock_from_sun, (unsigned char *) &sun_브리지_buf,
        sizeof(sun_브리지_buf))
```

● 메시지 변환

◆ 구조 1

Start 메시지는 DFC ID = 124 이다.

시뮬레이터로 부터 시뮬레이터가 받는 SvM DFC_ID 를 알아낸다음 컨트롤러가 보내는 메시지를 해당하는 실제 DFC_ID 로 바꾸어 시뮬레이터로 보낸다.

시뮬레이터가 TCP/IP 쪽의 컨트롤러에 보내는 메시지는 DFC ID = 40 이고 실제의 메시지 ID 는 데이터안에 들어 있다.

◆ 구조 2

Start 메시지는 DFC ID = 124 이다.

시뮬레이터와 컨트롤러 각각이 받는 SvM DFC_ID 를 알아낸 다음 Dream Net 에 전달되는 메시지를 살펴서 그래픽으로 보내야 하는 메시지이면 Format 을 바꾸어 준 후에 그래픽으로 보내준다.

◆ Endian Conversion

PC 의 Intel x86 CPU 는 little endian 이고 Sun 이 SPARC CPU 는 big endian 이므로 서로간에 Byte 순서를 한번씩 바꾸어 주면 된다. 이에 대한 함수는 void InterchangeByteFormats(unsigned char *데이터, int flag)이고 이중 flag 의 값에 따라 0 이면 2byte 1 이면 4byte 3 이면 8byte 를 순서를 바꾸어 준다. 이 함수를 이용하여 Dream Net 쪽에서 TCP/IP 쪽으로 가는 메시지와 TCP/IP 에서 Dream Net 으로 보내는 메시지들은 한번씩 바꾸어 주면 된다. 주의해야 할 사실은 브리지 가 Intel x86 에서 구현되어 있기 때문에 TCP/IP 쪽의 메시지의 한 field 를 사용하기 위해서는 endian 을 먼저 바꾸어 주어야 한다는 것이다.

마. 사용법

사용법은 간단히 그래픽 노드의 Internet 이름이나 IP 주소를 인수로 주고 프로그램을 실행시키면 된다. 실행이 끝났을 경우는 Esc 를 누르면 된다.

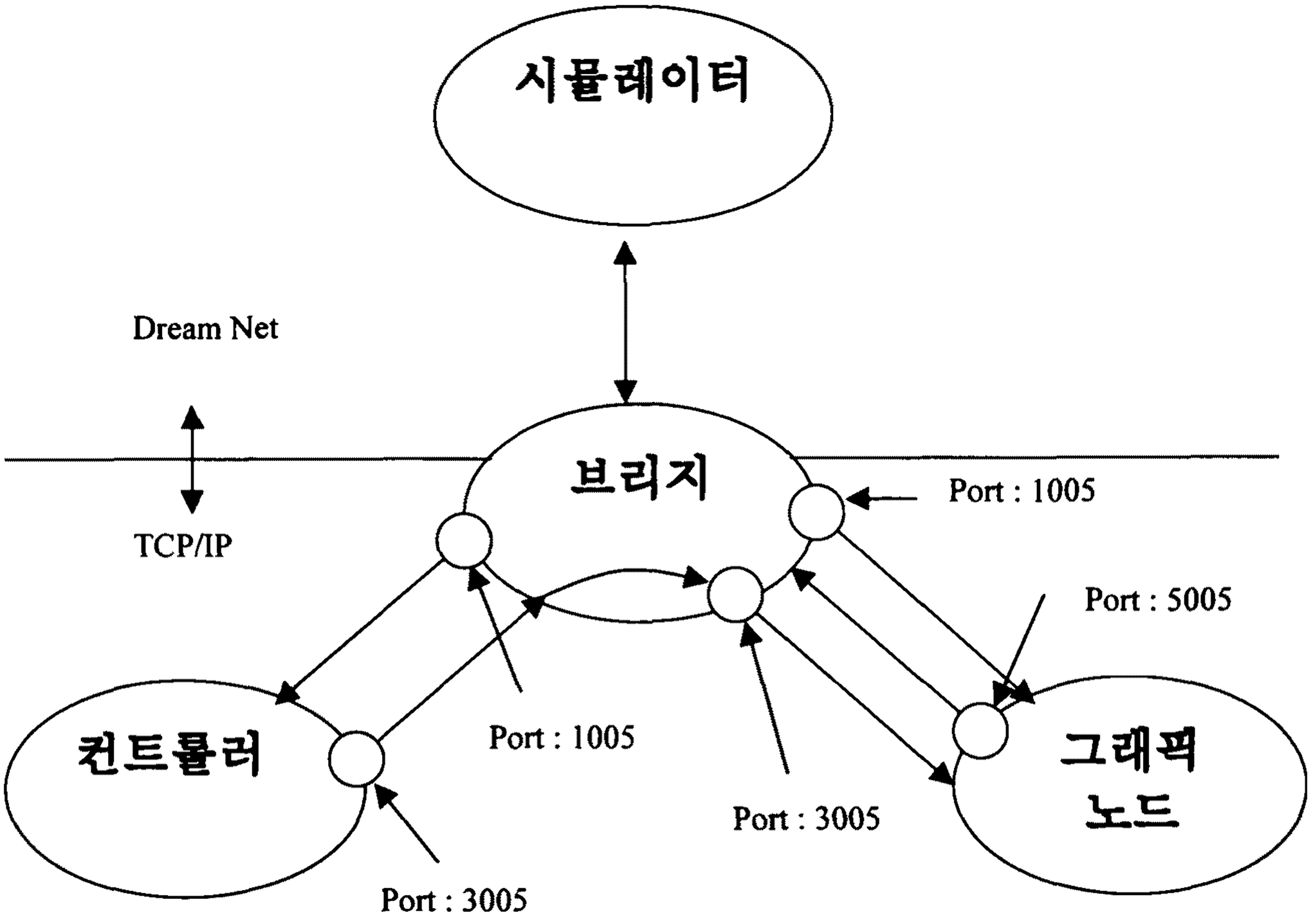


그림 10 시뮬레이터 노드간의 통신 개략도

제 3 절 압연공정 다스탠드 시뮬레이터의 구축

1. 개요

본 연구는 실시간 객체지향 모델을 기반으로 한 압연공정의 자동 판 두께 제어장치(Automatic Gauge Controller, AGC) 의 실시간 시뮬레이터의 개발에 관한 것이다. AGC 는 압연공정의 수학적 모델링에 기초한 제어 알고리즘을 이용하고 있기 때문에 모델링 오차의 존재 시에는 응답 특성이 크게 변화될 뿐만 아니라 전체 제어 시스템에도 영향을 미치게 되기 때문에 이의 개발 시 시뮬레이션의 활용이 필수적이다.

본 시뮬레이터는 2 차년도에 개발한 단일 스탠드 압연공정을 다 스탠드로 확장하였고 2 차원 그래픽에서 3 차원 그래픽으로 확장 하였다.

시뮬레이션 모델은 압연공정의 기능적 알고리즘 뿐 아니라 압연공정 각 서브모델의 시간적 제약조건을 포함하고 있으며 각 서브모델 간의 데이터 및 시간적 동기화(synchronization) 방법을 제공한다. 시뮬레이션 모델은 실제 압연공정 제어기의 제어주기와 동일한 주기 및 시간 제약조건으로 실행됨으로써 실제상황과 더욱 근접한 모의실험을 행할 수 있다.

본 시스템은 세 부분으로 구성되는데 첫번째 구성은 압연공정 환경 시뮬레이터이다. Pay-Off Reel(POR), Tension Reel(TR) 및 Work Roll1(WR1), Work Roll2(WR2)로 구성된 압연공정을 압연소재가 통과하면서 압연되는 과정을 모의하는 장치다. POR, TR 및 WR 들은 물리(phisics)법칙에 따라 구동이 되며 압연소재는 작업롤(WR)을 통과하면서 기계공학적인 압연원리에 따라 압연된다. 이들 압연환경은 각 서브모델에 부착된 센서(sensor)에 의해 주기적으로 감지되며 이 값은 실시간 제어기로 전송된다.

두 번째 구성은 압연공정 실시간 제어기이다. 압연공정의 상황을 감시하여 이상상황의 발생을 사전에 제거하고 일관성 있는 운영을 유지시키는 속도제

어기(Speed Controller)와 작업롤에 인가되는 소재의 입력 두께에 따른 압하력을 감지하여 원하는 출력 두께를 생성하기 위한 롤갭(Roll Gap)을 산출하여 소재를 압연하는 자동 판 두께 제어기로 이루어 진다. 제 1 구성과 제 2 구성은 LAN 상에서 각각의 컴퓨터에서 구현되며 실시간 주기에 의해 구동 되며 각 모델간의 시뮬레이션 행위는 모델에 기술된 시간제약에 따라 동기 된다.

세 번째 구성은 그래픽 노드 이다. 여기서는 압연공정 시뮬레이션 환경을 삼차원 객체로 구성하여 시뮬레이션의 수행에 따른 모의행위를 사용자에게 보여주며 시뮬레이션의 각종 변수를 삼차원 객체상의 그래프로 표현하여 사용자에 대한 시각적 효과를 증진시킨다. 또한 여기에 시뮬레이터의 시동 정지 등의 사용자 제어를 구현하여 시뮬레이션의 수행을 관찰하면서 사용자에 의한 시뮬레이터 제어가 가능하게 한다.

2. 시뮬레이션 모델 설계

가. 압연공정 모델링

시뮬레이션 시스템의 구성은 압연 공정 및 압연공정 실시간 제어기로 나뉘어 진다. 압연 공정부는 압연현상이 발생하는 롤스탠드(Roll Stand)부, 소재의 풀림 및 감김이 일어나는 Pay-off reel 부와 Tension reel 부로 이루어지고, 압연공정 제어기는 압연 스케쥴 및 압연공정부의 궤환 신호를 이용하여 원하는 두께의 소재를 얻을 수 있도록 각종 제어신호를 생성하는 제어시스템으로 구성된다. 압연공정부에 대한 입력신호에는 작업 롤 간격 기준신호(^{Sp}), 롤 속도 기준신호(^{V_m}), Pay-off reel 구동모터의 기준 전류신호(^{i_p}) 및 Tension reel 구동모터의 기준 전류신호(^{i_t})등이 있으며, 출력신호에는 입출력 소재두께(H, h), 입출측 장력(^{T_i}, ^{T_o}), 압하력(P) 및 롤 간격 (S)등이 있다. 압연공정 제어부기

입력신호에는 원하는 소재의 출측 두께, 입측 장력 기준신호, 출측 장력 기준신호 및 압연공정부로부터의 여러 가지 궤환 신호들이 있으며, 출력 신호들은 압연공정부의 입력 신호들과 동일하다.

그림 11에 압연공정 시뮬레이션 모델링의 개요도를 나타내었다. 압연공정은 시뮬레이션 환경인 압연공정과 AGC 제어기로 구분할 수 있고 다시 압연공정은 Pay-off reel, Work roll1, Work roll2 및 Tension reel로 구분할 수 있으며 AGC 제어기는 속도 제어기와 AGC1 및 AGC2로 구분할 수 있다. 즉 전체 시스템은 압연공정 4개, AGC 제어기 3등 7개의 객체로 분리할 수 있다. 이들 객체들은 모두 실시간 객체로서 각각을 RTO로 대응시킬 수 있고 각각의 RTO는 각각 고유한 주기로 구동 되는 SpM과 외부로부터의 메시지에 의해 구동 되는 SvM을 갖는다.

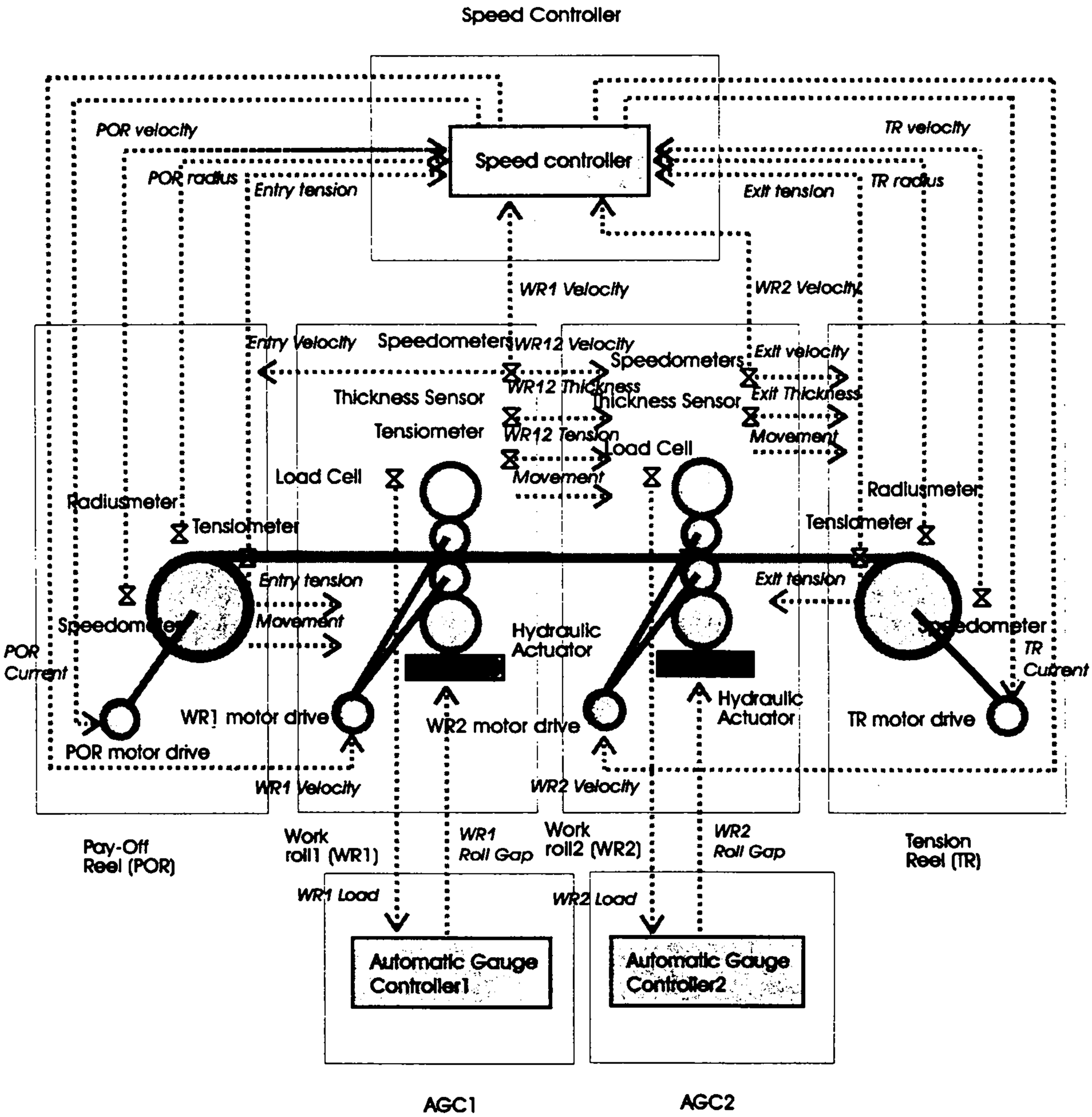


그림 11 압연공정 시뮬레이션 모델

Pay-Off Reel 객체는 POR 반지름 센서(Radiusmeter), POR 속도 센서(Speedometer) 및 입측 장력 센서(Entry Tensiometer)등 3 개의 센서로부터 POR 상태정보를 수집하며 이들을 주기적으로 속도 제어기(Speed Controller) 객체로 전송한다. 속도 제어기에서 전송한 전류신호는 POR 모터(motor)가 받아서 적절한 속도를 생성하며 이를 통해 일정한 장력을 유지하게 된다. Work Roll 객체들은 압하력 센서(Load Cell), WR 속도 센서, 입측 소재 두께 센서(Entry Thickness Sensor), 출측 소재 두께 센서(Exit Thickness Sensor), 입측 속도 센서 및 출측 속도 센서를 통해 WR 상태정보를 수집하여 속도 제어기, AGC, POR 및 TR 객체등에 전송한다. Load Cell 로 부터는 Work roll 의 압하력을 읽어들이고 이를 AGC 가 받아서 적절한 작업롤 간격(Roll Gap)을 생성하여 Work Roll 에 전송한다. Tension Reel 객체는 TR Radiusmeter, TR Speedometer 및 Exit Tensiometer 등 3 개의 센서로 부터 TR 상태정보를 수집하며 이들을 주기적으로 Speed Controller 객체로 전송한다. Speed Controller 에서 계산한 전류신호를 TR Motor 가 받아서 적절한 속도를 생성하며 이를 통해 일정한 장력을 유지하게 된다. Speed Controller 객체는 POR, WR1, WR2, TR 객체들의 센서로 부터 메시지를 수집하여 각 객체의 소재 장력을 일정하게 유지하기 위한 계산을 통해 전류값을 생성하며 이를 각각의 객체에 전송한다. 각각의 AGC 객체는 각 Work roll 의 Load Cell 로 부터 압하력 정보를 수집하여 Target Thickness 을 제공하기 위한 유압 엑츄에이터(Hydraulic Actuator)의 이송량을 계산 한다. 이 계산의 결과에 의해 Roll Gap 명령치를 Work Roll 에 송신한다.

나. 압연공정 RTO 의 분할

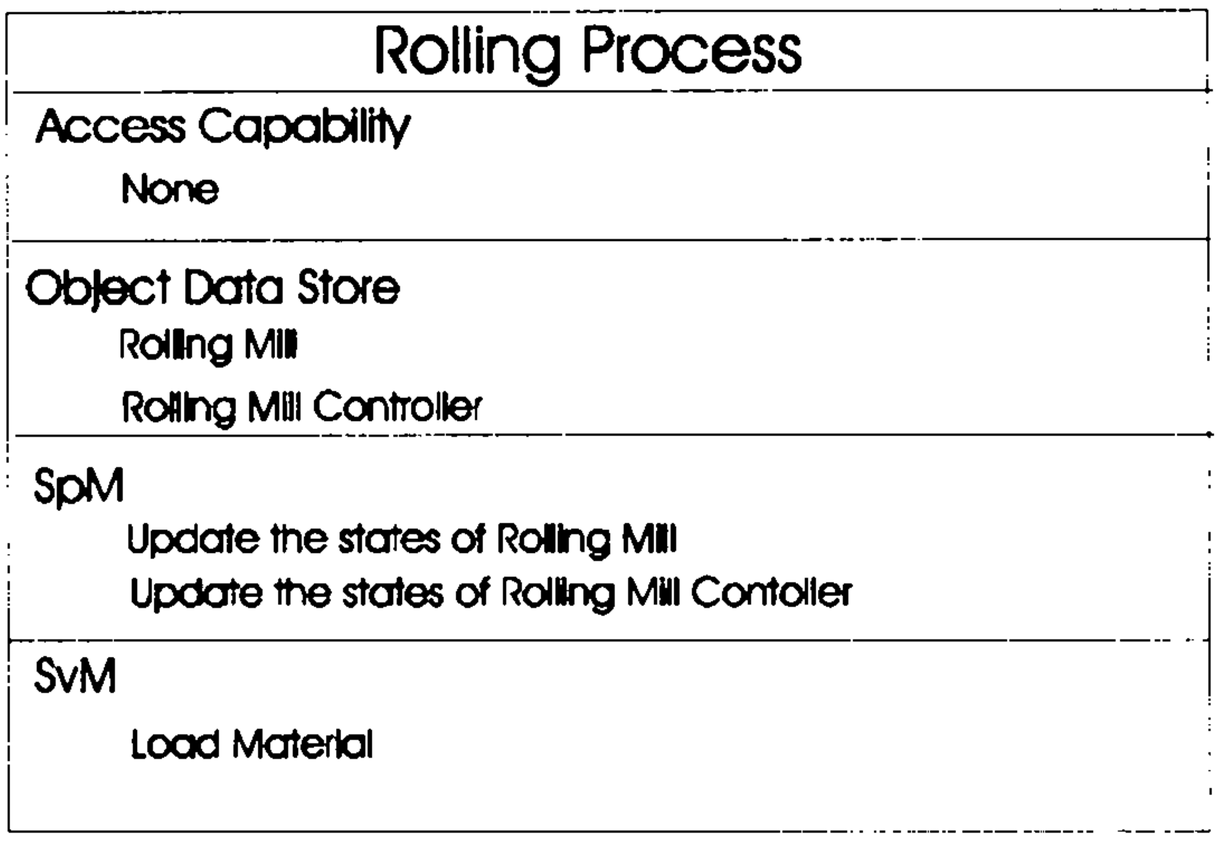


그림 12 압연공정 최상위 실시간 객체

그림 12 에 압연공정 최상위(Top-level) 실시간 객체를 나타내었다. RTO 모델링 방법은 전체 시스템을 하나의 RTO 로 기술하면서 시작한다. RTO 모델의 이해를 돕기 위하여 그림 12 에 표시된 표기법을 간단히 설명한다. RTO 는 4 개의 부분으로 나누어 진다. 접근 능력(Access Capability) 부분은 본 RTO 가 호출하는 타 RTO 를 기술한다. 여기에는 호출하는 RTO 명 및 메소드 명을 기술하고 전송하는 메시지 타입을 기술한다. 이 부분은 구현시 클라이언트(client) 및 서버(server) RTO 간의 메시지 전송 채널(channel)을 형성해 준다. 객체 데이터 공간(Object Data Store)은 RTO 내의 공유데이터 구조를 나타낸다. 이 명세는 각각의 ODS 데이터 구조가 read-only 또는 read-write 목적으로 접근될 수 있는지도 지정한다. 이것은 실행될 필요가 있는 객체 메소드들 간의 가능한 데이터 충돌을 감지 할 수 있게 한다. SpM 은 실시간 시계에 의하여 구동 되는 메소드를 나타낸다. 이 부분에는 메소드의 실행 시각, 종료 시각 및 실행 주기를 기술한다. 실시간 시계는 메소드에 기술된 어떤 값에 도달할 때 메소드의 실행을 구동하는 장치가 되며 각 메소드는 마감시간을 가진다. SvM 은 메시지에 의해 구동 되는 메소드를 나타낸다. 이 부분에서는 타 RTO 로 부터 메시

지를 수신하여 메소드를 구동하게 되며 메소드의 마감시간을 가질 수 있다.

대상 공정을 하나의 RTO 로 표현하면 다른 RTO 를 호출하는 Access Capability 는 없다. Object Data Store 는 대상 공정을 크게 두 부분으로 나눈 Rolling Mill 과 Rolling Mill Controller 가 된다. SpM 은 Object Data Store 의 두 데이터가 자기자신을 주기적으로 상태를 갱신(update)하는 것으로 표시할 수 있고, SvM 은 타 RTO 로 부터 메시지에 의해 구동되는 메소드 인데 이것은 외부로부터 장입되는 소재로 기술할 수 있다.

그림 13 은 그림 11 의 시스템 모델로부터 시뮬레이션 모델을 RTO 분할한 것 인데 그림 12 의 Top-level RTO 로 부터 Rolling Mill 과 Rolling Mill Simulator 의 두개의 RTO 로 분할 하고 이를 다시 Rolling Mill 4 개 및 Rolling Mill Simulator 3 개의 RTO 로 분할한다. 이렇게 하면 구현 가능한 단말 RTO 들을 유도하게 된다. 즉 7 개 RTO 로 본 발명의 시뮬레이터가 구성된다.

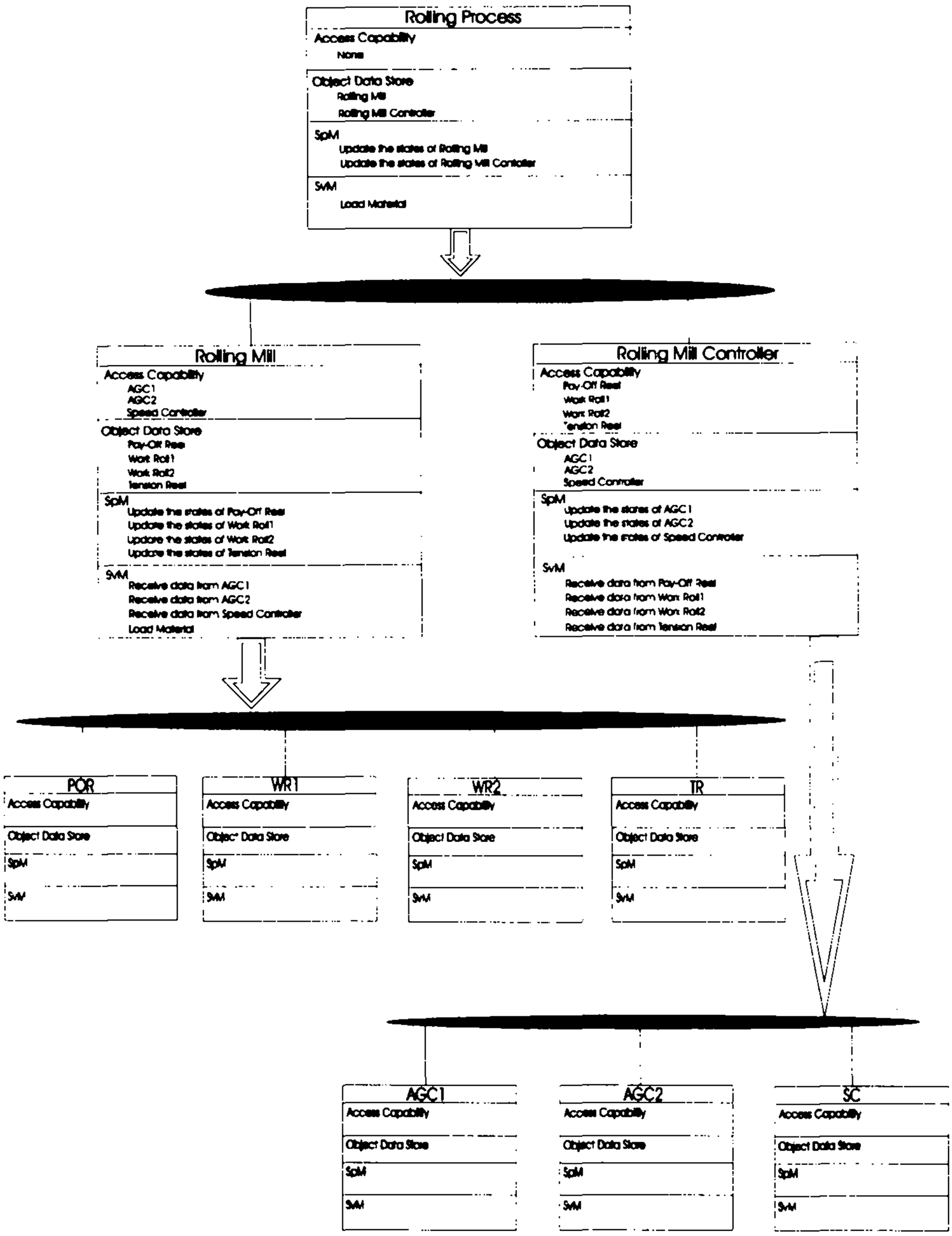


그림 13 압연공정 RTO의 분할

다. 타임라인 분석

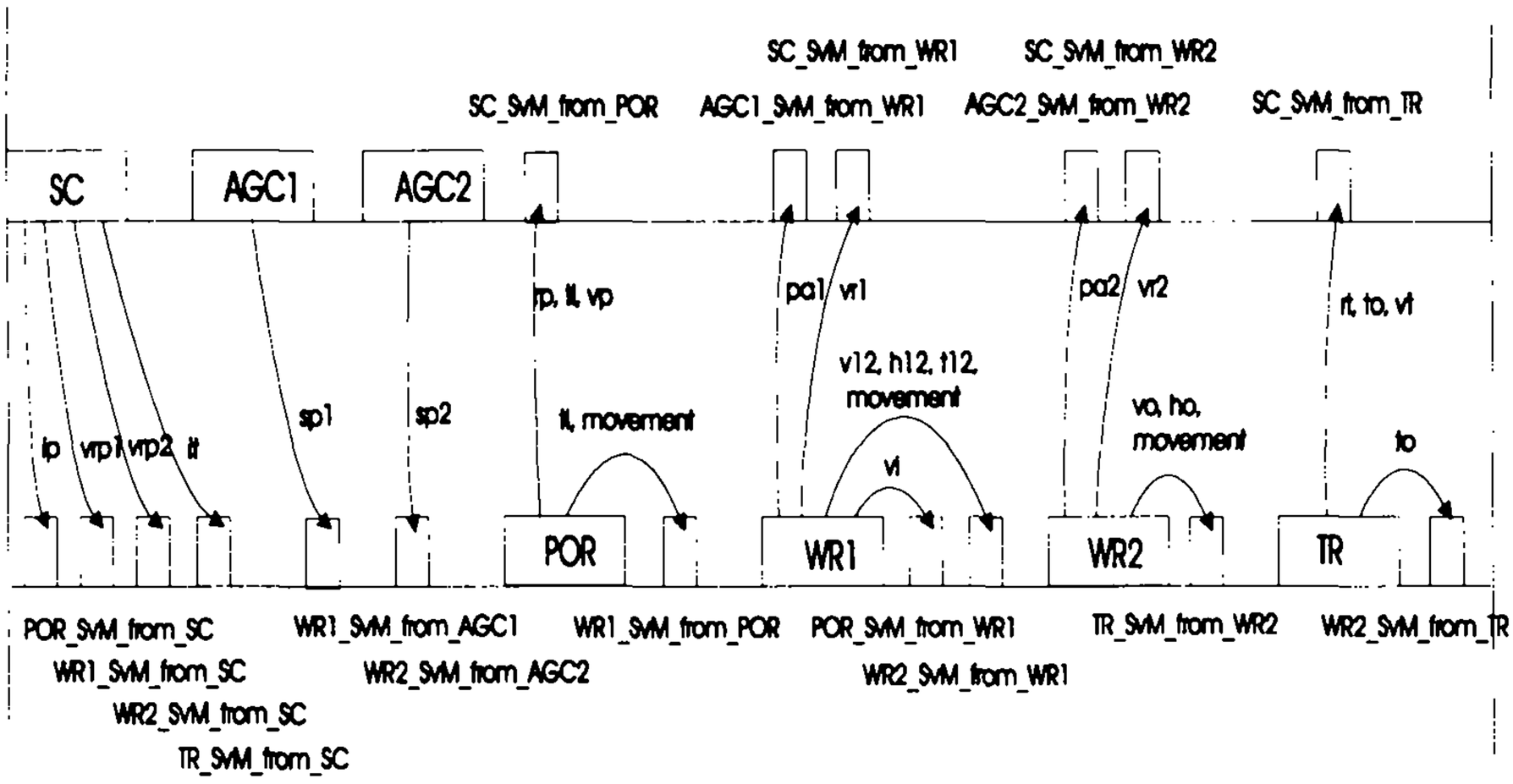


그림 14 압연공정 RTO 메소드의 타임라인

그림 14 는 RTO 메소드의 타임라인(timeline) 및 메소드간의 메시지 호출 관계를 나타낸 것이다. 본 시스템은 LAN 상의 두 컴퓨터에 구현할 수 있으며 또한 필요에 따라서 그 이상의 다수의 컴퓨터에 쉽게 분산할 수 있는데 윗 그림은 두 노드상에 제어기 및 시뮬레이터 RTO 를 할당할 예를 보인 것이다.

압연공정 시뮬레이션 모델을 두 노드에 할당하여 한 노드는 제어기 프로토타입, 다른 노드를 환경 시뮬레이터로 하면 제어기 노드에는 Speed Controller 및 AGC1, AGC2 RTO 가 할당되고 환경 시뮬레이터에는 Pay-Off Reel, Work Roll1, Work Roll2 및 Tension Reel 이 할당된다. 이들 RTO 를 각 노드별로 분리하고 각 RTO 의 메소드를 한 시뮬레이션 주기에서 구동 시간에 따라 표시하면 그림 14 와 같다.

두개의 수평선은 두개의 노드를 나타내고 두 노드상에 위치한 네모들이 각

RTO 의 메소드들이다. 네모의 폭은 마감시간내의 실행시간을 의미하며 큰 네모는 SpM 을 작은 네모는 SvM 을 나타낸다. 각각의 SpM 들은 SvM 을 호출하여 메시지를 전달하는 구조를 보여주고 있으며 SvM 의 구동 시각은 다른 SpM 이 실행되고 있지않을 때 자신이 호출된 시점임을 보여준다. 다만 POR SpM 이 WR1_SvM_from_POR 을 호출하는 것과 같이 한 노드상에서 호출이 일어날 때는 SpM 이 종료된 후에 SvM 이 실행된다.

3. 시뮬레이션 상세 설계

가. 사용자 접속

사용자 접속은 4 개의 메뉴로 구성된다. 시스템 설정에서는 압연공정의 환경변수가 입력되는데 이것은 POR, WR, TR 로 나뉘어서 각 객체의 특성값 들을 입력한다. 그림 15 에서 그림 18 까지에 이들 화면이 나타나 있다. 그림 19 에는 소재설정 화면이 나타나 있는데 여기에는 각각의 소재의 특성값 들을 입력한다. 시스템 설정 입력을 하면 시스템 설정 초기파일이 생성 또는 갱신되며 소재설정 입력 후에는 소재파일이 생성된다.

공정 프로그래밍은 설정된 공정 및 소재를 선택하는 화면으로 시스템 설정에 의해 생성된 파일을 메뉴상에서 선택할 수 있으며 그림 21 에 입력화면이 나타나 있다. 그림 22 는 공정 입력 및 수정 화면으로서 선택된 공정하에서 수행되는 공정 형태를 수정 입력할 수 있다. 공정 프로그래밍 입력을 하면 공정 파일이 생성 갱신된다.

그림 24 는 입력된 시스템 설정치에 의해 수행되는 시뮬레이션 모니터링 화면을 보여주고 있다.

주메뉴

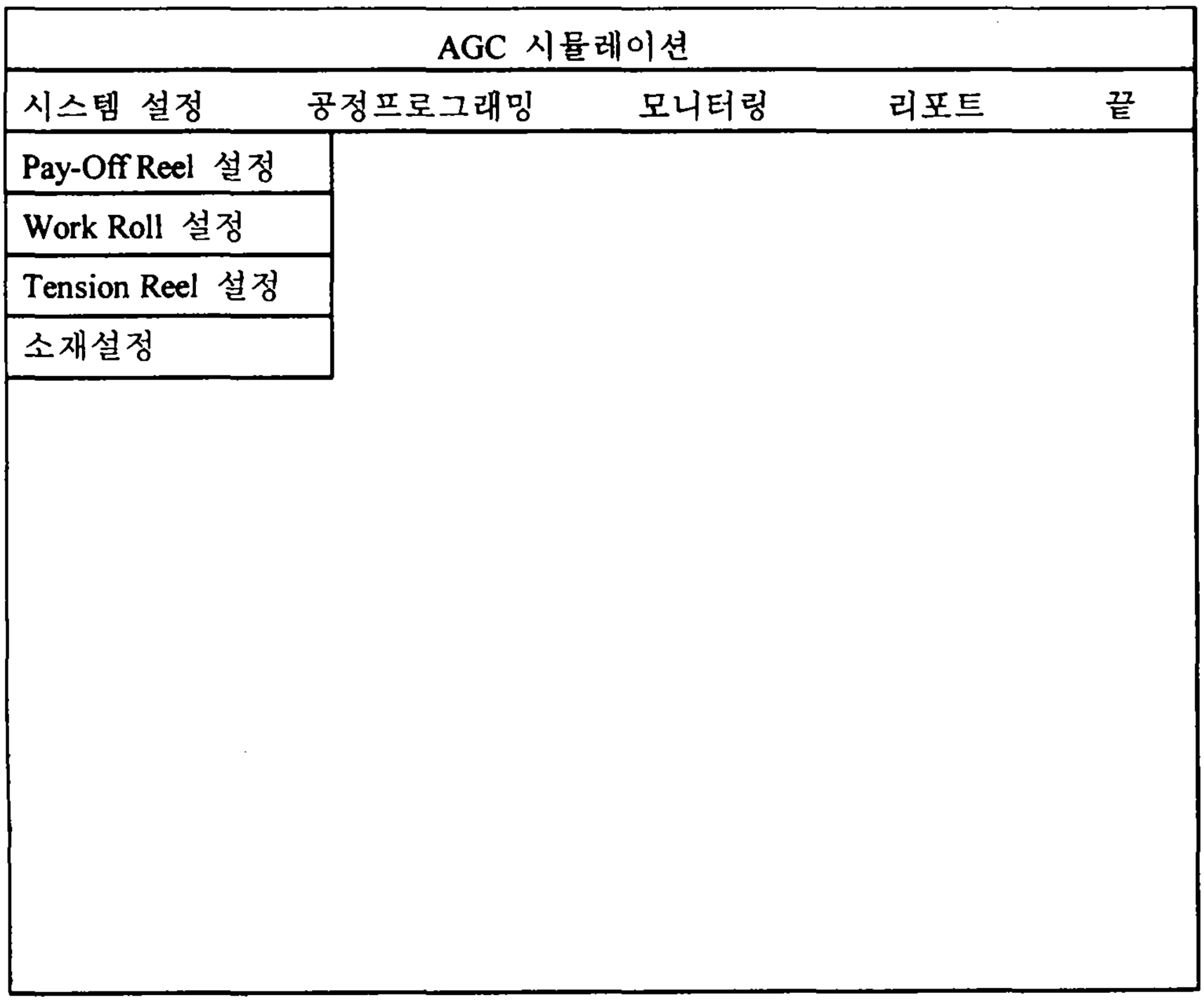


그림 15 사용자 접속 주메뉴

시스템 설정

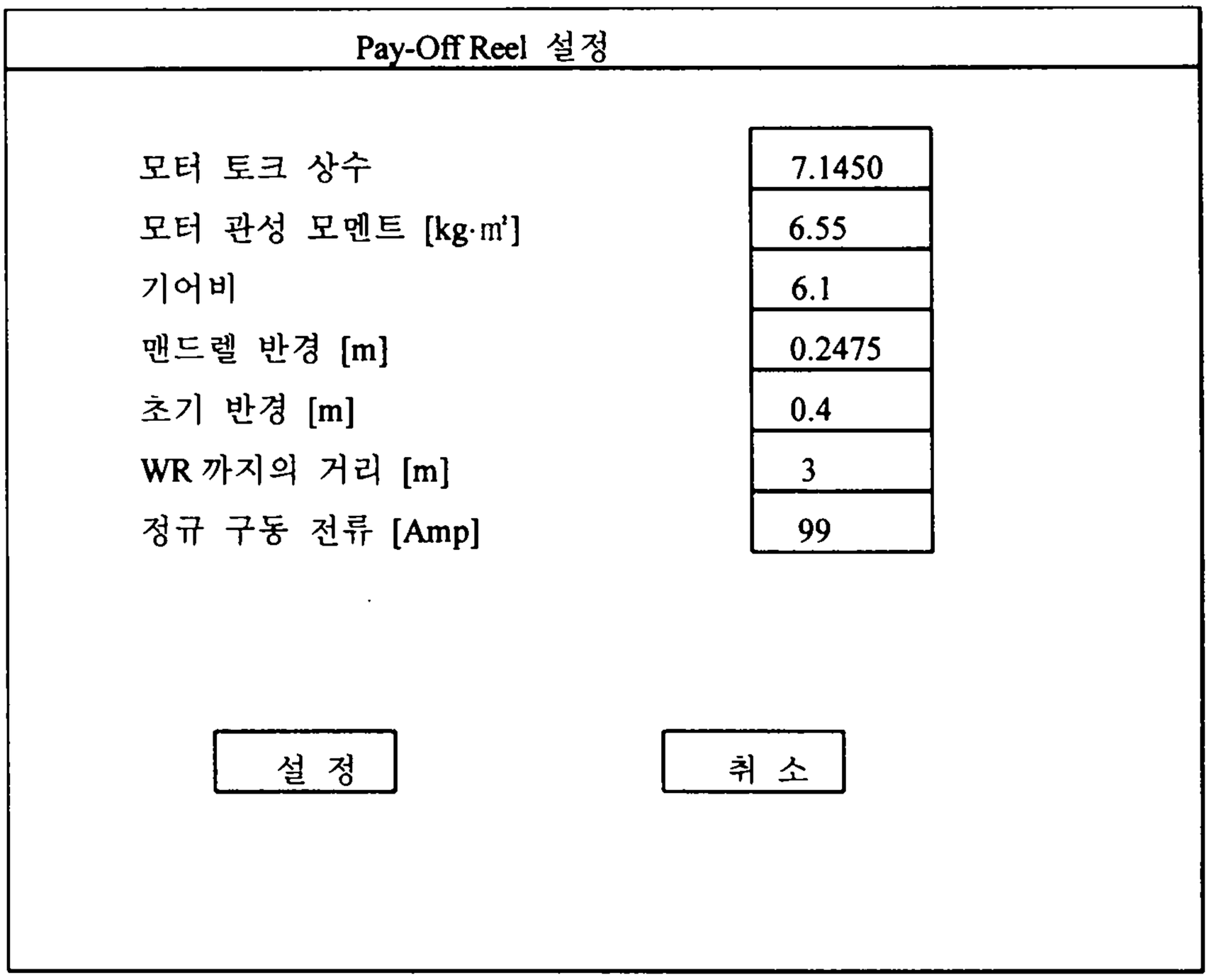


그림 16 입측롤 설정 화면

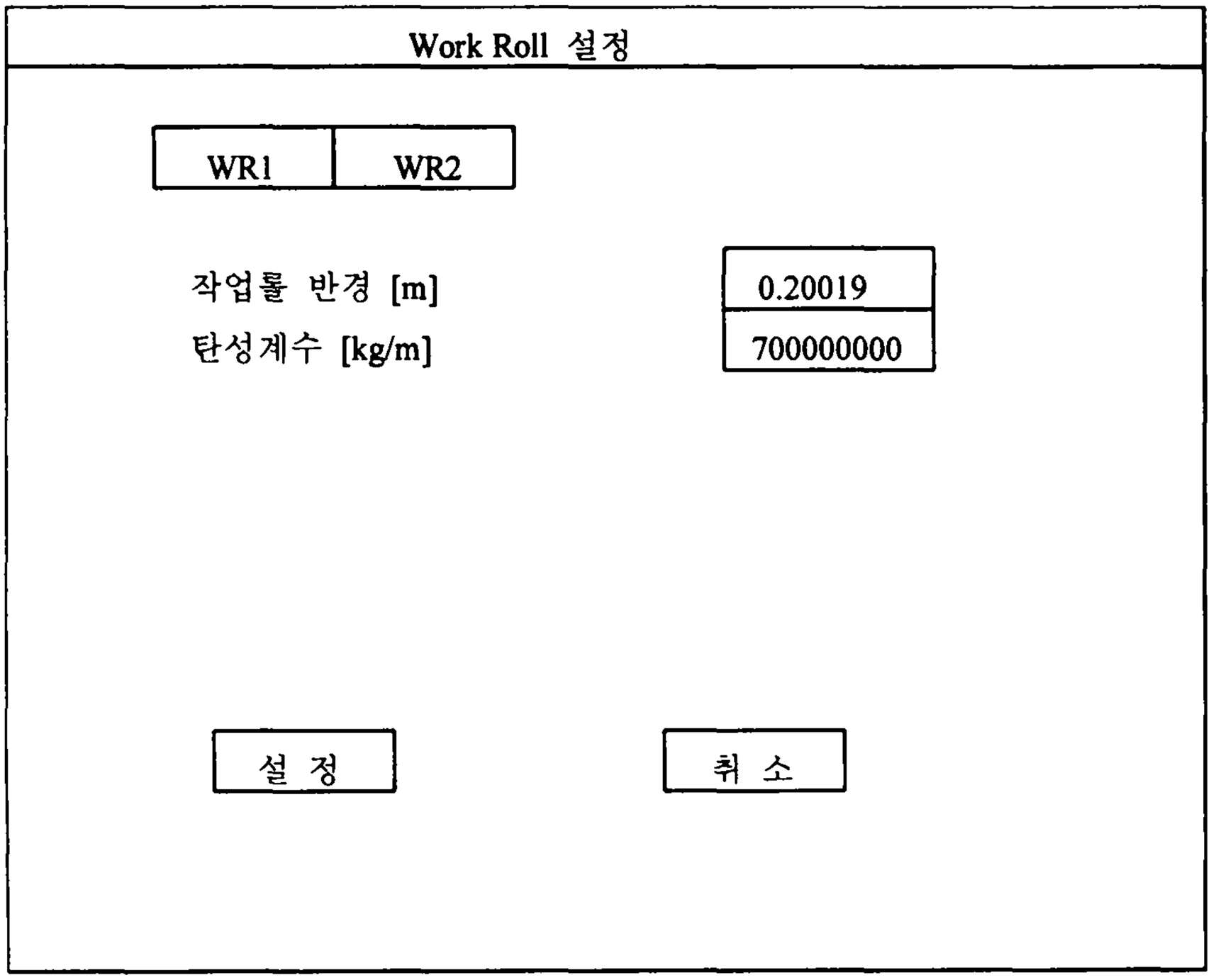


그림 17 작업롤 설정 화면

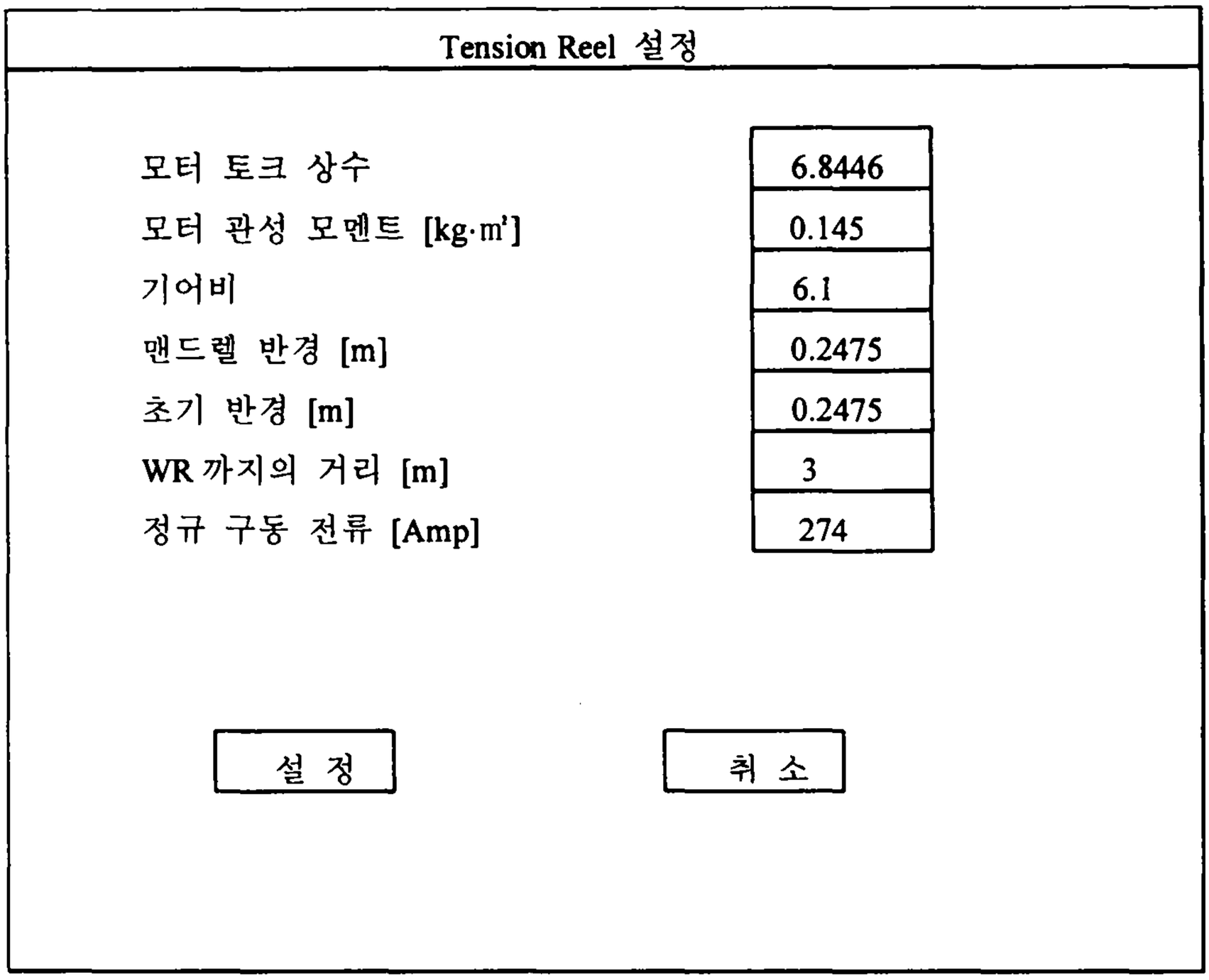


그림 18 출측롤 설정 화면

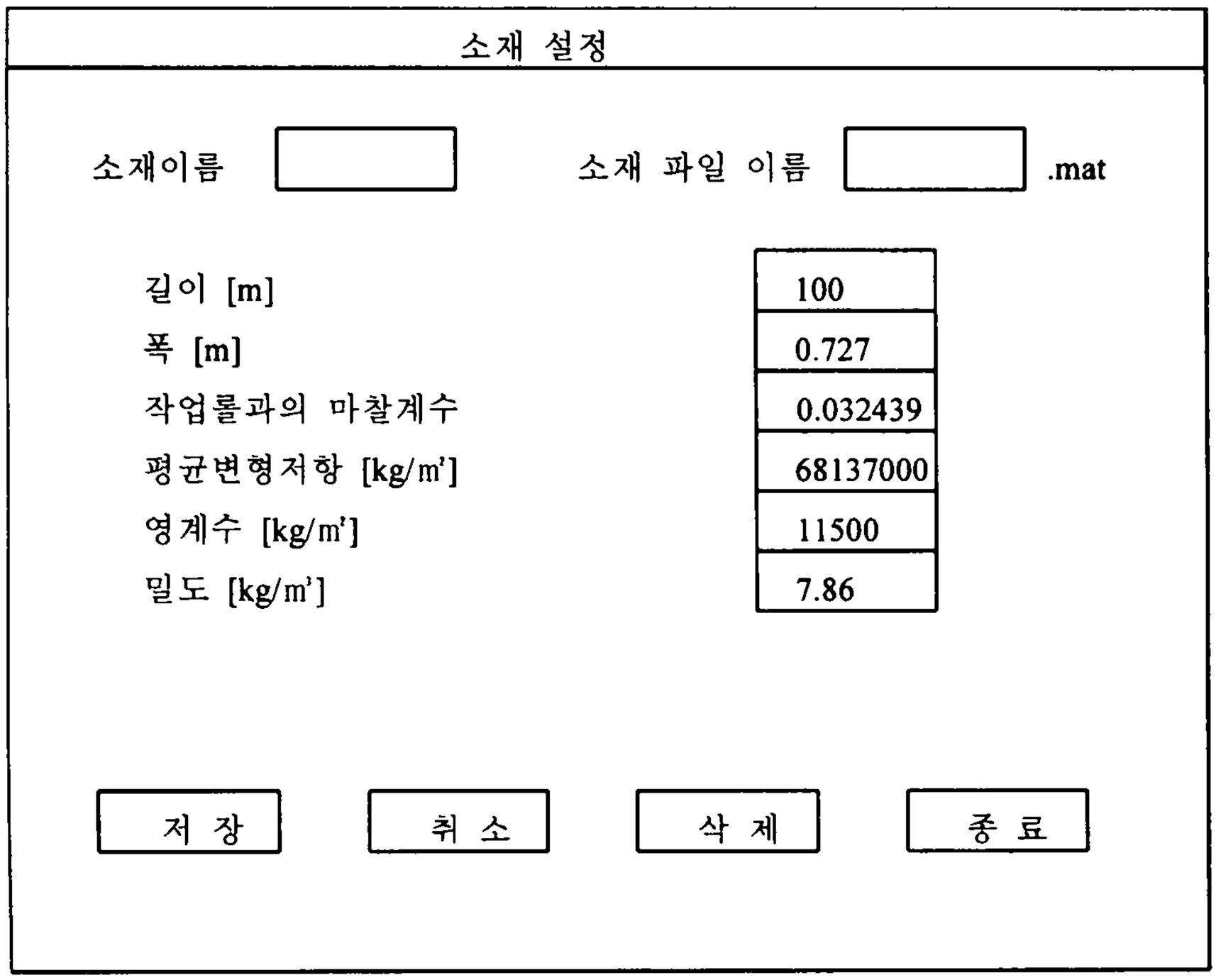


그림 19 소재 설정 화면

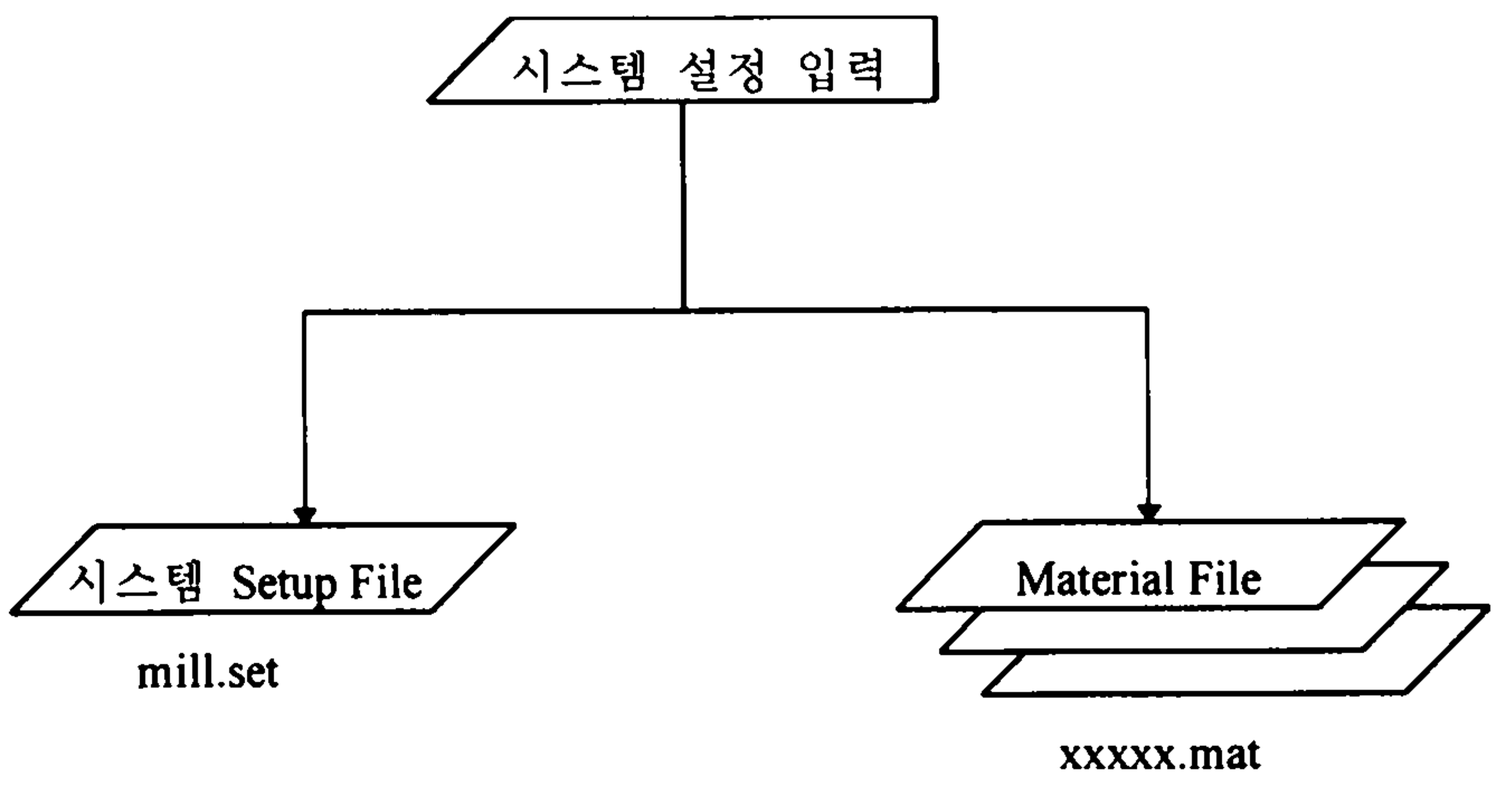


그림 20 시스템 설정 흐름도

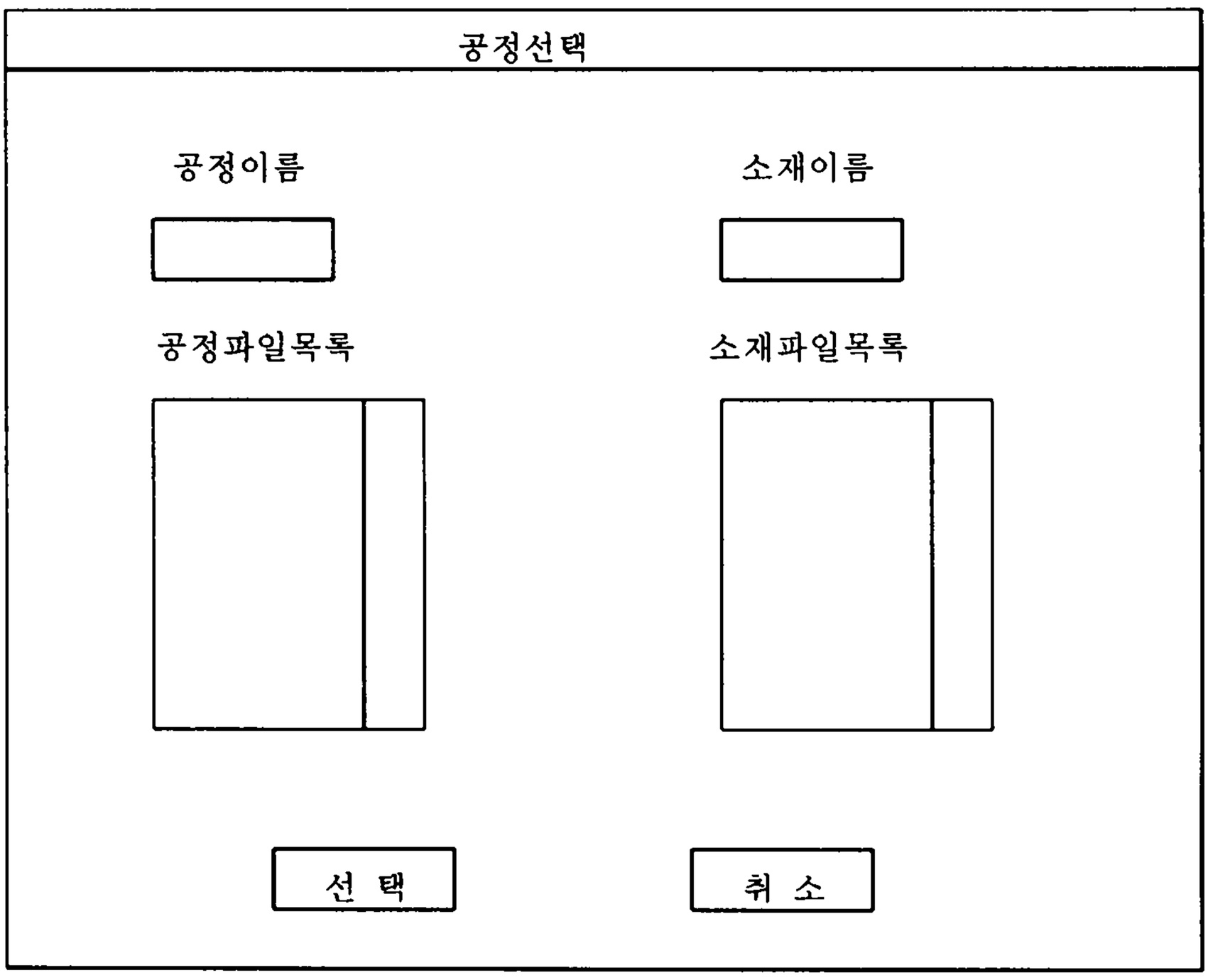


그림 21 공정 프로그램 화면

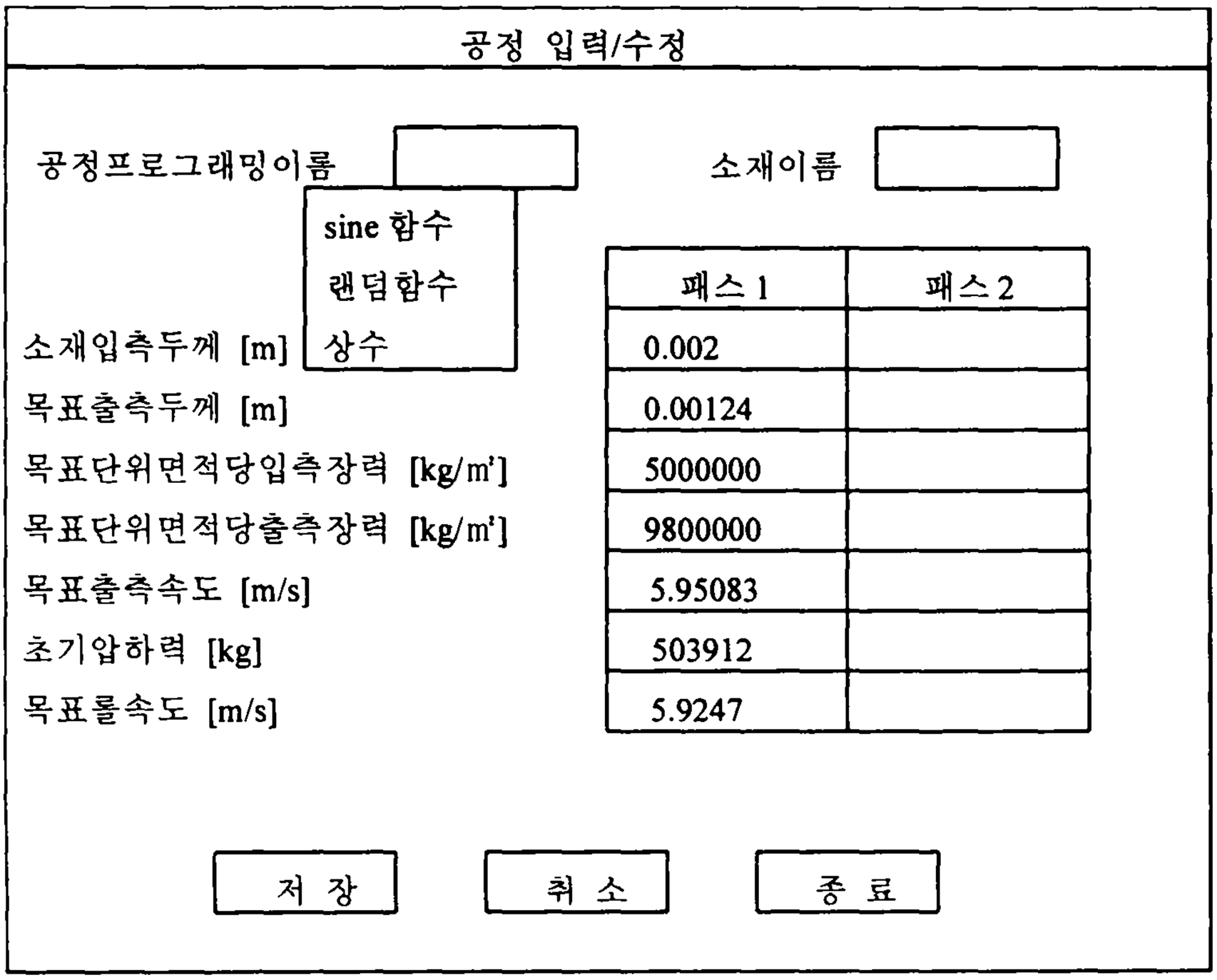


그림 22 공정 입력 화면

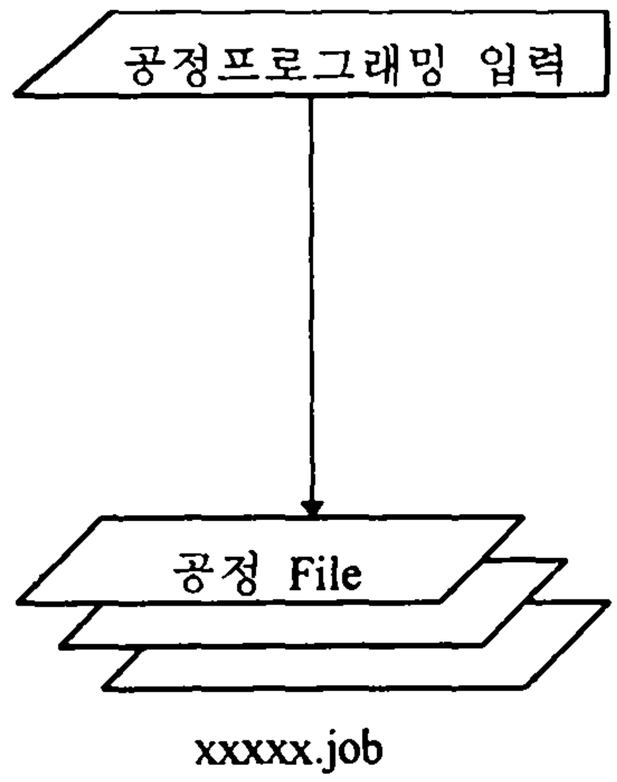
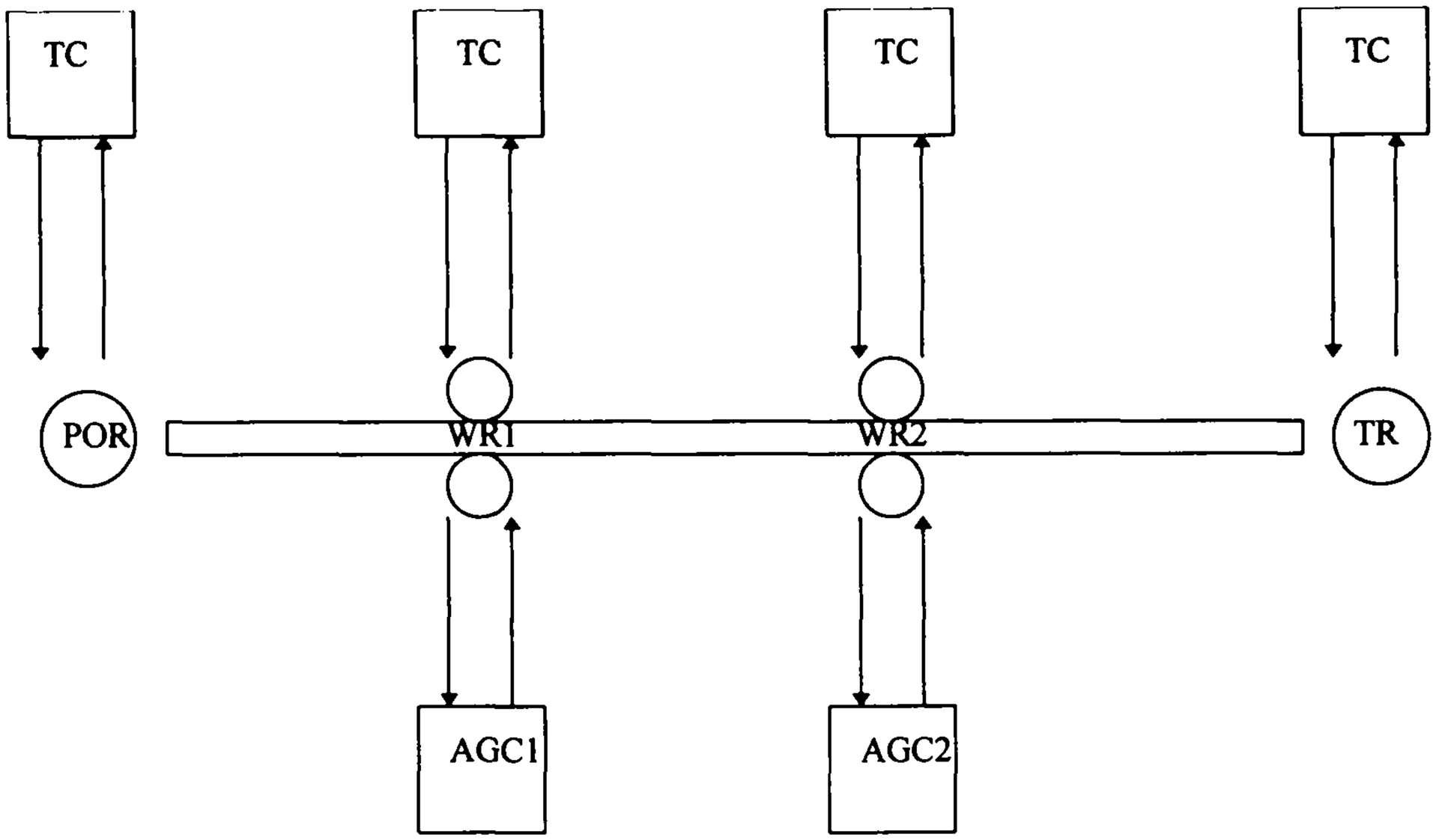
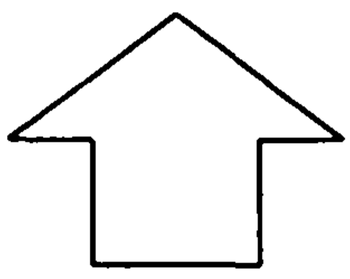
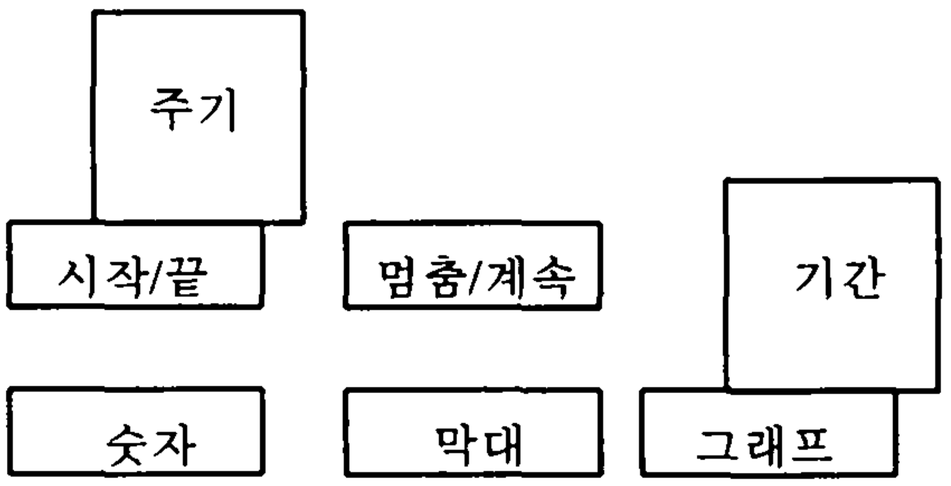


그림 23 공정 프로그래밍 흐름도

나. 모니터링







3-D Widget

(Navigation, 확대, 축소)

그림 24 모니터링 화면

다. RTO 명세

RTO를 분할하여 나오는 단말 노드의 RTO를 대상으로 설계 명세를 작성한다. RTO는 4개의 부분으로 구분된다. 그림 25에서 그림 31까지에 본 시스템의 RTO 명세가 나타나 있다. RTO의 4개 부분을 상세히 설명한다. 먼저 Access Capability는 RTO 자신이 호출하는 타 RTO들을 기술한다. 여기에는 RTO 자신이 호출하는 타 RTO들을 나열하고 각 RTO로 전송하는 메시지 타입(type)을 기술한다. 이것은 프로세서 노드내 및 프로세서 노드간에 분산되어 있는 RTO간의 전송 메시지 채널(channel)을 선언함으로써 분산 네트워크상의 RTO 간 메시지 전달을 가능케 하기 위한 것이다. 물론 분산 네트워크를 지원하는 통신 프로토콜(protocol)의 지원이 필요하다.

두번째 부분은 Object Data Store 인데 여기에는 RTO가 다루는 데이터들을 기술한다. 이들 데이터는 RTO의 속성 상수(Attribute Constant)도 있고 SpM과 SvM이 수시로 update하는 변수(variable)도 있는데 노드내의 RTO들 및 RTO 내의 메소드들이 한 노드에서 Concurrent 하게 구동 되므로 이 공유 데이터 구조(Shared Data Structure)를 update할 때 이를 모니터(monitor)해 주는 장치가 필요하다. 본 연구에서는 Dream Kernel의 CREW(Concurrent Read Exclusive Write)모니터가 이 역할을 하고 있다. 또한 데이터 선언 시 MVD(Maximum Validity Duration)를 기술해 주게 되는데 이는 데이터의 최대 유효 시간으로서 데이터가 어떤 시점으로부터 의미를 가질 수 있는 시간을 의미하며 비실시간 시스템에서는 이것이 무한대이다.

세번째 부분은 SpM이며 여기서는 시간 구동 메소드를 기술한다. 각 SpM은 각각의 구동 조건(Activation Condition)을 가지며 여기에는 RTO의 시작시간(start time), 구동주기(period) 및 마감시간(deadline)을 정의한다. 이와 같은 실시간 속성을 가진 메소드를 구동하기 위해서는 실시간 실행 지원 장치가 필요하다. 또한 SpM의 우선순위(priority)를 기술하는데 SpM은 SvM에 대해 항상

높은 우선순위를 가지나 SpM 간에는 동일한 우선순위에서는 선착순이며 서로 다른 우선순위를 가지는 SpM 이 경쟁하게 되면 우선순위에 따른 실행순서를 가지게 된다. 출력 명세(Output Specification)는 SpM 이 출력하는 메시지의 명세를 기술하며 목적지 RTO 및 서버(server) SvM 과 메시지 타입 및 마감시간을 기술한다.

네번째는 SvM 이며 메시지에 의해 구동되는 메소드를 기술한다. 각각의 SvM 은 마감시간을 가지며, Client RTO 로부터 메시지를 받고 필요한 일을 수행하며 ODS 를 update 하거나 다른 RTO 로 메시지를 출력한다. 여기서 입력 명세(Input Specification)는 전달 받는 메시지의 명세이며 출력 명세는 다른 RTO 로 전송하는 메시지의 명세이다. RTO 가 구동되면 실시간 시계의 호출에 따라 계속 살아 움직이는 SpM 과는 달리 SvM 은 메시지에 의해 호출될 때에만 살아서 움직이고 일을 끝내고 나면 죽어버리는(sleep) 버리는 수동적 프로세스(Passive Process)이다.

Pay_Off_Reel

Access Capability (to other RTO's)

- Speed_Controller (SC)

  (Receive_Info_from_Pay_Off_Reel_RTO (rp, ti, vp))

  (MSG_FROM_POR_TO_SC_TYPE)

- Work_Roll1(WR1)

  (Receive_Info_from_Pay_Off_Reel_RTO (movement_array, ti)

  (MSG_FROM_WR1_TO_SC_TYPE)

Object Data Store

Data Set: POR_Data_table: MVD: x msec

list of structure_variables

{ POR_radius (rp), POR_velocity (vp), POR_current (ip),
Entry_velocity (vi), Entry_thickness (hi),
Material_location_in_POR_area (por_entryp),
Entry_tension (ti), Movement_array (movement_array[10])

} Por_var

list of structure_constants

{ RatioOfCircumference (Pi), Width (W),
POR_array (Por_array[]), PORDistanceToWR1 (Lp),
AverageStiffness (Km), YoungCoefft (E), MaterialLen (L),
BIG_ARRAY_SZ (Por_array_num),
Movement_array_number (Movement_array_num),
PORInertiaMomentOfMotor (Jmp),
PORTorqueCoeffOfMotor (Kp), PORGearRatio (Np),
Density (D), PORMandrelRatio (Rcp)

} Por_const

SpM

SpM1 : *Simulate_Pay_Off_Reel*

- Get POR_Data_table from ODS. (gpv, pc)
- Calculate the Pay_Off_Reel radius with the parameters at just past simulation tick and send it to Speed Controller via Pay_Off_Reel Radiusmeter (via SvM request).

  (pv.rp = gpv.rp - (gpv.hi / 2.0 * pc.Pi) * (gpv.vp / gpv.rp) * delta_t)

- Calculate the entry tension with the parameters with the parameters at just past simulation tick and send it to Speed Controller and Work Roll1 via Tensiometer (via SvM request).

  (pv.ti = gpv.ti + ((pc.E * gpv.hi * pc.W) / pc.Lp) * (gpv.vi - gpv.vp) * delta_t

  rand_scope = drand()

  pv.ti = pv.ti + (pv.ti / 10.0) * rand_scope)

- Calculate the Pay_Off_Reel Velocity adding time delay and send it to Speed Controller via Speedometer (via SvM request).

  (jp = pc.Jmp + (pc.Pi * pc.D * pc.W / 2.0) *

  ((pow(gpv.rp, 4) - pow(pc.Rcp, 4)) / (pc.Np * pc.Np))

  pv.vp = gpv.vp + delta_t * ((pc.Kp * gpv.rp * gpv.ip) / (jp * pc.Np)

  + (gpv.rp * gpv.rp * gpv.ti) / (jp * pc.Np * pc.Np))

  pv.vp = pv.vp + (pv.vp / 10.0) * rand_scope

- Calculate the Material Position and update the material array in Pay_Off_Reel.
- Send Pay_Off_Reel entry tension, velocity and movement array to Work Roll1 RTO (via SvM request).
- Save radius, velocity, entry_thickness, material_location, entry_tension, movement_array to POR_Data_table in ODS. (pv)

AC: **for** T = **from RTO_START**+WARMUP_DELAY_SECS
to RTO_START+SYSTEM_LIFE_HOURS **every** PERIOD
start-during (EST, LST) **finish-by** T + DEADLINE
(In the current implementation, WARMUP_DELAY_SECS = 10 sec,
SYSTEM_LIFE_HOURS = 2 hours, PERIOD = 1000 msec (to be changed),
EST = T, LST = T+15 msec (to be changed), DEADLINE = 50 msec,
delta_t = SimulationInterval * 0.001, RTO_START = 500 msec)

OS: (to Speed Controller RTO) <deadline: 5 msec> Pay_Off_Reel radius, tension, and velocity
(to Work Roll1 RTO) <deadline: 5 msec> Pay_Off_Reel entry tension, velocity and movement array

SvM

SvM_from_SC: < **Accept-via-Service_Request_Channel-with-Delay_Bound-of** ACCEPTANCE_DEADLINE>
<**start-within** INITIATION_DEADLINE **under** MAX_REQUEST_RATE
finish-within EXECUTION_TIME_LIMIT>
(In the current implementation, ACCEPTANCE_DEADLINE = 10 msec(?),
INITIATION_DEADLINE = DEADLINE_SpM1+EXECUTION_TIME_LIMIT,
MAX_REQUEST_RATE = 1/100 msec (under this, $\mathcal{A}$ can be no queueing of SvM_from_SC requests), EXECUTION_TIME_LIMIT = 10 msec)

Receive_motor_drive_current_from_Speed_Controller_RTO

- Receive POR current from Speed Controller RTO.
  (MSG_FROM_SC_TO_POR_TYPE)
- Update current in POR_Data_table in ODS.

IC: Other SvM_from_SC invocations are not in place.

IS: current

OS: None

SvM_from_WR1: < **Accept-via-Service_Request_Channel-with-Delay_Bound-of** ACCEPTANCE_DEADLINE>
<start-within INITIATION_DEADLINE **under** MAX_REQUEST_RATE
finish-within EXECUTION_TIME_LIMIT>
(In the current implementation, ACCEPTANCE_DEADLINE = 10 msec(?),
INITIATION_DEADLINE = DEADLINE_SpM1+EXECUTION_TIME_LIMIT,
MAX_REQUEST_RATE = 1/100 msec (under this, ∽ can be no queueing of SvM_from_WR1 requests), EXECUTION_TIME_LIMIT = 10 msec)

Receive_Info_from_Work_Roll1_RTO

- Receive entry velocity and simulation stop flag from Work Roll1 RTO.
  (MSG_FROM_WR1_TO_POR_TYPE)
- Update entry velocity in POR_Data_table in ODS.

IC: Other SvM_from_WR1 invocations are not in place.

IS: Work Roll1 velocity

OS: None

그림 25 Pay-Off Reel 의 RTO 명세

Work_Roll1

Access Capability (to other RTO's)

- Pay_Off_Reel (POR)
  (Receive_Info_from_Work_Roll1_RTO (vi))
  (MSG_FROM_WR1_TO_POR_TYPE)
- Automatic_Gauge_Controller1 (AGC1)
  (Receive_roll_force_Info_from_Work_Roll1_RTO (pa1))
  (MSG_FROM_WR1_TO_AGC1_TYPE)
- Speed_Controller (SC)
  (Receive_Info_from_Work_Roll1_RTO (vr1))
  (MSG_FROM_WR1_TO_SC_TYPE)
- Work_Roll2 (WR2)
  (Receive_Info_from_Work_Roll1_RTO (v12, h12, t12, movement_array_o)

Object Data Store

Data Set: WR1_Data_table: MVD: x msec

list of structure_variables

{ WR1_roll_gap (s1), WR1_command_roll_gap (sp1),
WR1_roll_force (pa1), WR1_velocity (vr1),
WR1_command_velocity (vrp1), Entry_velocity (vi),
WR12_velocity (v12), Entry_thickness (hi)
WR12_thickness (h12), Entry_tension (ti),
WR12__tension (t12),
Material_location_in_WR1_Area (wr1_entryp),
Input_movement_array (movement_array_i[10]),
Output_movement_array (movement_array_o[10]),
WR1_thickness_array (wr1_array[])
}

list of structure_constants

{ WR1Radius (R), WR1CoeffOfElasticity (M1),
Width (W), FrictionCoefft(Fc), AverageStiffness (Km)
MaterialLen (L), BIG_ARRAY_SZ (Wr_array_num),
Movement_array_number (Movement_array_num)
}

SpM

SpM1 : *Simulate_Work_Roll1*

- Get WR_Data_table from ODS. (gwv, wc)
- Calculate the roll_force at the Work Roll1 caused by moving material and send it to Automatic Gauge Controller RTO via Load Cell (via SvM request).

  (r = (gwv.hi - gwv.h12) / gwv.hi

  qp = 1.08 + (1.79 * wc.Fc * r * sqrt(1.0 - r) * sqrt(wc.R / gwv.h12)) - 1.02 * r

  ti_per_area = gwv.ti / (wc.W * gwv.hi)

  t12_per_area = gwv.t12 / (wc.W * gwv.h12)

  M1 = 0.5

  M2 = 0.5

  pt = 1.0 - ((M1 * ti_per_area) + (M2 * t12_per_area)) / wc.Km

  wv.pa1 = wc.W * wc.Km * qp * pt * sqrt(wc.R * (gwv.hi - gwv.h12)))
- Calculate the velocity adding time delay and send it to Speed Controller (via SvM request).

  (rand_scope = drand()

  wv.vr1 = gwv.vrp1 + (gwv.vrp1 / 10.0) * rand_scope)
- Calculate the roll_gap adding time delay.

  (wv.s1 = gwv.sp1 + (gwv.sp1 / 10.0) * rand_scope)
- Calculate the WR12 velocity of material.

  (Cc = 0.5

  fa = (sqrt(r) / 2.0) - (1.0 / wc.Fc) * sqrt(gwv.h12 / wc.R)

  * (((2.0 * r) / (2.0 -r)) - ((t12_per_area - ti-per_area) / wc.Km)) * Cc

  f = fa * fa

  wv.v12 = gwv.vr1 * (f + 1.0))
- Calculate the Entry velocity of material.

  (wv.vi = (gwv.h12 / gwv.hi) * gwv.v12
- Send Entry velocity to Pay-Off Reel RTO via Entry Speedometer (via SvM request)
- Calculate the WR12 Tension of material

  (wv.t12 = gwv.t12 + ((wc.E * gwv.h12 * wc.W) / wc.L12) * (gwv.v12 - gwv.vi) * delta_t)
- Calculate the material position
  - Update material_location by using WR12_velocity.
  - Copy the input_movement_array to thickness_array.
  - Update entry_thickness from the thickness_array.
  - Calculate and update WR12_thickness using command_roll_gap.
  - Copy the portion of thickness_array to output_movement_array.
- Send WR12 velocity, WR12 tension, WR12 thickness and output_movement_array to WR2 RTO (via SvM request).
- Set WR1_Data_table to ODS. (wv)

AC: **for** T = **from RTO_START**+WARMUP_DELAY_SECS

to **RTO_START**+SYSTEM_LIFE_HOURS **every** PERIOD

start-during (EST, LST) **finish-by** T + DEADLINE

(In the current implementation, WARMUP_DELAY_SECS = 10 sec,

SYSTEM_LIFE_HOURS = 2 hours, PERIOD = 1000 msec (to be changed),

EST = T, LST = T+15 msec (to be changed), DEADLINE = 50 msec,

delta_t = SimulationInterval * 0.001, RTO_START = 600)

OS: (to Automatic Gauge Controller1 RTO) <deadline: 5 msec> roll_force

(to Speed Controller RTO) <deadline: 5 msec> velocity

(to Pay-Off Reel RTO) <deadline: 5 msec> entry_vel

(to WR12 Reel RTO) <deadline: 5 msec> WR12 velocity, WR12 tension,

WR12 thickness and output_movement_array

SvM

SvM_from_SC: < **Accept-via-Service_Request_Channel-with-Delay_Bound-of** ACCEPTANCE_DEADLINE>

<**start-within** INITIATION_DEADLINE **under** MAX_REQUEST_RATE

finish-within EXECUTION_TIME_LIMIT>

(In the current implementation, ACCEPTANCE_DEADLINE = 10 msec(?),

INITIATION_DEADLINE = DEADLINE_SpM1+EXECUTION_TIME_LIMIT,

MAX_REQUEST_RATE = 1/100 msec (under this, $\mathscr{A}$ can be no queueing of SvM_from_SC requests), EXECUTION_TIME_LIMIT = 10 msec)

Receive_motor_drive_command_velocity_from_Speed_Controller_RTO

- Receive command velocity and speedup from Speed Controller RTO. (MSG_FROM_SC_TO_WR1_TYPE)
- Update command velocity and speedup in ODS.

IC: Other SvM_from_SC invocations are not in place.

IS: command velocity

OS: None

SvM_from_AGC1: < **Accept-via-Service_Request_Channel-with-Delay_Bound-of** ACCEPTANCE_DEADLINE>

<**start-within** INITIATION_DEADLINE **under** MAX_REQUEST_RATE

finish-within EXECUTION_TIME_LIMIT>

(In the current implementation, ACCEPTANCE_DEADLINE = 10 msec(?),

INITIATION_DEADLINE = DEADLINE_SpM1+EXECUTION_TIME_LIMIT,

MAX_REQUEST_RATE = 1/100 msec (under this, $\mathscr{A}$ can be no queueing of SvM_from_AGC1 requests), EXECUTION_TIME_LIMIT = 10 msec)

Receive_roll_gap_Info_from_Automatic_Gauge_Controller1_RTO

- Receive roll_gap from Automatic Gauge Controller1 RTO.
  (MSG_FROM_AGC1_TO_WR1_TYPE)
- Update roll_gap in ODS.

IC: Other SvM_from_AGC1 invocations are not in place.

IS: roll_gap

OS: None

SvM_from_POR: < **Accept-via-Service_Request_Channel-with-Delay_Bound-of** ACCEPTANCE_DEADLINE>
<start-within INITIATION_DEADLINE **under** MAX_REQUEST_RATE
finish-within EXECUTION_TIME_LIMIT>
(In the current implementation, ACCEPTANCE_DEADLINE = 10 msec(?), INITIATION_DEADLINE = DEADLINE_SpM1+EXECUTION_TIME_LIMIT, MAX_REQUEST_RATE = 1/100 msec (under this, ∞ can be no queueing of SvM_from_POR requests), EXECUTION_TIME_LIMIT = 10 msec)

Receive_Info_from_Pay_Off_Reel_RTO

- Receive POR_entry_tension, POR_velocity and POR_movement_array from Pay-Off Reel RTO.
  (MSG_FROM_POR_TO_WR1_TYPE)
- Update POR_velocity and copy POR_entry_tension to entry_tension and POR_movement_array to input_movement_array in ODS.

IC: Other SvM_from_POR invocations are not in place.

IS: POR_entry_tension and POR_movement_array

OS: None

그림 26 Work Roll1 의 RTO 명세

Work_Roll2

Access Capability (to other RTO's)

- Automatic_Gauge_Controller2 (AGC2)

  (Receive_roll_force_Info_from_Work_Roll2_RTO (pa2))

  (MSG_FROM_WR2_TO_AGC2_TYPE)

- Speed_Controller (SC)

  (Receive_Info_from_Work_Roll2_RTO (vr2))

  (MSG_FROM_WR2_TO_SC_TYPE)

- Tension_Reel (TR)

  (Receive_Info_from_Work_Roll1_RTO (vo, ho, movement_array_o)

Object Data Store

Data Set: WR2_Data_table: MVD: x msec

list of structure_variables

{ WR2_roll_gap (s2), WR2_command_roll_gap (sp2),
WR2_roll_force (pa2), WR2_velocity (vr2),
WR2_command_velocity (vrp2), WR12_velocity (v12),
Exit_velocity (vo), WR12_thickness (h12)
Exit_thickness (ho), WR12_tension (t12),
Exit_tension (to),
Material_location_in_WR2_Area (wr2_entryp),
Input_movement_array (movement_array_i[10]),
Output_movement_array (movement_array_o[10]),
WR2_thickness_array (wr2_array[])
}

list of structure_constants

{ WR2Radius (R), WR2CoeffOfElasticity (M2),
Width (W), FrictionCoefft(Fc), AverageStiffness (Km)
MaterialLen (L), TRDistanceToWR2 (Lt),
BIG_ARRAY_SZ (Wr_array_num),
Movement_array_number (Movement_array_num)
}

SpM

SpM1 : *Simulate_Work_Roll2*

- Get WR_Data_table from ODS. (gwv, wc)
- Calculate the roll_force at the Work Roll1 caused by moving material and send it to Automatic Gauge Controller RTO via Load Cell (via SvM request).

  (r = (gwv.h12 - gwv.ho) / gwv.h12

  qp = 1.08 + (1.79 * wc.Fc * r * sqrt(1.0 - r) * sqrt(wc.R / gwv.ho)) - 1.02 * r

  t12_per_area = gwv.t12 / (wc.W * gwv.h12)

  to_per_area = gwv.to / (wc.W * gwv.ho)

  M1 = 0.5

  M2 = 0.5

  pt = 1.0 - ((M1 * t12_per_area) + (M2 * to_per_area)) / wc.Km

  wv.pa2 = wc.W * wc.Km * qp * pt * sqrt(wc.R * (gwv.h12 - gwv.ho)))
- Calculate the velocity adding time delay and send it to Speed Controller (via SvM request).

  (rand_scope = drand()

  wv.vr2 = gwv.vrp2 + (gwv.vrp2 / 10.0) * rand_scope)
- Calculate the roll_gap adding time delay.

  (wv.s2 = gwv.sp2 + (gwv.sp2 / 10.0) * rand_scope)
- Calculate the Exit velocity of material.

  (Cc = 0.5

  fa = (sqrt(r) / 2.0) - (1.0 / wc.Fc) * sqrt(gwv.ho / wc.R)

  * (((2.0 * r) / (2.0 -r)) - ((to_per_area - t12-per_area) / wc.Km)) * Cc

  f = fa * fa

  wv.vo = gwv.vr2 * (f + 1.0))
- Calculate the material position
  - Update material_location by using Exit_velocity.
  - Copy the input_movement_array to thickness_array.
  - Update WR12_thickness from the thickness_array.
  - Calculate and update exit_thickness using command_roll_gap.
  - Copy the portion of thickness_array to output_movement_array.
- Send Exit velocity, Exit thickness and output_movement_array to TR RTO (via SvM request).
- Set WR2_Data_table to ODS. (wv)

AC: **for** T = **from RTO_START**+WARMUP_DELAY_SECS

to RTO_START+SYSTEM_LIFE_HOURS **every** PERIOD

start-during (EST, LST) ***finish-by*** T + DEADLINE

(In the current implementation, WARMUP_DELAY_SECS = 10 sec,

SYSTEM_LIFE_HOURS = 2 hours, PERIOD = 1000 msec (to be changed),

EST = T, LST = T+15 msec (to be changed), DEADLINE = 50 msec,

delta_t = SimulationInterval * 0.001, RTO_START = 700 msec)

OS: (to Automatic Gauge Controller2 RTO) <deadline: 5 msec> roll_force

(to Speed Controller RTO) <deadline: 5 msec> velocity

(to Tension Reel RTO) <deadline: 5 msec> exit velocity,

exit thickness and output_movement_array

SvM

SvM_from_SC: < **Accept-via-Service_Request_Channel-with-Delay_Bound-of** ACCEPTANCE_DEADLINE>

<**start-within** INITIATION_DEADLINE **under** MAX_REQUEST_RATE

finish-within EXECUTION_TIME_LIMIT>

(In the current implementation, ACCEPTANCE_DEADLINE = 10 msec(?), INITIATION_DEADLINE = DEADLINE_SpM1+EXECUTION_TIME_LIMIT, MAX_REQUEST_RATE = 1/100 msec (under this, ∞ can be no queueing of SvM_from_SC requests), EXECUTION_TIME_LIMIT = 10 msec)

Receive_motor_drive_command_velocity_from_Speed_Controller_RTO

- Receive command velocity and speedup from Speed Controller RTO. (MSG_FROM_SC_TO_WR2_TYPE)
- Update command velocity and speedup in ODS.

IC: Other SvM_from_SC invocations are not in place.

IS: command velocity

OS: None

SvM_from_AGC2: < **Accept-via-Service_Request_Channel-with-Delay_Bound-of** ACCEPTANCE_DEADLINE>

<**start-within** INITIATION_DEADLINE **under** MAX_REQUEST_RATE

finish-within EXECUTION_TIME_LIMIT>

(In the current implementation, ACCEPTANCE_DEADLINE = 10 msec(?), INITIATION_DEADLINE = DEADLINE_SpM1+EXECUTION_TIME_LIMIT, MAX_REQUEST_RATE = 1/100 msec (under this, ∞ can be no queueing of SvM_from_AGC2 requests), EXECUTION_TIME_LIMIT = 10 msec)

Receive_roll_gap_Info_from_Automatic_Gauge_Controller2_RTO

- Receive roll_gap from Automatic Gauge Controller2 RTO. (MSG_FROM_AGC2_TO_WR2_TYPE)
- Update roll_gap in ODS.

IC: Other SvM_from_AGC2 invocations are not in place.

IS: roll_gap

OS: None

SvM_from_WR1: < **Accept-via-Service_Request_Channel-with-Delay_Bound-of** ACCEPTANCE_DEADLINE>

<start-within INITIATION_DEADLINE **under** MAX_REQUEST_RATE

finish-within EXECUTION_TIME_LIMIT>

(In the current implementation, ACCEPTANCE_DEADLINE = 10 msec(?), INITIATION_DEADLINE = DEADLINE_SpM1+EXECUTION_TIME_LIMIT, MAX_REQUEST_RATE = 1/100 msec (under this, ∽ can be no queueing of SvM_from_WR1 requests), EXECUTION_TIME_LIMIT = 10 msec)

Receive_Info_from_Work_Roll1_RTO

- Receive WR12_tension, WR12_velocity, WR12_thickness and WR12_movement_array from Work_Roll1 RTO.
  (MSG_FROM_WR1_TO_WR2_TYPE)
- Update WR12_tension, WR12_velocity, WR12_thickness and copy WR1_output_movement_array to WR2_input_movement_array in ODS.

IC: Other SvM_from_WR1 invocations are not in place.

IS: WR12_ tension, WR12_velocity and WR12_movement_array

OS: None

SvM_from_TR: < **Accept-via-Service_Request_Channel-with-Delay_Bound-of** ACCEPTANCE_DEADLINE>

<start-within INITIATION_DEADLINE **under** MAX_REQUEST_RATE

finish-within EXECUTION_TIME_LIMIT>

(In the current implementation, ACCEPTANCE_DEADLINE = 10 msec(?), INITIATION_DEADLINE = DEADLINE_SpM1+EXECUTION_TIME_LIMIT, MAX_REQUEST_RATE = 1/100 msec (under this, ∽ can be no queueing of SvM_from_TR requests), EXECUTION_TIME_LIMIT = 10 msec)

Receive_Info_from_Tension_Reel_RTO

- Receive Exit_tension from Tension_Reel RTO.
  (MSG_FROM_TR_TO_WR2_TYPE)
- Update Exit_tension in ODS.

IC: Other SvM_from_TR invocations are not in place.

IS: Exit_tension

OS: None

그림 27 Work Roll2 의 RTO 명세

Tension_Reel

Access Capability (to other RTO's)

- Speed_Controller (SC)

  (Receive_Info_from_Tension_Reel_RTO (rt, to, vt))

  (MSG_FROM_TR_TO_SC_TYPE)

- Work_Roll2(WR2)

  (Receive_Info_from_Work_Roll2_RTO (to))

  (MSG_FROM_TR_TO_WR2_TYPE)

Object Data Store

Data Set: TR_Data_table: MVD: x msec

list of structure_variables

{ TR_radius (rt), TR_velocity (vt), TR_current (it),
Exit_velocity (vo), Exit_thickness (ho),
Material_location_in_TR_area (tr_entryp),
Exit_tension (to), Movement_array (movement_array[10])

} Tr_var

list of structure_constants

{ RatioOfCircumference (Pi), Width (W),
TR_array (Tr_array[]), TRDistanceToWR2 (Lt),
AverageStiffness (Km), YoungCoefft (E), MaterialLen (L),
BIG_ARRAY_SZ (Tr_array_num),
Movement_array_number (Movement_array_num),
TRInertiaMomentOfMotor (Jmt),
TRTorqueCoeffOfMotor (Kt), TRGearRatio (Nt),
Density (D), TRMandrelRatio (Rct)

} Tr_const

SpM

SpM1 : *Simulate_Tension_Reel*

- Get TR_Data_table from ODS. (gtv, tc)
- Calculate the Tension_Reel radius with the parameters at just past simulation tick and send it to Speed Controller via Tension_Reel Radiusmeter (via SvM request).

  (tv.rt = gtv.rt + (gtv.ho / 2.0 * tc.Pi) * (gtv.vt / gtv.rt) * delta_t)

- Calculate the entry tension with the parameters with the parameters at just past simulation tick and send it to Speed Controller and Work Roll2 via Tensiometer (via SvM request).

  (tv.to = gtv.to + ((tc.E * gtv.ho * tc.W) / tc.Lt) * (gtv.vt - gpv.vo) * delta_t

  rand_scope = drand()

  tv.to = tv.to + (tv.to / 10.0) * rand_scope)
- Calculate the Tension_Reel Velocity adding time delay and send it to Speed Controller via Speedometer (via SvM request).

  (jt = tc.Jmt + (tc.Pi * tc.D * tc.W / 2.0) *

  ((pow(gtv.rt, 4) - pow(tc.Rct, 4)) / (tc.Nt * tc.Nt))

  tv.vt = gtv.vt + delta_t * ((tc.Kt * gtv.rt * gtv.it) / (jt * tc.Nt)

  - (gtv.rt * gtv.rt * gtv.to) / (jt * tc.Nt * tc.Nt))

  tv.vt = tv.vt + (tv.vt / 10.0) * rand_scope
- Calculate the Material Position and update the material array in Tension_Reel.
- Send Tension_Reel exit tension, velocity and movement array to Work Roll2 RTO (via SvM request).
- Save radius, velocity, entry_thickness, material_location, entry_tension, movement_array to TR_Data_table in ODS. (tv)

AC: **for** T = **from RTO_START**+WARMUP_DELAY_SECS
to RTO_START+SYSTEM_LIFE_HOURS **every** PERIOD
start-during (EST, LST) **finish-by** T + DEADLINE
(In the current implementation, WARMUP_DELAY_SECS = 10 sec,
SYSTEM_LIFE_HOURS = 2 hours, PERIOD = 1000 msec (to be changed),
EST = T, LST = T+15 msec (to be changed), DEADLINE = 50 msec,
delta_t = SimulationInterval * 0.001, RTO_START = 800)

OS: (to Speed Controller RTO) <deadline: 5 msec> Tension_Reel radius, tension, and velocity
(to Work Roll2 RTO) <deadline: 5 msec> Tension_Reel exit tension,
velocity and movement array

SvM

SvM_from_SC: < **Accept-via-Service_Request_Channel-with-Delay_Bound-of** ACCEPTANCE_DEADLINE>
<**start-within** INITIATION_DEADLINE **under** MAX_REQUEST_RATE
finish-within EXECUTION_TIME_LIMIT>
(In the current implementation, ACCEPTANCE_DEADLINE = 10 msec(?),
INITIATION_DEADLINE = DEADLINE_SpM1+EXECUTION_TIME_LIMIT,
MAX_REQUEST_RATE = 1/100 msec (under this, ~~a~~ can be no queueing of

SvM_from_SC requests), EXECUTION_TIME_LIMIT = 10 msec)

Receive_motor_drive_current_from_Speed_Controller_RTO

- Receive TR current from Speed Controller RTO.
  (MSG_FROM_SC_TO_TR_TYPE)
- Update current in TR_Data_table in ODS.

IC: Other SvM_from_SC invocations are not in place.

IS: current

OS: None

SvM_from_WR2: < **Accept-via-Service_Request_Channel-with-Delay_Bound-of** ACCEPTANCE_DEADLINE>
<start-within INITIATION_DEADLINE **under** MAX_REQUEST_RATE
finish-within EXECUTION_TIME_LIMIT>
(In the current implementation, ACCEPTANCE_DEADLINE = 10 msec(?),
INITIATION_DEADLINE = DEADLINE_SpM1+EXECUTION_TIME_LIMIT,
MAX_REQUEST_RATE = 1/100 msec (under this, ∝ can be no queueing of SvM_from_WR2 requests), EXECUTION_TIME_LIMIT = 10 msec)

Receive_Info_from_Work_Roll2_RTO

- Receive entry velocity and simulation stop flag from Work Roll2 RTO.
  (MSG_FROM_WR2_TO_TR_TYPE)
- Update entry velocity in TR_Data_table in ODS.

IC: Other SvM_from_WR2 invocations are not in place.

IS: Work Roll2 velocity

OS: None

그림 28 Tension Reel 의 RTO 명세

Speed_Controller

Access Capability (to other RTO's)

- Pay_Off_Reel (POR)

  (Receive_motor_drive_current_from_Speed_Controller_RTO (ip))

  (MSG_FROM_SC_TO_POR_TYPE)

- Work_Roll1(WR1)

  (Receive_motor_drive_command_velocity_from_Speed_Controller_RTO (vrp1))

  (MSG_FROM_SC_TO_WR1_TYPE)

- Work_Roll2(WR2)

  (Receive_motor_drive_command_velocity_from_Speed_Controller_RTO (vrp2))

  (MSG_FROM_SC_TO_WR2_TYPE)

- Tension_Reel (TR)

  (Receive_motor_drive_current_from_Speed_Controller_RTO (it))

  (MSG_FROM_SC_TO_TR_TYPE)

Object Data Store

Data Set: SC_Data_table: MVD: x msec

list of structure_ variables

{ POR_radius (rp), POR_velocity (vp), POR_current (ip),
Entry_tension (ti),
WR1_velocity (vr1), WR1_command_velocity (vrp1),
WR2_velocity (vr2), WR2_command_velocity (vrp2),
TR_radius (rt), TR_velocity (vt), TR_current (it), Exit_tension (to)

} Sc_var

list of structure_constants

{ PORMandrelRadius (Rcp), PORTorqueCoeffOfMotor (Kp),
PORGearRatio (Np), PORInertiaMomentOfMotor (Jmp),
TRMandrelRadius (Rct), TRTorqueCoeffOfMotor (Kt),
PORGearRatio (Nt), PORInertiaMomentOfMotor (Jmt),
Density (D), Width (W), RatioOfCircumference (Pi)

} Sc_const

SpM

SpM1 : *Simulate_Speed_Controller*

- Get SC_Data_table from ODS. (gsv, gsct)
- Calculate the Pay-Off Reel motor drive current and send it to

  Pay-Off Reel motor drive (via SvM request).

  (sv.ip = -gsv.ip * gsv.ti / (gsct.Kp * gsct.Np)
- Calculate the Work Roll1 motor drive command velocity and send it to

  Work Roll motor drive (via SvM request).

  (sv.vrp1 = gsv.vr1)
- Calculate the Work Roll2 motor drive command velocity and send it to

  Work Roll motor drive (via SvM request).

  (sv.vrp2 = gsv.vr2)
- Calculate Tension Reel motor drive current and send it to

  Tension Reel motor drive (via SvM request).

  (sv.it = gsv.it * gsv.to / (gsct.Kt * gsct.Nt)
- Set SC_Data_table to ODS. (sv)

AC: **for** T = **from RTO_START**+WARMUP_DELAY_SECS
to RTO_START+SYSTEM_LIFE_HOURS **every** PERIOD
start-during (EST, LST) **finish-by** T + DEADLINE
(In the current implementation, WARMUP_DELAY_SECS = 10 sec,
SYSTEM_LIFE_HOURS = 2 hours, PERIOD = 1000 msec (to be changed,
EST = T, LST = T+15 msec (to be changed), DEADLINE = 50 msec,
RTO_START = 0 msec)

OS: (to Pay-Off Reel RTO) <deadline: 5 msec> Pay-Off Reel current
(to Work Roll1 RTO) <deadline: 5 msec> Work Roll1 command velocity
(to Work Roll2 RTO) <deadline: 5 msec> Work Roll2 command velocity
(to Tension Reel RTO) <deadline: 5 msec> Tension Reel command velocity

SvM

SvM_from_POR: < **Accept-via-Service_Request_Channel-with-Delay_Bound-of** ACCEPTANCE_DEADLINE>
<**start-within** INITIATION_DEADLINE **under** MAX_REQUEST_RATE
finish-within EXECUTION_TIME_LIMIT>
(In the current implementation, ACCEPTANCE_DEADLINE = 10 msec(?),
INITIATION_DEADLINE = DEADLINE_SpM1+EXECUTION_TIME_LIMIT,
MAX_REQUEST_RATE = 2/100 msec (under this, ∽ can be no queueing of SvM_from_POR requests), EXECUTION_TIME_LIMIT = 10 msec)

Receive_Info_from_Pay_Off_Reel_RTO

- Receive velocity, radius and tension from Pay-Off Reel RTO.
  (MSG_FROM_POR_TO_SC_TYPE)
- Update velocity, radius and tension in SC_Data_table in ODS.

IC: Other SvM_from_POR invocations are not in place.

IS: Pay-Off Reel velocity, radius, and tension

OS: None

SvM_from_WR1: < **Accept-via-Service_Request_Channel-with-Delay_Bound-of** ACCEPTANCE_DEADLINE>
<start-within INITIATION_DEADLINE **under** MAX_REQUEST_RATE
finish-within EXECUTION_TIME_LIMIT>
(In the current implementation, ACCEPTANCE_DEADLINE = 10 msec(?),
INITIATION_DEADLINE = DEADLINE_SpM1+EXECUTION_TIME_LIMIT,
MAX_REQUEST_RATE = 2/100 msec (under this, $\mathscr{A}$ can be no queueing of SvM_from_WR1 requests), EXECUTION_TIME_LIMIT = 10 msec)

Receive_Info_from_Work_Roll1_RTO

- Receive velocity from Work Roll1 RTO.
  (MSG_FROM_WR1_TO_SC_TYPE)
- Update velocity in SC_Data_table in ODS.

IC: Other SvM_from_WR1 invocations are not in place.

IS: Work Roll1 velocity

OS: None

SvM_from_WR2: < **Accept-via-Service_Request_Channel-with-Delay_Bound-of** ACCEPTANCE_DEADLINE>
<start-within INITIATION_DEADLINE **under** MAX_REQUEST_RATE
finish-within EXECUTION_TIME_LIMIT>
(In the current implementation, ACCEPTANCE_DEADLINE = 10 msec(?),
INITIATION_DEADLINE = DEADLINE_SpM1+EXECUTION_TIME_LIMIT,
MAX_REQUEST_RATE = 2/100 msec (under this, $\mathscr{A}$ can be no queueing of SvM_from_WR2 requests), EXECUTION_TIME_LIMIT = 10 msec)

Receive_Info_from_Work_Roll2_RTO

- Receive velocity from Work Roll2 RTO.
  (MSG_FROM_WR2_TO_SC_TYPE)
- Update velocity in SC_Data_table in ODS.

IC: Other SvM_from_WR2 invocations are not in place.

IS: Work Roll2 velocity

OS: None

SvM_from_TR: < **Accept-via-Service_Request_Channel-with-Delay_Bound-of** ACCEPTANCE_DEADLINE>

<start-within INITIATION_DEADLINE **under** MAX_REQUEST_RATE

finish-within EXECUTION_TIME_LIMIT>

(In the current implementation, ACCEPTANCE_DEADLINE = 10 msec(?), INITIATION_DEADLINE = DEADLINE_SpM1+EXECUTION_TIME_LIMIT, MAX_REQUEST_RATE = 2/100 msec (under this, ∞ can be no queueing of SvM_from_TR requests), EXECUTION_TIME_LIMIT = 10 msec)

Receive_Info_from_Tension_Reel_RTO

- Receive velocity, radius and tension from Tension Reel RTO. (MSG_FROM_TR_TO_SC_TYPE)
- Update velocity, radius and tension in SC_Data_table in ODS.

IC: Other SvM_from_TR invocations are not in place.

IS: Tension Reel velocity, radius, and tension

OS: None

그림 29 Speed Controller 의 RTO 명세

Automatic_Gauge_Controller1

Access Capability (to other RTO's)

- Work_Roll1 (WR1)

  (Receive_roll_gap_Info_from_Automatic_Gauge_Controller1_RTO (sp1))

  (MSG_FROM_AGC1_TO_WR1_TYPE)

Object Data Store

Data Set: AGC1_Data_table: MVD: x msec

list of structure_variables

{ WR1_roll_force (pa1), WR1_sensed_roll_force (p1),
WR1_command_roll_gap (sp1)

} Agc1_var

list of structure_variables

{ WR1RollForceSetup (P10), WR1RollGapSetup (S10),
YoungCoefft (E), WR1CoeffOfElasticity (M1)

} Agc1_const

SpM

SpM1 : *Simulate_Automatic_Gauge_Controller*

- Get AGC1_Data_table from ODS. (av, act)
- Generate WR1_sensed_roll_force by using WR1_roll_force.

  (r = drand())

  (av.p = av.pa + act.E * r * delta_t)
- Calculate the WR1_command_roll_gap and send it to Hydraulic Actuator in Work Roll1 (via SvM request).

  (av.sp = act.S10 - (av.p1 - act.p10) / act.M1)
- Save sensed_roll_force and command_roll_gap to AGC_Data_table in ODS. (av)

AC: **for** T = **from RTO_START**+WARMUP_DELAY_SECS
to RTO_START+SYSTEM_LIFE_HOURS **every** PERIOD
start-during (EST, LST) **finish-by** T + DEADLINE
(In the current implementation, WARMUP_DELAY_SECS = 10 sec,
SYSTEM_LIFE_HOURS = 2 hours, PERIOD = 1000 msec (to be changed),

EST = T, LST = T+15 msec (to be changed), DEADLINE = 50 msec,

CTRL_INTERVAL = 60, delta_t = CTRL_INTERVAL * 0.001,

RTO_START = 100)

OS: (to Work Roll RTO) <deadline: 5 msec> command_roll_gap

SvM

SvM_from_WR1: < **Accept-via-Service_Request_Channel-with-Delay_Bound-of** ACCEPTANCE_DEADLINE>

<**start-within** INITIATION_DEADLINE **under** MAX_REQUEST_RATE

finish-within EXECUTION_TIME_LIMIT>

(In the current implementation, ACCEPTANCE_DEADLINE = 10 msec(?),

INITIATION_DEADLINE = DEADLINE_SpM1+EXECUTION_TIME_LIMIT,

MAX_REQUEST_RATE = 2/100 msec (under this, ∞ can be no queueing of SvM_from_WR1 requests), EXECUTION_TIME_LIMIT = 10 msec)

Receive_WR1_roll_force_Info_from_WR1RTO

- Receive WR1_roll_force from Work Roll1 RTO.

(MSG_FROM_WR1_TO_AGC1_TYPE)

- Update roll_force in ODS.

IC: Other SvM_from_WR1 invocations are not in place

IS: WR1_roll_force

OS: None

그림 30 Automatic Gauge Controller1 의 RTO 명세

Automatic_Gauge_Controller2

Access Capability (to other RTO's)

- Work_Roll2 (WR2)

  (Receive_roll_gap_Info_from_Automatic_Gauge_Controller2_RTO (sp2))

  (MSG_FROM_AGC2_TO_WR2_TYPE)

Object Data Store

Data Set: AGC2_Data_table: MVD: x msec

list of structure_variables

{ WR2_roll_force (pa2), WR2_sensed_roll_force (p2),
WR2_command_roll_gap (sp2)

} Agc2_var

list of structure_variables

{ WR2RollForceSetup (P20), WR1RollGapSetup (S20),
YoungCoefft (E), WR2CoeffOfElasticity (M2)

} Agc2_const

SpM

SpM1 : *Simulate_Automatic_Gauge_Controller*

- Get AGC2_Data_table from ODS. (av, act)
- Generate WR2_sensed_roll_force by using WR2_roll_force.

  (r = drand())

  (av.p = av.pa + act.E * r * delta_t)
- Calculate the WR2_command_roll_gap and send it to Hydraulic Actuator in Work Roll2 (via SvM request).

  (av.sp = act.S20 - (av.p2 - act.p20) / act.M2)
- Save sensed_roll_force and command_roll_gap to AGC_Data_table in ODS. (av)

AC: **for** T = **from RTO_START**+WARMUP_DELAY_SECS
to RTO_START+SYSTEM_LIFE_HOURS **every** PERIOD
start-during (EST, LST) **finish-by** T + DEADLINE
(In the current implementation, WARMUP_DELAY_SECS = 10 sec,
SYSTEM_LIFE_HOURS = 2 hours, PERIOD = 1000 msec (to be changed),
EST = T, LST = T+15 msec (to be changed), DEADLINE = 50 msec,

CTRL_INTERVAL = 60, delta_t = CTRL_INTERVAL * 0.001,

RTO_START = 200)

OS: (to Work Roll RTO) <deadline: 5 msec> command_roll_gap

SvM

SvM_from_WR2: < **Accept-via-Service_Request_Channel-with-Delay_Bound-of** ACCEPTANCE_DEADLINE>

<start-within INITIATION_DEADLINE **under** MAX_REQUEST_RATE

finish-within EXECUTION_TIME_LIMIT>

(In the current implementation, ACCEPTANCE_DEADLINE = 10 msec(?), INITIATION_DEADLINE = DEADLINE_SpM1+EXECUTION_TIME_LIMIT, MAX_REQUEST_RATE = 2/100 msec (under this, ∞ can be no queueing of SvM_from_WR1 requests), EXECUTION_TIME_LIMIT = 10 msec)

Receive_WR2_roll_force_Info_from_WR2RTO

- Receive WR2_roll_force from Work Roll2RTO. (MSG_FROM_WR2_TO_AGC2_TYPE)
- Update roll_force in ODS.

IC: Other SvM_from_WR2 invocations are not in place.

IS: WR2_roll_force

OS: None

그림 31 Automatic Gauge Controller2 의 RTO 명세

4. 시뮬레이터 구현

본 시뮬레이터의 구동 예를 보이면 그림 32 와 33 과 같다.

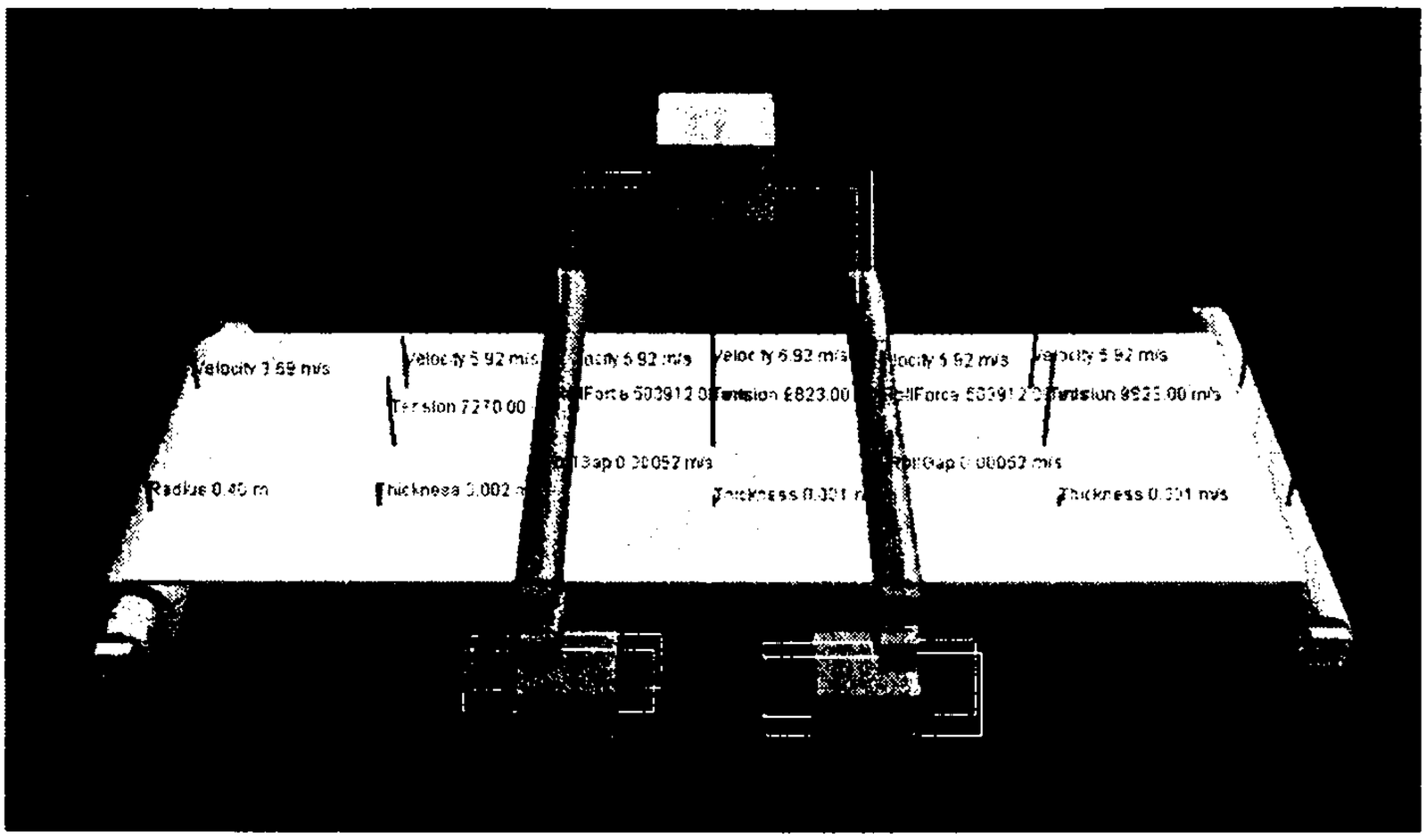


그림 32 시뮬레이터의 구동예 1

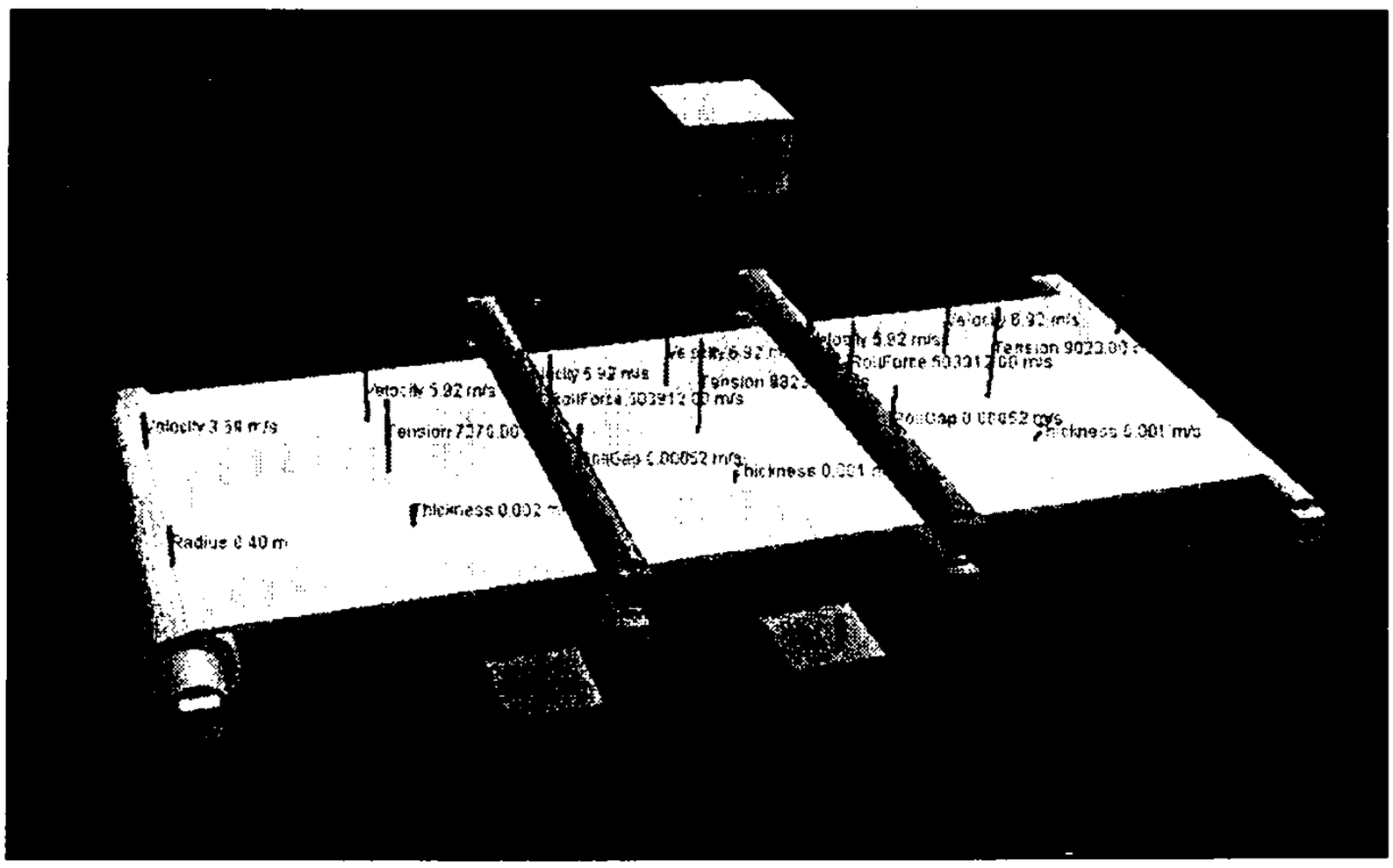


그림 33 시뮬레이터의 구동예 2

가. 시스템 매개변수 설정

본 시뮬레이터의 구동을 위한 시스템 설정 변수의 일실시예를 보이면 다음과 같다.

1) System Setup File (Mill.ini)

```
PORTorqueCoeffOfMotor=7.1450          (float)
PORInertiaMomentOfMotor=6.55          (float)
PORGearRatio=6.1                      (float)
PORMandrelRadius=0.2475               (float)
PORInitialRadius=0.4                  (float)
PORVelocitySetup=3.6895               (float)
PORDistanceToWR1=3                    (int)
PORCurrentSetup=99                    (float)

TRTorqueCoeffOfMotor=6.8446           (float)
TRInertiaMomentOfMotor=0.145          (float)
TRGearRatio=6.1                       (float)
TRMandrelRadius=0.2475                (float)
TRInitialRadius=0.2475                (float)
TRVelocitySetup=8.9508                (float)
TRDistanceToWR2=3                     (int)
TRCurrentSetup=274                    (float)
WR1Radius=0.20019                     (float)

WR1CoeffOfElasticity=700000000        (long)
WR1DistanceToWR2=3                    (int)
WR2Radius=0.20019                     (float)
WR2CoeffOfElasticity=700000000        (long)
WR2DistanceToTR=3                     (int)
```

WarmupDelaySecs=10 (int)

SystemLifeHours=2 (int)

SpMInterval=1 (int)

SimulationInterval=60 (int)

ControlInterval=60 (int)

RatioOfCircumference=3.14159 (float)

2) 소재 File (Steel.ini)

MaterialLen=100	(int)
MateralWidth=0.727	(float)
FrictionCoefft=0.032439	(float)
AverageStiffness=68137000	(long)
YoungCoefft=11500	(int)
Density=7.86	(float)

3) 공정파일 (Process.ini)

EntryThickness=0.002	(float)
EntryTensionSetup=7270.0	(float)
WR1RollForceSetup=503912	(float)
WR1VelocitySetup=5.9247	(float)
WR1RollGapSetup=0.00052	(float)
WR12Thickness=0.00124	(float)
WR12TensionSetup=8834.5	(float)
WR12VelocitySetup=6.9247	(float)
WR2RollGapSetup=0.00032	(float)
WR2RollForceSetup=403912	(float)
WR2VelocitySetup=7.9247	(float)
ExitThickness=0.001	(float)
ExitTensionSetup=9823.0	(float)

나. 개발 및 구현환경

본 시스템의 개발환경은 다음과 같다.

1) 시뮬레이터

- DOS V6
- Borland C++ V4.5
- Dream Library V3
- TNT DOS Extender

2) 그래픽 노드

- Windows 95
- Microsoft Visual C++ V4.2
- Open Inventor V2.2.1

3) Bridge

- DOS V6
- PC/TCP SDK V3.2

4) W/S

- Unix
- TCP/IP

본 시스템의 운영환경은 다음과 같다.

1) 시뮬레이터

- DOS V6
- Borland C++ V4.5
- Dream Kernel V3
- TNT DOS Extender
- Packet Driver

2) 그래픽 노드

- Windows 95
- Microsoft Visual C++ V4.2
- Open Inventor V2.2.1
- WinPacket Driver

3) Bridge

- DOS V6
- PC/TCP V3.2
- Packet Driver

4) W/S

- Unix
- TCP/IP

제 4 절 결론

전통적인 시뮬레이션은 가상 시간(Virtual Time)에 의해 시뮬레이션이 진행되며 사건(event) 처리를 위해 사건 리스트를 필요로 한다. 실시간 시뮬레이션은 실시간에 의해 시뮬레이션이 진행되며 사건 리스트를 필요로 하지 않는다. 실시간 시계를 이용함으로써 시뮬레이션 대상의 시간적 행위를 모의할 수 있다는 것은 전통적 시뮬레이션과 대비되는 특징이다.

실시간 분산 시뮬레이션을 구현함에 있어서 실시간 객체 지향(RTO) 방법은 설계 및 구현에 있어서 많은 장점이 있음을 확인하였다. 시간 구동 메소드와 메시지 구동 메소드를 구분함으로써 실시간 행위의 설계 명확성을 가져다 주었고 시뮬레이션 모델을 실시간 객체로 분할함으로써 구현의 용이성을 가져왔다. 또한 RTO 방법으로 구현함에 따라 시뮬레이션 모델의 노드 분산이 매우 용이함을 경험하였다.

실시간 객체에 기반한 시뮬레이터의 개발은 많은 장점을 가지고 있으며 이는 보다 복잡한 대규모 실시간 응용에 적합한 접근 방법으로 판단된다. 예를 들면 복잡한 로직을 가지고 있으며 매우 많은 분산 노드로부터 데이터를 실시간으로 수집, 통제 해야 하는 교통제어 시스템 모델에 본 연구의 접근 방법이 유용할 것으로 기대된다. 또한 안전이 최우선시 되는 원자력 발전소는 시스템 개발 시 설계의 명확성이 매우 필요한 분야 인데 이러한 분야에도 필요할 것으로 생각되며 공장 및 물류 자동화 시스템의 시뮬레이션 또한 본 연구의 접근 방법이 유용할 것으로 생각된다.

제 3 차년도
연차 보고서

RTS 기술개발

RTS Technology Development

Development of a Real-Time Object Model and Supporting Operating System Facilities for Real-Time Simulation

연구기관

University of California, Irvine

과 학 기 술 처

여 백

Real-Time Object Structuring and Real-Time Simulation in New-Generation Defense System Engineering

K. H. (Kane) Kim
Dept. of Electrical & Computer Engineering
University of California
Irvine, California 92697, U.S.A.
Kane@Ece.Uci.Edu

Abstract

This author believes that the *real-time object structuring* technology and the *real-time simulation* technology are among the most important computer technologies needed to support future engineering of advanced defense systems. Although the real-time object structuring technology has not yet been established in a mature form, there is no alternative to it with respect to enabling economic and reliable design of complex computer based application systems. It facilitates not only reuse of modules tested in earlier applications but also production of system designs which are easy to understand and modify. The other important technology, the real-time simulation technology, supports an advanced mode of simulation in which the simulator modules are designed to show the same timing behavior that the simulation targets do. High-fidelity real-time simulators of application environments can thus be used in effective validation of real-time computer-based control systems. Such validation has great cost and flexibility advantages and is becoming further attractive due to the on-going decrease in the costs of developing real-time simulators. The author presents one version of the real-time object structuring technology and the related real-time simulation technology along with some laboratory experiences in prototyping small-scale defense applications by using these technologies.

1. Introduction

Large-scale command-control systems needed in future air defense are the most challenging subject for the computer-based system technologists who are about to enter a new millenium. In order to meet the public expectations of the development efficiency, the costs, and the reliability of such systems, the system engineering methods different from those used in the past must be employed. This author believes that the *real-time object structuring* technology and the *real-time simulation* technology are among the most important real-time computer based system engineering technologies to be established.

First, although the real-time object structuring technology has not yet been established in a mature form [Wor94, Wor96, Wor97], there is no alternative to it with respect to enabling economic and reliable design of complex distributed and parallel computing application systems. It facilitates not only reuse of modules tested in earlier applications but also production of system designs which are easy to understand and modify.

Secondly, real-time simulation is an advanced mode of simulation in which the simulator modules are designed to show the same timing behavior that the simulation targets do [Kim96b]. High-fidelity real-time simulators of application environments can thus be used in effective validation of real-time computer-based control systems. Such validation has great cost and flexibility advantages and is becoming further attractive due to the on-going decrease in the costs of developing real-time simulators. The cost reduction is being realized not only by the reduction in the hardware costs of distributed and parallel computer systems but also by the development of easy-to-use object programming tools and stable real-time operating systems. This author believes that the real-time simulation technology will advance rapidly in coming years.

This paper presents briefly one version of the real-time object structuring technology and the related object-structured real-time simulation technology as examples of those which will be increasingly studied by the defense computing technology development community in coming years. This paper also presents some laboratory experiences in prototyping small-scale defense applications by using these technologies.

The two technologies needed in future engineering of challenging defense systems are also needed in future automation of industry and social infrastructures. However, advanced defense systems present the greatest challenges to the technology development community. Therefore, investment by the defense research and development community in these technology developments is expected to have very positive impacts on the commercial sector dealing with industry automation and social infrastructure development applications.

2. The needs for real-time object structuring in engineering of challenging defense systems

In this section, several major reasons why new real-time object structuring technologies are needed in engineering of challenging defense systems are discussed.

2.1 Design complexity

This author believes that large-scale command-control systems needed in future air defense will present greater design complexity than any other artifacts designed so far by human beings have presented. Among the factors that contribute to the design complexity are the following imposed on the systems:

(1) the stringent response time requirements,
(2) the requirements for cooperative decision-making by widely distributed dynamic subsystems, and
(3) the requirements for fusion and noise-filtering of fuzzy sensor information that is large in volume, generation rate, and variety.

Such complex systems cannot be built to possess an acceptable level of quality without using futuristic structuring techniques that have the potential of being highly effective in producing easily understandable designs of real-time distributed and parallel computing systems.

2.2 Exploitation of advanced technologies developed in the commercial sector

Defense systems needed are much smaller in quantity in comparison to the typical products in commercial markets. Experiences have shown that when the defense community develops computer technologies in complete separation from the computer technology developments in the commercial business sector for a substantial length of time, the results are generally overshadowed by similar but more reliable technologies emerging from the commercial sector.

Over the past 15 years *object-oriented* (OO) design approaches have become a common practice in development of non-real-time business data processing software due to the modularity, generality, and natural abstraction benefits that the OO approaches bring in [Dah72, Ell90, Boo91, Rum91, Sel94]. On the other hand, OO-structuring has had minimal impacts in real-time computer systems (RTCS) engineering in contrast to its pervasive use in non-RTCS engineering. This means that much of the capabilities existing in the vast business data processing software field is currently not utilized in development of RTCS's such as those needed in defense applications. It also means that the currently practiced RTCS engineering process and the current real-time application software themselves take peculiar forms unfamiliar to the vast main-stream software engineering community. The consequence is the poor economy of scale in RTCS development and the relatively low reliability of the software products except in cases of small-scale simplistic phase-locked loop control types of applications.

2.3 Reuse of modules tested in earlier applications

Module reuse is of great importance as a means of keeping future software engineering costs down. OO programming tools emerged in the last decade represent a major advance in facilitating module reuse in non-RTCS engineering fields. To facilitate module reuse in RTCS engineering, an extension of the conventional object structure and supporting tools that are effectively applicable to RTCS engineering and reengineering must be established first.

2.4 Reliability

Development of complex RTCS's in esoteric forms, which has been widely practiced up to now, is very unlikely to lead to reliable products. When large-scale defense systems have low reliability, consequences can be devastating to large communities. Also, it is impossible to test and validate many defense systems to the same degree of thoroughness that is achieved in testing of commercial non-RTCS's. Therefore, general-form design and easily understandable/analyzable design are of critical importance in producing defense systems. The reliability concerns are then essentially as the concerns on how to conquer the design complexity discussed above in Section 2.1.

2.5 A natural solution: Real-time object structuring

The arguments made above in Sections 2.1 - 2.4 lead to one conclusion. One of the most important technologies that need to be established is an extension of the existing object structuring technology that is effectively applicable to design of RTCS's and supports the following which is called the general-form timiliness-guaranteed (GT) design paradigm [Kim95a, Kim97a, Kim97b]:

(1) *General-form design*: Future real-time computing must be realized in the form of a generalization of the non-real-time computing, rather than in a form looking like an esoteric specialization. In other words, under a properly established real-time system design methodology, every practically useful non-real-time computer system must be realizable by simply filling the time constraint specification part with unconstrained default values.

(2) *Design-time guarantee of timely service capabilities of subsystems*: To meet the demands of the general public on the assured reliability of future RTCS's in safety-critical applications such as defense applications, there does not appear to be any adequate way but to require the system engineer to produce design-time guarantees for timely service capabilities of various subsystems (which will take the form of objects in object-oriented system designs). Experiences of practicing engineers indicate that testing alone is not sufficient for assuring the level of reliability of RTCS's which the customers have started demanding. It is also known that in general, verifying the full logical behavior of a sizable real-time software is not practical. However, this author believes that verification of the timing behavior is economically feasible and must be pursued. It is actually the timing behavior which presents the biggest difficulties to the system engineer relying on the testing for assuring proper behavior to a reasonable degree. The ease or difficulty of verifying the timing behavior depends on the way time constraints are specified in real-time objects.

Therefore, the essence of the GT design paradigm is to realize real-time computing in a general manner not alienating the main-stream computing industry and yet allowing system engineers to confidently produce certifiable RTCS's for safety-critical applications. Conventional object structures do not have concrete mechanisms for flexible and accurate representation of temporal behavior of complex dynamically changing systems. They need to be extended to support general-form design of RTCS's. Research activities in this direction for establishing extended object structuring technologies, which can be called *real-time object structuring* technologies, have started growing rapidly in recent years [Att91, Ish92, Kim94b, Sel94, Tak92] although most of the views held by other researchers may not have been as idealistic as this author를 view of accepting both the general-form design paradigm and the design-time guarantee of timely service capabilities as the feasible and most appropriate goals.

3. The needs for real-time simulation in engineering of challenging defense systems

Real-time simulation is an advanced accurate mode of simulation in which *the simulation objects are designed to show the same timing behavior that the simulation targets do* [Kim96b]. Efficient real-time simulation technologies are under increasing demands. Two major application areas of such technologies are discussed in this section and both application areas represent many defense applications.

3.1 Virtual reality applications

A growing application field of the real-time simulation is the virtual reality field. A virtual reality environment corresponding to a dynamic physical environment, e.g., a compartment in a moving train or a flying airplane, is not of high quality if the virtual environment does not change at the same tempo at which the physical counterpart changes. To effect precise real-time simulation, the *simulation execution engine* (a computer to run the simulation program) must be of the type that exhibits predictable and dependable timing behavior.

3.2 Cost-effective testing of RTCS's with real-time simulators of application environments

After a control computer system has been implemented, its testing typically involves interfacing it with the application environment and performing test-runs. In the case of a command-control computer system, finding or setting up a proper application environment is very expensive if not impossible. Therefore, it is often inevitable or at least highly useful to connect such control computer system to a simulator of the application environment for its validation of rather than directly proceeding to interface the computer system with the application environment. A highly desirable simulator here is one capable of accurately imitating the timing behavior of the environment, i.e., real-time simulation of the environment.

The environment simulator based testing approach has also other advantages such as great cost and flexibility advantages. Its high flexibility characteristics enables high-coverage testing involving simulation of a large variety of environment conditions. On the other hand, these cost and flexibility advantages are meaningless if the accuracy and precision of the simulation are low. They are meaningful only if accurate true real-time simulation is achieved.

Real-time simulation of complex application environments such as defense application requirements requires dependable real-time parallel and distributed computing technologies. The slow maturing of the latter has had the preventive effect on the emergence and wide use of the former. As real-time parallel and distributed computing technologies started growing faster in recent years, the real-time simulation technology has also started showing faster advances. In other words, the costs of developing real-time simulators have started decreasing fast. The cost reduction is being realized not only by the reduction in the hardware costs of distributed and parallel computer systems but also by the development of easy-to-use object programming tools and stable real-time operating systems. This author believes that the real-time simulation technology will advance even more rapidly in coming years.

3.3 Object structured real-time simulation

In order to support accurate real-time simulation, new approaches to model the simulation targets, especially those which enable multi-fidelity representation of the timing behavior of the simulation target, are needed. In our view, a preferred modeling approach for use in real-time simulation should be of object-oriented (OO) type because the modularity, generality, and natural abstraction benefits of OO approaches in non-real-time conventional simulation have been amply demonstrated in the practicing field [Dah72].

Since existing object models do not possess adequate capabilities for representing the timing behavior accurately, we are again forced to search for a proper extension of the basic object model. As mentioned in the preceding section, the object structure which possesses strong capabilities for representing the timing behavior accurately and varying degrees of precision, is needed to support cost-effective design of real-time embedded computer systems as well. Naturally, finding such extended object models has emerged as one of

the most important research issues in the real-time computing field in 1990뭽 [Kim95a, Kim97b]. Ideally, a model which is capable of uniformly and accurately representing both real-time embedded computer systems and application environments, is the most desirable.

4. An overview of the RTO.k object structuring scheme

As an example of newly emerging extensions of the conventional object structure that are aimed for supporting RTCS engineering, the real-time object structuring scheme called the RTO.k object structuring scheme, is reviewed in this section. The RTO.k object is particularly suited for flexible and yet accurate specification of the timing behavior of modeled subjects. An absract precursor to the RTO.k object was jointly formulated in late 1980's by the author and his research collaborator, Hermann Kopetz [Kop90], and it has evolved into the RTO.k object structuring scheme with a concrete syntax structure and execution semantics in recent years [Kim94a, Kim94b, Kim96a, Kim97d].

Only a brief overview of the RTO.k object structuring scheme is given in this section. More details can be found in the references. The RTO.k object, also called the *time-triggered real-time object* (TT-RTO), was devised with the ultimate goal of facilitating the idealistic design paradigm discussed in Section2, i.e., the GT design paradigm. It facilitates not only production of easily understandable and analyzable designs of RTCS's but also efficient production of real-time simulators of application environments. It thus enables uniform structuring of control computer systems and application environment simulators.

The basic structure of an RTO.k object is depicted in Figure 1. It is an extension of the conventional objects and three most important extensions are the following:

(a) *Two clearly separated groups of methods*:

For some methods of an RTO.k object, a real-time clock serves as the mechanism for triggering the method executions. These object methods whose executions are triggered as the clock reaches some values specified at design time, are called time-triggered (TT-) methods or spontaneous methods (SpM's). They are *clearly* separated from the conventional service methods (SvM's) triggered by request messages from clients. The two types of methods in an RTO.k object are different not only in the way their executions are triggered but also in that "actions to be taken at real times *which can be determined at the design time* can appear only in SpM's". Therefore, actions of the type "at constant-clock-value do S" or the type "sleep-until constant-clock-value" can appear only in SpM's.

Triggering times for SpM's must be fully specified as constants in the section called the autonomous activation condition (AAC) section during the design time. The AAC for SpM1 in Figure 1 may be: "for t = from 10am to 10:50am every 30min start-during (t, t+5min) finish-by t+10min", which has the same effect as

{"start-during (10am, 10:05am) finish-by 10:10am",
"start-during (10:30am, 10:35am) finish-by 10:40am"}.

(b) *Basic concurrency constraint* (BCC):

In order to dramatically reduce the designer's efforts in guaranteeing timely service capabilities of RTO.k objects, the execution rule which prevents conflicts between SpM's and SvM's is incorporated. Basically, *activation of an SvM triggered by a message from an external client is allowed only when potentially conflicting SpM executions are not in place.* To be exact, when a message-triggered SvM is not free of conflict with an SpM in accessing the same portion of the *object data store* (ODS), execution of the former method (SvM) must not be allowed in a time zone earmarked for a TT-execution of the latter method (SpM). This restriction is called the basic concurrency constraint (BCC). Therefore, SpM's are given higher priorities for execution over the SvM's. Note that this BCC does not impose any restriction on concurrent execution of SpM's or concurrent execution of SvM's. Therefore, executions of SpM's are not disturbed by SvM executions and possible triggering times of SpM's are fixed at the design time. At least this makes it very easy to analyze the execution time behavior of SpM's. For example, if a statement of the type "at 10am do S" appears in an SpM, its reliable execution can be easily assured.

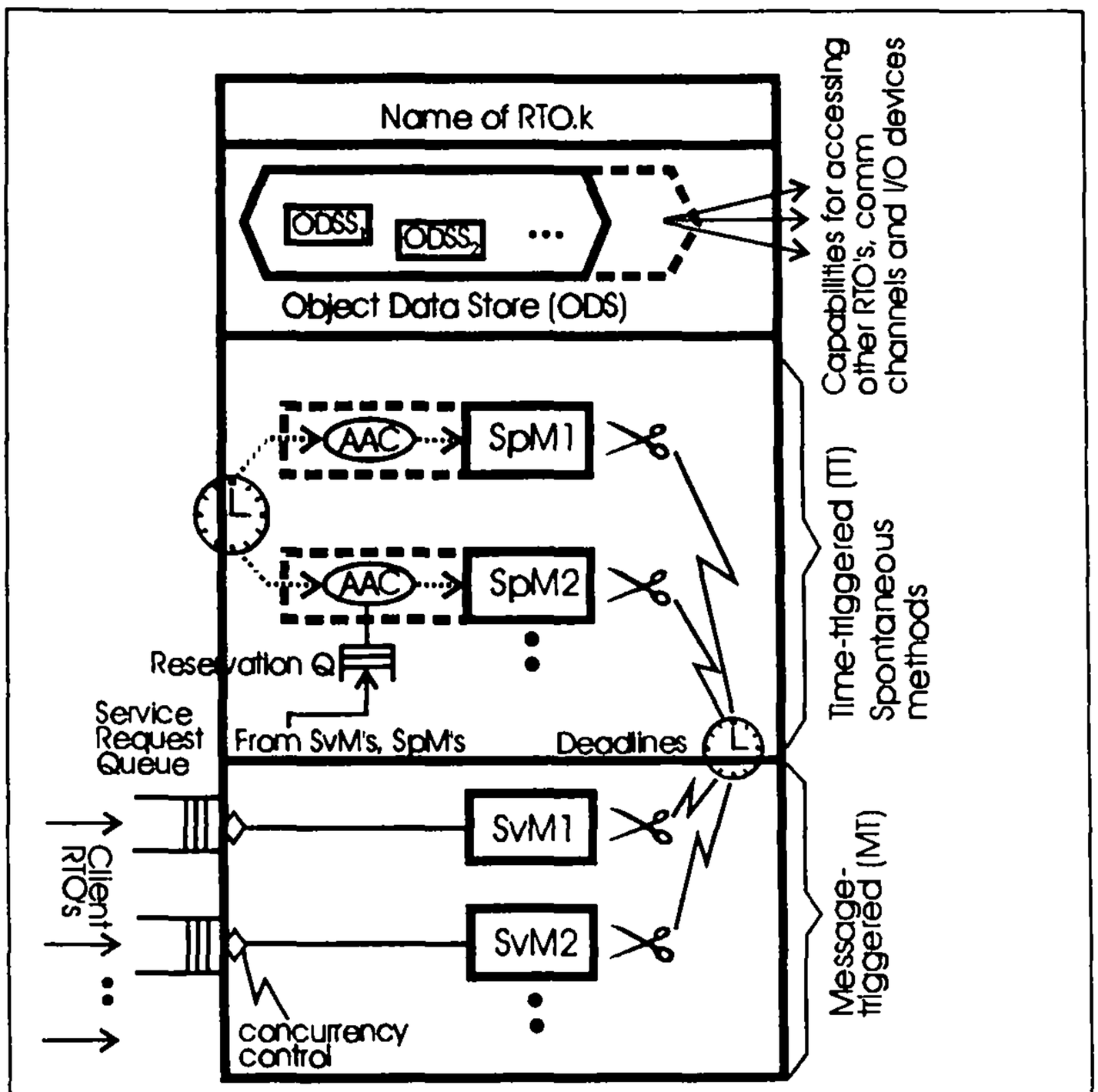


Figure 1. The RTO.k object structure (adapted from [Kim94a])

(c) For each execution of a method of an RTO.k object (to be exact, for each output action to occur during a method execution), a *deadline* is imposed.

The first two features (a) and (b) mentioned above make the RTO.k object structure clearly distinguished from other proposed real-time object structures [Att91, Ish92, Kim94a, Tak92].

The designer of each RTO.k object provides a guarantee of timely service capabilities of the object by indicating the *deadline for every output* produced by each SvM (and each SpM which may be executed on requests from SvM's) in the specification of the SvM (and some relevant SpM's) advertised to the designers of potential client objects. Before determining the deadline specification, the server object designer must convince himself/herself that with the object execution engine (hardware plus operating system) available, the server object can be implemented to always execute the SvM such that the output action is performed within the deadline. Again, the BCC contributes to major reduction of these burdens imposed on the designer.

The RTO.k object structuring scheme is effective not only in the multiple-level abstraction of real-time (computer) control systems under design but also in the accurate representation and simulation of the application environments. This uniform structuring presents considerable potential benefits to the system engineers. An illustration of this aspect is given in the next section.

The relationship between the RTO.k object structuring scheme to the conventional process-oriented structuring of real-time concurrent programs is analogous to that between the high level language programming and the assembly language programming.

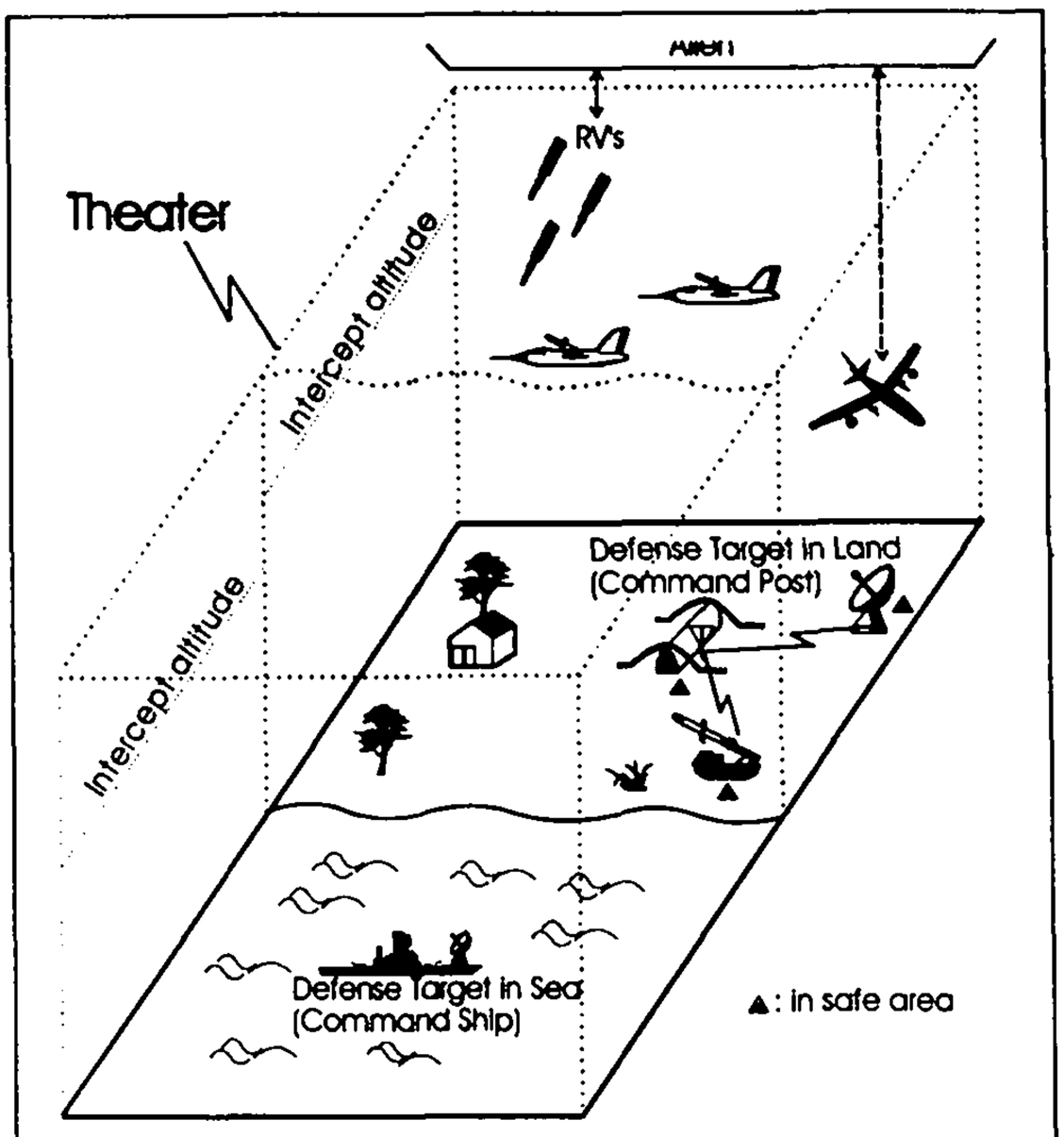


Figure 2 CAMIN application environment: Theater
(Adapted from [Kim96c]

5. RTO.k object based design and simulation: An example

Consider the anti-missile defense scenario depicted in Figure 2. The application environment in this context is a sky+land+sea segment of interest, called the "*theater*", in which moving objects including a *valuable target to be defended* (i.e., command ship in sea) and *flying objects* of both hostile and non-threatening types, appear and move around. The defense system to be constructed is called the *Coordinated Anti-Missile Interceptor Network* (CAMIN) [Kim96c].

Initially, the high-level requirements are given by the customer who places an order for the CAMIN:
(1) Each reentry vehicle (RV) should be intercepted if it is considered dangerous; and
(2) If it is useful in avoiding the dangers posed by RV's, move the defense target (e.g., command ship) around.

5.1 Step 0: High-level Specification of the Application Environment of the CAMIN as an RTO.k Object

Initially, sensors such as radars and actuators such as interceptor launchers (both air-borne and ground-based) do not exist because the system engineer (team) has not decided which types to use. As the first step, the system engineer may describe the application environment of the CAMIN as an RTO.k object depicted in Figure 3 without the components enclosed by square brackets. The RTO.k object is called the *Theater RTO*. The object data store (ODS) of the Theater RTO basically consists of the "*state descriptors*" for a defense target in sea (command ship), a defense target in land (command post), dynamically varying numbers of RV's and *non-threatening flying objects* (NTFO's) (e.g., commercial airplanes and birds), and the space

(=sky+land+sea space) in the theater. The information kept in the Theater RTO is thus a composition of the information kept in all the state descriptors within its ODS.

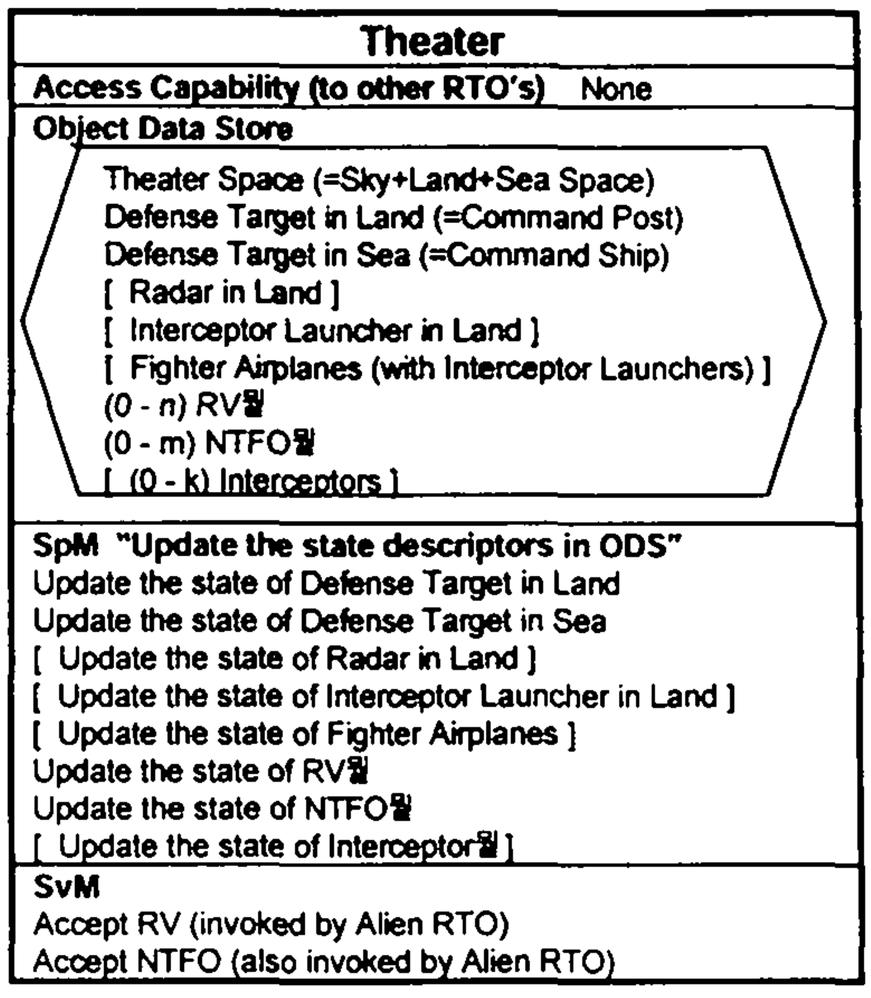


Figure 3 High-level specification of the Theater RTO
(Components enclosed by square brackets are chosen by the engineer)

For each *environment object* represented by a state descriptor in the Theater RTO, there is a spontaneous method (SpM) for periodically updating the state descriptor. Conceptually the SpM's in the Theater RTO are *activated continuously* and each of their executions is *completed instantly*. The SpM's can then represent continuous state changes that occur naturally in the environment objects. The natural parallelism that exists among the environment objects can also be precisely represented by use of multiple SpM's which may be activated simultaneously.

The service methods (SvM's) in the Theater RTO are provided as an interface for the clients outside the theater. The only conceivable clients here are the enemy which sends RV's into the theater and the "natural forces" which cause NTFO's (such as commercial airplanes and birds) to enter the theater. Entry of an RV into the theater is represented by an enemy's call for the SvM "Accept RV". Both the enemy and the external forces are represented by an RTO.k object called the *Alien RTO*.

So far, the Theater RTO in Figure 3 has been interpreted as a mere description of the application environment. However, if the activation frequency of each SpM is chosen such that it can be supported by an object execution engine, then the resulting Theater RTO becomes a *simulation model*. The behavior of the application environment is represented by this simulation model somewhat less accurately than by the earlier description model based on continuous activation of SpM's. In general, the accuracy of an RTO.k object structured simulation is a function of the chosen activation frequencies of SpM's.

5.2 Step 1: High-level design of the application environment simulator based on the RTO.k object model

Upon receiving the customer's order, the system engineer will first decide on the set of sensors and actuators to be deployed in the theater. Figure 2 already includes some sensors, i.e., radars which are located both in land and in the command ship, and some actuators, i.e., interceptor launchers which are located in

land, in the command ship, and in the fighter airplanes. After the set of sensors and actuators is determined, the Theater RTO in Figure 3 is expanded to incorporate all the components enclosed by square brackets. The ODS now contains the selected sensors (e.g., Radar in Land) and actuators (i.e., Interceptor Launcher in Land with Interceptors). The radar and interceptor launcher loaded on the command ship and another interceptor launcher on the fighter airplane are not shown in the ODS of the Theater RTO but these environment objects are described in the corresponding parts of the state descriptors for the command ship and the fighter airplane, respectively.

The Theater Space component in the ODS of the Theater RTO not only provides geographical information about the theater but also maintains the position information of every moving object in the theater. This information is used to determine the occurrences of collisions among objects and to recognize the departure of any object from the theater space to the outside.

In addition to the selection of sensors and actuators, the system engineer should decide on the deployment of the computer-based control system in the theater. The functions of the control computer system will be determined based on the control theory logic adopted. In this experimental development, we deployed two control computer systems; one inside the command post (in land) and the other in the command ship.

5.3 Step 2: Expansion of the Theater RTO into an RTO network

As the system engineer refines the single RTO.k representation of the theater, a component in the ODS of the Theater RTO may be taken out of the Theater RTO and form a new RTO.k object. For example, the command post and the command ship can be separated out of the Theater RTO and become represented by separate RTO.k objects, the Command Post RTO and the Command Ship RTO. When the new RTO.k objects are created, the SvM's that serve as front-end interfaces of those new RTO.k objects and the call links from the earlier born RTO.k objects to the new objects should also be created. As a result, the Theater RTO becomes a network of three RTO.k objects. The two new RTO.k objects may describe or simulate the command post and the command ship more accurately than the Theater RTO in Figure 3 did.

The Command Ship RTO contains a control computer system and so does the Command Post RTO. The two control computer systems can be separated out of the Command Post RTO and the Command Ship RTO, respectively and become represented by two separate RTO.k objects. This makes the theater to be represented by a network of five RTO.k objects. By now, a requirement specification to be given to the computer engineer (team) has become ready. The full specification (in a form similar to Figure 4) of the network of six RTO.k objects including the Alien RTO plus the following statement can form such a requirement specification:

> "Embed one control computer system in the Command Post and another in the Command Ship such that the computer systems control the chosen sensors (radar's) and the chosen interceptor launchers in order (1) to intercept incoming dangerous RV's by using air-borne launchers first, ground-based launchers next, and ship-borne launchers last, and (2) to move the command ship appropriately to further reduce the danger."

5.4 Step 3: Detailed design of the control computer system as an RTO.k object network

Let us now consider only the control computer system to be housed in the command post in land. Initially, the computer engineer starts with a single RTO.k object structuring of an abstract design of the control computer system. The ODS of the RTO.k object contains several major data structures such as *Radar Data Queue* (RDQ). In the next step, the single RTO.k design is decomposed into multiple RTO.k objects, each of which is centered around a major data structure contained in the ODS of the previous single RTO.k design. Figure 4 shows a partially detailed design specification of the RDQ RTO. Further details on this application scenario are referred to [Kim96d, Kim97c].

Radar_Data_Queue
Access Capability (to other RTO's) - Flying Object Tracking Information (Receive_new_info_from_Radar_Data_Queue)

Object Data Store

Scan_Search_Result: MVD: x msec : list of {current_position, current_time}
Spot_Check_Predicted: MVD: y msec : list of {object_ID, predicted_position, predicted_time }
Spot_Check_Result: MVD: z msec : list of {current_position, current_time, predicted_time}

SpM

SpM1 *Send_New_Data_To_Flying_Object_Tracking_Information*

- For each spot-check result in Data Set [Spot_Check_Result], match with predicted status data in □ .
  - Send the message to Flying_Object_Tracking_Information (via SvM request).
- For each scan search results in Data Set [Scan_Search_Result], construct a message □ .
  - Send the message to Flying_Object_Tracking_Information (via SvM request).

AAC: **for** T = **from** **RTO_START** + WARMUP_DELAY_SECS **to** **RTO_START** + SYSTEM_LIFE_HOURS
every PERIOD **start-during** (T, T + START_WINDOW) **finish-by** T + DEADLINE

OutputSpec: <deadline: xxx msec> list of (object_ID, position, time)

SvM

SvM 1 **< Accept-via-Service_Request_Channel-with-Delay_Bound-of** ACCEPTANCE_DEADLINE
under MAX_REQUEST_RATE ***finish-within*** EXECUTION_TIME_LIMIT>
Receive_From_Radar_In_Land (pos_list)
- Receive from Radar_In_Land the information on all recent object detections.
- Update Data Set [Scan_Search_Result] or [Spot_Check_Result] according to the value of detection_type.

InitiationCond: Other SvM1 invocations are not in place.
InputSpec: pos_list = array of (return_type (=scan_search/spot_check), position, time, predicted_time)

SvM 2 **< Accept-via-** ..... **>**
Receive_From_Flying_Object_Tracking_Information (spot_list)
...

Figure 4 A partially detailed design specification of the RDQ RTO

Figure 5 depicts the fully decomposed network of RTO.k objects in the CAMIN. The right-hand side of the network depicts the real-time simulator of the application environment, which consists of Radar RTO, Fighter-Airplane RTO, etc., whereas the left-hand side depicts the designs of the control computer systems. A prototype implementation (version 3.0) of the timeliness-guaranteed OS kernel model called the DREAM kernel [Kim95b], was used in the experimental implementation of the CAMIN and its application environment simulator. This real-time distributed application software consists of nine different types of tailorable RTO.k objects and runs on a network of three PC's. In this experimental effort, the RTO.k objects were implemented in C++ with the support of a library named the DREAM library (version 3.0b) [Kim96a, Kim97d] which serves as a friendly API of the DREAM kernel implementation.

6. Advantages of the RTO.k object based design and simulation

The RTO.k object based uniform structuring approach brings the following major advantages which have been observed in our multiple experiments conducted so far:

(1) Strong traceability between requirement specification and design.

(2) Cost-effective high-coverage validation

(3) Autonomous subsystems and ease of maintenance

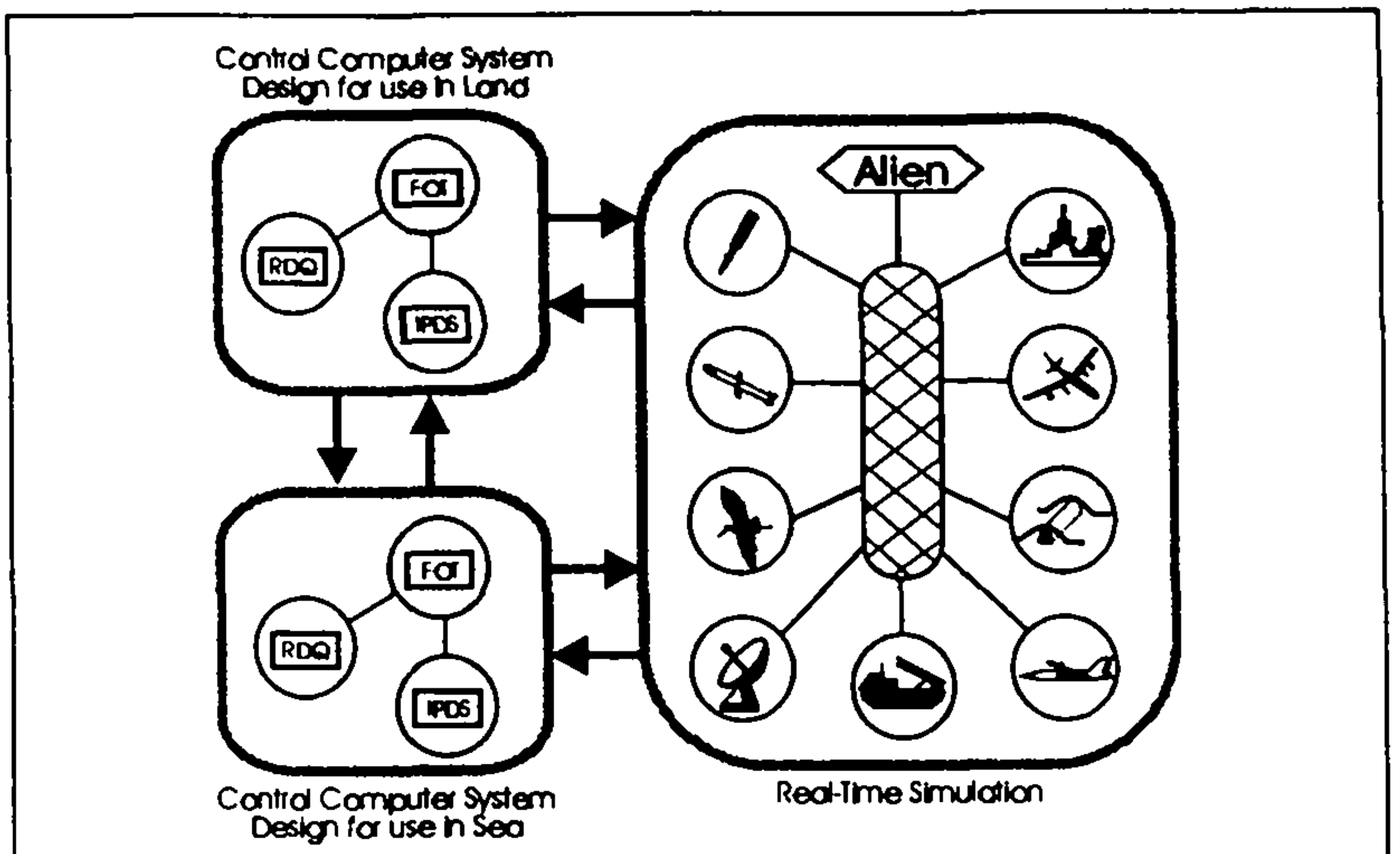


Figure 5. Network of RTO뭡 in CAMIN
(Adapted from [Kim96c])

Prior to the development of the RTO.k object structuring scheme, multiple experimental implementations of defense applications similar to the CAMIN had been conducted in the author's laboratory. Those early implementations were done by using conventional process structured design approaches without rigorous specification and analysis of timing aspects. Our experiences indicate that the RTO.k object structured design approach yields at least several times improvement, if not an order-of-magnitude improvement, in design productivity over the conventional process structured design approaches.

7. Conclusion

This author believes that the real-time object structuring technology and the real-time simulation technology are of vital importance in facilitating highly reliable engineering of large-scale RTCS's such as defense command-control systems, which the society has started demanding. These technologies will bring not only improved product reliability but also improved development economy. Considering that real-time application markets in the non-military world are about to explode due to the improved reliability of multimedia handling technologies, the needs of the real-time object structuring technology and the real-time simulation technology are as accute in the general data processing area as in the defense area. Therefore, although these technologies have not yet been established in mature forms, it is safe to bet that their advancement will be very rapid from now on.

As a sample of what is coming in the area of the real-time object structuring technology, the RTO.k object technology was briefly presented in this paper. The uniform structuring of both RTCS's and real-time simulators of their application environments that is facilitated by the RTO.k object scheme brings in significant improvements in various parts of the system engineering process, which in turn results in significantly improved design productivity. Although limited in scope, the RTCS development experiments conducted so far are strongly supportive of this positive assessment. However, to fully realize the potential, many new specification, design, and execution tools need to be developed.

Acknowledgment: The research work reported here was supported in part by US Navy, NSWC Dahlgren Division under Contract No. N60921-92-C-0204, in part by the University of California MICRO Program

under Grant No. 96-169, in part by the California Transportation Department via the UCI Institute for Transportation Studies, in part by Hitachi, Ltd, in part by ETRI, in part by Postech, and in part by LG Electronics.

References

[Att91] Attoui, A. and Schneider, M., "An Object Oriented Model for Parallel and Reactive Systems", *Proc. IEEE CS 12th Real-Time Systems Symp.*, 1991, pp. 84-93.

[Boo91] Booch, G., '*Object-Oriented Design*', Benjamin Cummings, CA, 1991.

[Dah72] Dahl, O.J., "Hierarchical Program Structuring", in Dahl, Dijkstra, & Hoare eds., '*Structured Programming*', Aca. Press, NY, 1972.

[Ell90] Ellis, M.A. and Stroustrup, B., '*The Annotated C++ Reference Manual*', Addison-Wesley Pub. Co., Reading, MA, 1990.

[Ish92] Ishikawa, Y., Tokuda, H., and Mercer, C. W. An Object-Oriented Real-Time Programming Language, *IEEE Computer* (Oct. 1992), 66-73.

[Kim94a] Kim, K.H. et al., "Distinguishing Features and Potential Roles of the RTO.k Object Model", *Proc. 1994 IEEE CS Workshop on Object-oriented Real-time Dependable Systems (WORDS)*, Oct. '94, Dana Point, pp.36-45.

[Kim94b] Kim, K.H. and Kopetz, H., "A Real-Time Object Model RTO.k and an Experimental Investigation of Its Potentials", *Proc. 1994 IEEE CS Computer Software and Applications Conf. (COMPSAC)*, Nov. 1994, Taipei, pp.392-402.

[Kim95a] Kim, K.H., "Toward New-Generation Real-Time Object-Oriented Computing", *Proc. IEEE CS 5th Workshop on Future Trends of Distributed Computing Systems (FTDCS)*, Cheju Island, Aug. '95, pp.520-529.

[Kim95b] Kim, K.H. et al., "A Timeliness-Guaranteed Kernel Model - DREAM Kernel and Implementation Techniques", *Proc. 1995 Int'l Workshop on Real-Time Computing Systems and Applications (RTCSA 95)*, Tokyo, Japan, Oct. 1995, pp.80-87.

[Kim96a] Kim, K.H. et al., "The DREAM Library Support for PCD and RTO.k programming in C++", *Proc. 1996 IEEE CS Workshop on Object-oriented Real-time Dependable Systems (WORDS)*, Feb. 96, Laguna Beach, pp. 59-68.

[Kim96b] Kim, K.H., Nguyen, C., and Park, C., "Real-Time Simulation Techniques Based on the RTO.k Object Modeling", *Proc. COMPSAC '96 (IEEE CS Software & Applications Conf.)*, Seoul, August 1996, pp.176-183.

[Kim96c] Kim, K.H., Kim, Y.S., and Kim, H.J., "An RTO.k Object Based Uniform Integrated Design of Real-Time Computing Systems and their Application Environment Simulators", *Proc. 2nd World Conf. on Integrated Design and Process Technology*, IDPT-Vol.2, Austin, TX, Dec. 1996, pp.106-113.

[Kim97a] Kim, K.H. and Subbaraman, C., "Fault-Tolerant Real-Time Objects", *Communications of the ACM*, January 1997, pp. 75-82.

[Kim97b] Kim, K.H., "Toward New-Generation Object-Oriented Real-Time Software and System Engineering" Invited paper, *SERI Journal*, Taejon, Korea, Vol.1, No.1, Jan. 1977, pp.1-23.

[Kim97c] Kim, K.H., "Uniform Object Structuring of Complex Systems and Environment Simulators", to appear in *Computer* (The IEEE CS Magazine), 1997.

[Kim97d] Kim, K.H., Subbara, C., and Bacellar, L., "Support for RTO.K Object Structured Programming in C++", to appear in *Control Engineering Practice* (an IFAC Journal), 1997.

[Kop90] Kopetz, H. and Kim, K.H., 1990, "Temporal Uncertainties in Interactions among Real-Time Objects", Proc. IEEE CS 9th Symp. on Reliable Distributed Systems, Huntsville, AL, Oct. 1990, pp.165-174.

[Rum91] Rumbaugh, J. et al., '*Object-Oriented Modeling and Design*', Prentice Hall, NJ, 1991.

[Sel94] Selic, B., Gullekson, G., and Ward, P.T., '*Real-Time Object-Oriented Modeling*', John Wiley & Sons, New York, 1994.

[Tak92] Takashio, K., and Tokoro, M., "DROL: An Object-Oriented Programming Language for Distributed Real-Time Systems", *Proc. OOPSLA*, 1992, pp. 276-294.

[Wor94] '*Proc. 1994 IEEE CS Workshop on Object-oriented Real-time Dependable Systems (WORDS), Oct. '94, Dana Point*', IEEE CS Press, CA, 1995.

[Wor96] '*Proc. 1996 IEEE CS Workshop on Object-oriented Real-time Dependable Systems (WORDS), Feb. '96, Laguna Beach*', IEEE CS Press, CA, 1996.

[Wor97] '*Proc. 1997 IEEE CS Workshop on Object-oriented Real-time Dependable Systems (WORDS), Feb. '97, Newport Beach*', to appear in June 1997, IEEE CS Press, CA.

SUPPORT FOR RTO.k OBJECT STRUCTURED PROGRAMMING IN C++

K. H. (Kane) Kim and Chittur Subbaraman

Dept. of Electrical and Computer Engineering
University of California, Irvine, U.S.A.
kane@ece.uci.edu

Luiz Bacellar

United Technologies Research Center
Hartford, Connecticut, USA
bacellar@utrc.utc.com

Abstract: In the past few years the authors have established a real-time object-oriented (OO) structuring approach called the RTO.k object structuring scheme for the purpose of realizing real-time computing in the form of a generalization of non-real-time computing, that yet allows system engineers to confidently produce certifiable real-time systems for safety-critical applications. Also, a model of an operating system kernel named the DREAM kernel (which can support both conventional real-time processes and RTO.k objects with guaranteed timely services), has been formulated. A user-friendly interface to DREAM kernel services for RTO.k-structured real-time application programs is provided by a collection of specific C++ classes called the DREAM library. In a sense, C++ augmented with the DREAM library can be viewed as an RTO.k object programming language, although it does more than that. This paper discusses the essence of RTO.k object programming in C++ with the aid of the DREAM library.

Keywords: Real-time systems, programming support, object-oriented programming, operating systems, software tools.

1. INTRODUCTION

The necessity for a new methodology for real-time system structuring that eases the job of managing the system complexity and producing reliable real-time systems is becoming more acute than ever. The time is now ripe for pursuing the following paradigms, which may be collectively called the General-form Timeliness-guaranteed (GT) design, see (Kim, 1995b).

(1) *General-form design style.* Future real-time computing must be realized in the form of a generalization of the non-real-time computing, rather than in a form that looks like an esoteric specialization.

(2) *Design-time guarantee of timely service capabilities of subsystems.* To meet the demands of the general public for the assured reliability of future real-time computing systems (RTCSs) in safety-critical applications, it is inevitable that system engineers will be required to produce design-time guarantees for the timely service capabilities of various subsystems (which will take the form of objects in object-oriented system designs).

With the purpose of facilitating GT design, a real-time object-oriented structuring approach has been established recently by the first co-author, together with his collaborating researchers. The new object-structuring approach, which has several unique features, is called the RTO.k object structuring scheme, see (Kim, 1994). The RTO.k object structuring scheme differs from other object-oriented structuring approaches which support real-

time computer system (RTCS) design, in that it is aimed at supporting the design-time guarantee of timely service capabilities of objects.

A model of an operating-system kernel which can support real-time processes with guaranteed timely services was formulated by the first co-author, see (Kim, 1995). The model is called the DREAM (Distributed Real-time Ever Available Micro-computing) kernel. The DREAM kernel can support RTO.k object-structured application software as well as conventional process-structured real-time application software. The DREAM kernel was formulated to accomplish the key property of providing guaranteed timely services to real-time applications with minimal loss of hardware utilization. Several prototype implementations of the DREAM kernel have been produced by the authors and their research collaborators, to run on networks of PCs connected by Ethernet. A prototype kernel (v.D3) has been used to run an RTO.k structured non-trivial defense C^3 application, together with an RTO.k structured real-time simulator of the application environment.

A user-friendly interface to the DREAM kernel can reduce the burden imposed on the RTO.k object implementers. To achieve this goal, the authors have designed a library, called the DREAM Library, which is a collection of several C++ classes. Each class functions as an interface between a component of an RTO.k object program and the kernel support for that component. The DREAM library hides various details of parameter passing between application and the DREAM kernel from the application programmer.

The paper starts in Section 2 with a discussion on the essence of the RTO.k object-structuring scheme. Section 3 discusses the major features of the DREAM kernel offered to real-time application programmers. Section 4 presents the DREAM library support for RTO.k structured application programming. This paper concludes in Section 5.

2. OVERVIEW OF THE RTO.k OBJECT STRUCTURING SCHEME

Several years ago, Kopetz and Kim (1990) proposed an abstract precursor of the *RTO.k object model* for cost-effective development of *hard-real-time* systems. In recent years, the RTO.k object, also called the time-triggered real-time object (TT-RTO), has been formalized to possess a concrete syntactic structure and execution semantics (Kim, 1994a; Kim, 1994b).

The basic structure of the RTO.k object model is shown in Figure 1. It is an extension of the conventional object model(s) in four major ways:

(a) *Clear separation between two types of methods.*

The RTO.k object model contains two types of methods, time-triggered (TT-) methods, (also called the spontaneous methods or SpMs) which are clearly separated from the conventional service methods (SvMs). The SpM executions are triggered when the real-time clock reaches specific values determined at

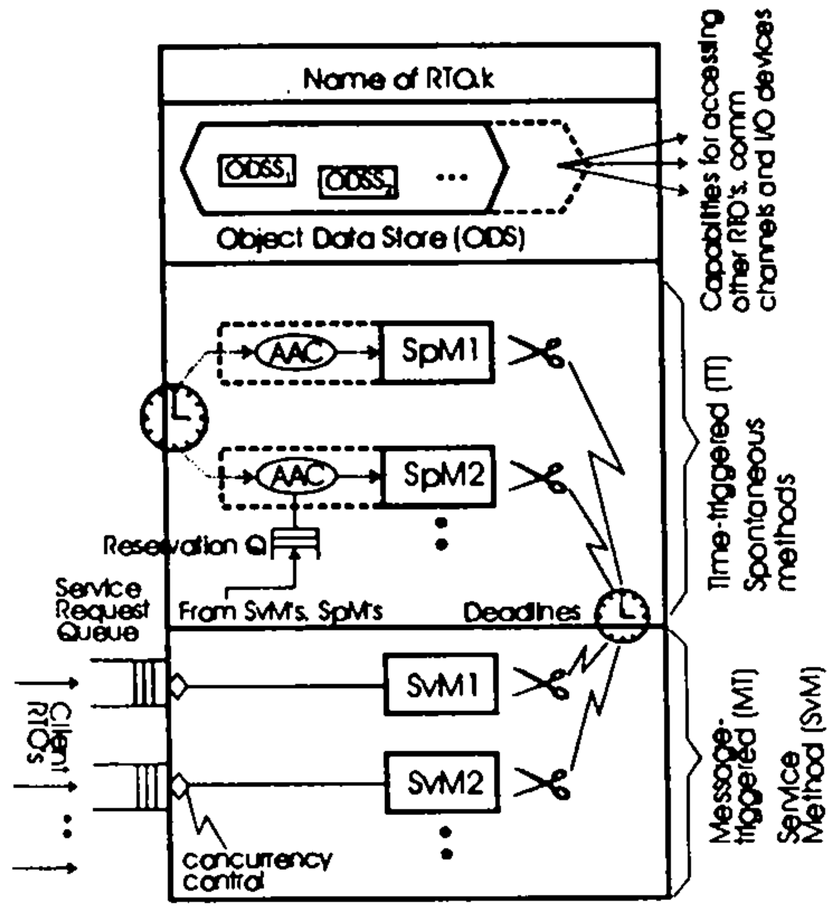


Fig. 1. Structure of the RTO.k Object Model (Adapted from (Kim, 1994a))

design time, whereas the SvM executions are triggered by service request messages from clients. Moreover, actions to be taken at real times *which can be determined at the design time* can appear only in SpMs.

(b) *Basic concurrency constraint (BCC).*

This rule prevents potential conflicts between SpMs and SvMs and reduces the designer's efforts in guaranteeing timely service capabilities of RTO.k objects. Basically, *activation of an SvM triggered by a message from an external client is allowed only when no potentially conflicting SpM executions are in place.* An SvM is allowed to execute only if no SpM that accesses the same part of the object data store (ODS) to be accessed by this SvM has an execution time window that will overlap with the execution time window of this SvM. Thus, SpM executions are given higher priority over SvM executions. However, the BCC does not stand in the way of either concurrent SpM executions or concurrent SvM executions.

(c) A deadline is imposed for each execution of a method of an RTO.k object;

(d) After an interval called the maximum validity duration passes, real-time data contained in the ODS of an RTO.k object become invalid.

Extensions (a) and (b) are unique to the RTO.k object model in comparison with other object models, see (Attoui, 1991; Bihari, *et al.*, 1989; Ishikawa, *et al.* 1992; Shrivastava and Waterworth, 1991; Takashio and Tokoro, 1992). In an RTO.k object, both types of methods, the SpMs and the SvMs, perform operations on the RTO.k object data store (defined in the ODS section). The triggering times for SpMs should be defined by the application designer as part of an SpM specification. The definition of these triggering times should be placed in the autonomous activation condition (AAC) section of an SpM. An example of an AAC is

"for t = from 10am to 10:50am every 30min start-during (t, t+5min) finish-by t+10min"

which has the same effect as

{"start-during (10am, 10:05am) finish-by 10:10am□
"start-during (10:30am, 10:35am) finish-by 10:40am□.

Instead of specifying actual triggering times in the AAC section of an SpM, the designer may specify a set of candidate triggering times by starting the AAC with the declaration "if-demanded". A subset of these candidate triggering times may be chosen dynamically for actual triggering when an SvM within the same RTO.k object requests future executions of this specific SpM.

The authors consider that any RTCS can be constructed as a network of RTO.k objects. Client methods (SpMs or SvMs) request services from SvMs in other RTO.k objects. To maximize the concurrent executions of client and server methods, client methods are allowed to make non-blocking types of service requests to SvMs.

The designer of each RTO.k object indicates the *deadline for every output* produced by each SvM (and each SpM which may be executed on requests from SvMs) in the specification of the SvM (and some relevant SpMs) and advertises this to the designers of potential client objects. The designer of the server object thus guarantees the timely services of the object. Before determining the deadline specification, the server object designer must convince himself/herself that with the object execution engine (hardware plus operating system) available, the server object can be implemented to always execute the SvM such that the output action is performed within the deadline. Again, the BCC contributes to a major reduction of these burdens imposed on the designer.

The RTO.k object model is effective not only in the multiple-level abstraction of real-time (computer) control systems under design, but also in the accurate representation and simulation of the application environments. In fact, it enables uniform structuring of control computer systems and application environment simulators, see (Kim, 1994b; Kim, 1996), and this presents considerable potential benefits to the system engineers.

3. MAJOR FEATURES OF THE DREAM KERNEL

In order for the RTO.k objects to provide guaranteed timely services to external clients, the engine that supports the execution of RTO.k objects must obviously provide guaranteed timely services to the requesting RTO.k objects. In addition to containing a hardware platform, such an execution engine should be formed by a *timeliness-guaranteed operating system*. Existing commercial operating systems are not yet capable of providing the guaranteed timely services needed by a great deal of real-time application software.

The DREAM kernel has been formulated as a model of an operating-system kernel that can support guaranteed timely services, not only for RTO.k object-structured real-time applications, but also for conventional process-structured real-time applications. An experimental prototype of the DREAM kernel has been implemented in the DREAM laboratory of University of California, Irvine. The experimental version runs on a network of PCs connected via Ethernet and equipped with i486 or Pentium processors, DOS-BIOS device drivers, and the Packet Ethernet driver. Several variations of the DREAM kernel are under development in other cooperating organizations. This section briefly discusses some major features of the DREAM kernel facility offered to C++ RTO.k program designers.

3.1 *Guaranteed worst-case response*

One of the most distinguishing features of the DREAM kernel is that it provides guaranteed timely services to real-time applications with minimal loss of hardware utilization.

The DREAM kernel guarantees the timely response to service requests by adopting a unique approach for the layering of its components, see (Kim, 1995b). This approach is based on the organizational principle called time-leasing machine layering, which is of fundamental nature. The kernel consists of five layers L0 through L4 in total. Under the time-leasing machine-layering principle, the bottom layer (L0 in the DREAM kernel) owns the full power of the hardware machine. So, the bottom layer uses the hardware machine at will. The remainder of the hardware machine time, after the bottom-layer use of the hardware machine, is "leased" to the next upper layer (L1 in the DREAM kernel). The time-leasing relationship is recursive, i.e., it is maintained through all the layers that compose the kernel (in the DREAM kernel layers, L0-L4). This means that

> *when a lower layer is using the hardware machine, it cannot be disturbed by the upper layer.*

Moreover, there is a tight bound on the amount of machine time that each layer uses. This means

> *to enforce that the amount of hardware machine time which is used by any layer during a basic response period be limited within a specific threshold.*

The *basic response period* here refers to the maximum interval between the instant at which a process calls for a kernel service and the instant at which the service is completed. The obvious purpose of adopting this principle is to guarantee a certain amount of machine time being available to each of the upper layers. This implies in the cases of some layers a bound on the frequency of certain types of interrupts that are accepted.

The five layers in the DREAM kernel contain the following components:

(1) L0 contains a manager of a real-time clock which supplies the current time upon receiving

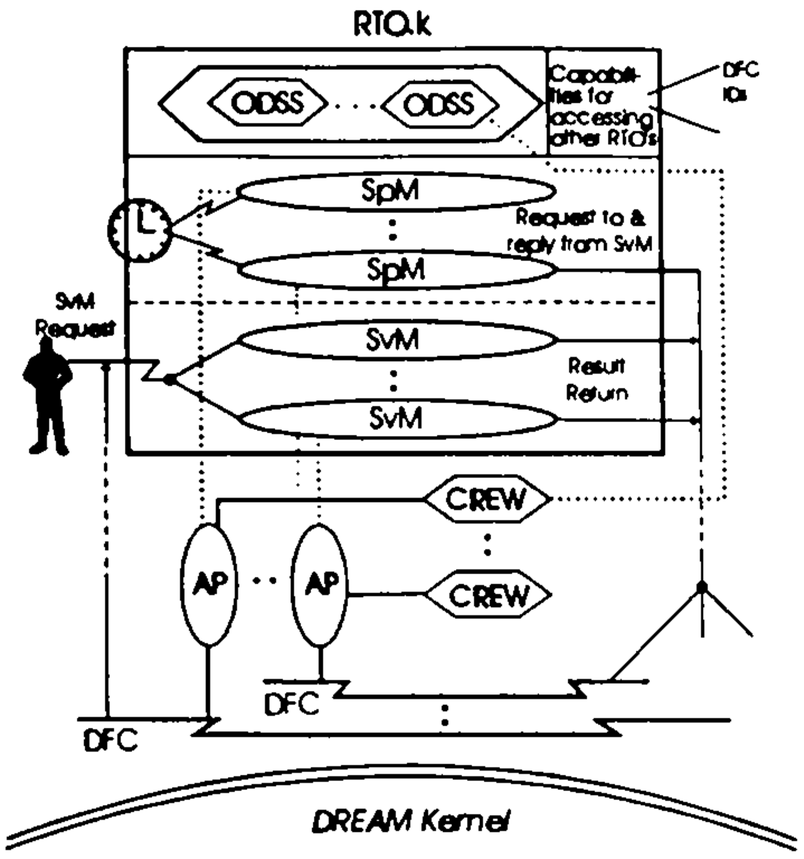


Fig. 2. Mapping of an RTO.k object to a conventional process-structured program

a request and also provides the "wake-up call" service;

(2) L1 contains basic support functions for high-bandwidth I/O such as the local area network (LAN) interface;

(3) L2 contains the kernel-thread scheduler which is activated upon expiration of a thread-time-slice or upon donation of an unfinished thread-time-slice by its current owner; (kernel-threads themselves to be discussed in Section 3.2 are components in L4.)

(4) L3 contains basic support functions for low-bandwidth I/O such as serial character I/O;

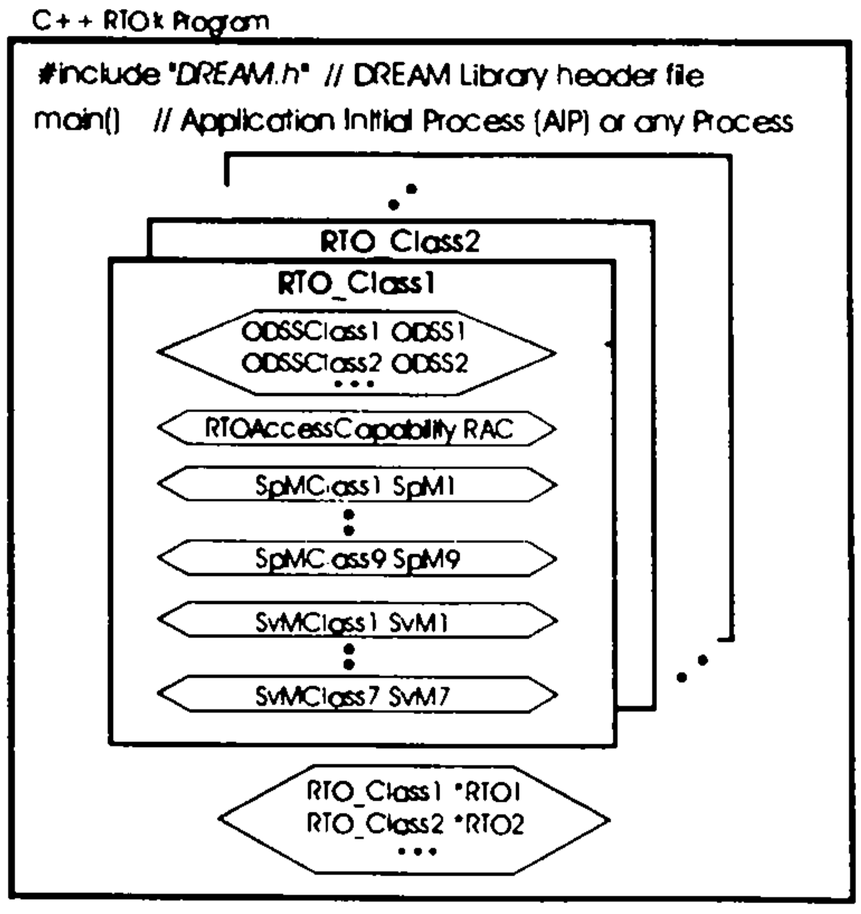


Figure 3. The definition of RTO class

(5) L4 contains the process scheduler and other support functions for processes, and inter-process communication. The process scheduler is activated upon expiration of a *process-time-slice* as well as at some other times. Here the size of the process-time-slice is usually an integ-ral multiple of the size of the thread-time-slice.

3.2 *Exploitation of parallelism within the kernel by use of kernel-threads*

In order to maintain a high degree of hardware utilization, parallelism is exploited extensively within the DREAM kernel through the introduction of kernel-threads. The kernel-thread, or thread for short, is an active concurrency unit operating inside the DREAM kernel. The set of threads is fixed at the operating system loading time. All the threads share the same address space and they are strictly *periodic threads* which make the analysis of their worst-case execution behavior a trivial task. Currently, four basic kernel-threads are used in the DREAM kernel. Additional application-specific custom kernel-threads can be introduced if fast response activities specific to the application must be supported.

3.3 *Support for execution of RTO.k objects*

For supporting the components of RTO.k structured application programs, the DREAM kernel utilizes the same support provided to the components of conventional process-structured real-time application programs. In other words, the DREAM kernel uses the support provided for

processes,

concurrent-read-&-exclusive-write (CREW) monitors (shared data structure monitors for intra-node inter-process communication), and

data field channels (DFCs), see (Mori, 1993; Kim, 1995a), (which are real-time logical multicast channels for both intra-node and inter-node inter-process-group message communication to execute the components of an RTO.k-structured application program).

The DREAM kernel thus executes RTO.k object components as follows (as shown in Figure 2):

(1) ODS segments (ODSSs) as CREW monitors,
(2) Both SpMs and SvMs as application-specific program bodies of processes (called *method execution processes* or MEPs),
(3) Paths for accessing SvMs in the form of DFCs monitored by the SvMs,
(4) Result-return paths to clients in the form of DFCs monitored by the clients, and
(5) capabilities for accessing SvMs in other RTOs in the form of the IDs of the DFCs monitored by the SvMs.

4. DREAM LIBRARY SUPPORT FOR RTO.k STRUCTURED PROGRAMMING IN C++

The DREAM library greatly reduces the burden of the C++ RTO.k application programmer in the

construction of RTO.k objects, and in designing kernel service calls (KSCs) to the DREAM kernel. The C++ RTO.k application programmer typically defines an *RTOClass* inside the *application initial process* (AIP). The AIP is the starting point of the entire application in a node. Figure 4 shows such RTO.k classes defined as C++ classes. In this figure, the *RTO_Class1* represents one RTO.k class. It consists of

(1) an instance of *RTOAccessCapabilityClass* (representing access rights for SvMs in other RTOs),
(2) instances of each *ODSSClass* (representing the ODS segments in this RTO),
(3) instances of each *SpMClass* (representing the SpMs of this RTO), and
(4) instances of each *SvMClass* (representing the SvMs of this RTO).

In order to ease the creation of these user-defined classes, the DREAM library provides a set of pre-defined classes (templates) which can be inherited by the user-defined classes. The set of pre-defined classes includes

(1) the *BasicRTOAccessCapabilityClass*,
(2) the *BasicODSSClass* for constructing an ODSS,
(3) the *BasicSpMClass*, for creating an SpM, and
(4) the *BasicSvMClass*, for creating an SvM.

Thus, as will be shown in Section 4.2, a user-defined *RTOAccessCapabilityClass* can easily be created by inheriting the *BasicRTOAccessCapabilityClass* from the DREAM library, and then adding a certain number of application-specific program components. Other user-defined classes can also be created in a similar manner.

Subsequent subsections will describe in some detail the construction of the individual components of an RTO.k object using the DREAM library. First, consider the manner in which the DREAM kernel executes SpMs and SvMs.

4.1 *Execution of SvMs and SpMs as MEPs*

An RTO.k object method should be assigned to a method execution process (MEP) before the method can be executed. To give maximum flexibility to the application programmer, the DREAM kernel offers different types of method-to-MEP assignments. These include:

(1) *Exclusive binding of a method to an MEP.* In this case, a long-term one-to-one connection is established between a method and an MEP by the programmer. The exclusively bound method cannot be executed by any other MEP, and the MEP cannot be used to execute any other method.

(2) *Shared binding of multiple methods to the same MEP.* In this case, multiple methods are assigned by the programmer to the same MEP for a long duration. Such an MEP can execute only the set of methods that are bound to it. One important restriction for sharing an MEP among multiple methods is that these methods *cannot be active concurrently.* This requirement has been imposed in order to reduce the complexity of implementation. The RTO.k designer who wishes to use the shared binding facility should ensure that the methods that share the same MEP do not have timing requirements that necessitate their concurrent execution.

(3) *Unbound methods.* In this case, the methods are not bound to any MEP by the programmer. When it is time to execute the method, the DREAM kernel automatically finds an unbound free MEP (i.e., an MEP that is neither bound to an object method nor executing another unbound method at the time) and makes an one-shot assignment of the MEP to the method. Unbound methods may thus execute on different MEPs during different execution runs.

The *RTOExecManClass* of the DREAM library offers functions for creating both *generic MEPs* (for which standard parameters such as stack size, priority, etc. are used) and *custom MEPs* (for which user-specified parameters are used), permanently binding a method to an MEP, sharing an MEP among several methods, etc. The C++ RTO.k programmer first creates an instance of the *RTOExecManClass* inside the AIP. Then the programmer may create as many MEPs as necessary and may bind methods to MEPs.

An SpM can be exclusively bound to an MEP, can share an MEP with other SpMs, or can be left unbound. However, an SvM is always left unbound. One reason for leaving an SvM unbound is that it may be executed *in a pipelined fashion*, i.e. multiple invocations of the same SvM may progress concurrently to effect fast responses to service requests. The number of invocations of the same SvM that are active concurrently is of course determined dynamically at execution time. Therefore, to allow a binding of an SvM to one particular MEP (at initialization time) requires a highly complicated execution engine. On the other hand, an SpM does not have a pipelined execution mode and thus a binding between an SpM and an MEP does not require a complicated execution engine.

The shared binding of multiple object methods to the same MEP enables a large number of object methods to be executed on the same local node which supports a moderate number of processes and hence results in increased resource utilization. However, all such methods must have non-overlapping timing constraints. It is up to the designer to ensure this.

4.2 *Construction of RTO access capabilities*

The access capability section in an RTO.k object determines access paths to the SvMs of other RTO objects. The access capability is represented in the form of the DFC ID, that should be used by a client RTO.k object (method) to request execution of the corresponding SvM. The SvM can be viewed as the owner of such a *service-request DFC*, and constantly monitors the DFC. In the approach adopted by the DREAM kernel, the creator of a

server RTO.k should inform all potential clients about the creation of certain services by broadcasting the service-request DFC IDs of the SvMs of the server RTO.k via a special *global broadcast DFC*. This approach can be implemented efficiently in Ethernet-type local area networks, since by broadcasting a single message the creator of an SvM can inform all potential clients about the service-request path. At a client site, the client RTO.k makes a KSC during initialization, to obtain the DFC ID of the external server method.

The C++ RTO.k application programmer can use a DREAM library class called the *BasicRTOAccess-CapabilityClass* for obtaining access capabilities for server RTO.ks. The programmer should design an *RTOAccessCapabilityClass* by inheriting this *Basic-RTOAccessCapabilityClass*. The *BasicRTOAccess-CapabilityClass* provides a friendly interface to the KSC *RegisterRTOAccessCapability*(). This KSC is made with the symbolic names of the server RTO.k and the corresponding SvM as parameters and, if successful, it returns the DFC ID of the SvM. It is a good practice to define inside the *RTOAccess-CapabilityClass* the parameters for each service request to be sent to the relevant server RTO from the host RTO. The following illustrates the construction of *RTO-ccessCapabilityClass*:

```
class RTOAccessCapabilityClass:
    public BasicRTOAccessCapabilityClass
{
  public:
      // parameter for access to SvM3 of RTO2
    struct ParType_RTO2_SvM3 {
        ...;
    } Par_RTO2_SvM3;
      // Service-request DFC ID of SvM3 of RTO2
    int DFCID_RTO2_SvM3;
      // constructor
    RTOAccessCapabilityClass();
};

RTOAccessCapabilityClass::
    RTOAccessCapabilityClass()
{
  RegisterRTOAccessCapability ("RTO2", "SvM3",
      &DFCID_RTO2_SvM3);
}

    // Instantiation of the RTOAccessCapability class
RTOAccessCapabilityClass RAC;
```

4.3 *Construction of ODSS*

The DREAM kernel treats the Object Data Store Segment (ODSS) of an RTO.k object as a special CREW monitor. A mechanism for concurrency control is necessary so as to maintain consistency of an ODSS which is typically accessed by multiple methods of the host RTO.k. The CREW monitor facilitates flexible and yet safe concurrency control.

In order to construct an ODSS, the C++ RTO.k programmer can start designing the ODSSClass by inheriting the *BasicODSSClass* of the DREAM library. This *BasicODSSClass* in turn inherits the *CREWMgtClass* of the DREAM library. The *BasicODSSClass* provides a friendly interface to kernel services such as: creating and destroying ODSS (CREW) support and obtaining and releasing ODSS access rights. The following sample program shows the use of the ODSS class:

```
class ODSSClass:public BasicODSSClass {
    int x,□   // private data area
    public:
    void Method1(); // a member function
};
void ODSSClass::Method1() {
     // Obtaining RW access rights to the CREW
     // monitor by calling the member function of
the
     // CREWMgtClass of the DREAM library
   EnterCREW_RW();
   □   // access the data area
     // Releasing RW access rights to the CREW
       // monitor
   ExitCREW_RW();
}

    // Instantiation of the ODSS class
ODSSClass ODSS1;
```

4.4 *Construction of SpMs*

The C++ RTO.k programmer can start designing the *SpMClass* by inheriting the *BasicSpMClass* of the DREAM library. Figure 4 shows the construction of an *SpMClass*. The functions provided by the *BasicSpMClass* include: registering the SpM with the kernel, making service requests to SvMs and obtaining the return-results, informing the kernel about the completion of one execution run, etc. The SpM body is defined as a member function of the *SpMClass*.

An SpM may make one of the following types of service requests to an SvM:

(1) *blocking* service requests, in which the client (SpM) is blocked after making a service request until either the result is obtained from the SvM or a pre-specified time-out occurs.
(2) *non-blocking* service requests, in which the client proceeds after making a service request and retrieves the result of the request later during its execution.

In order for the SpM to make the above calls, the owner RTO should already have obtained the service-request DFCID of the SvM, as mentioned in Section 4.1. The SpM should use this DFCID as an input parameter in the aforementioned service calls. Also, when an SpM is registered, the kernel assigns a unique result-return DFCID to the SpM, and enables the SpM to receive return-results via this DFC. This result-return DFCID is passed on by the DREAM kernel automatically, along with the service request, so that the SvM may use this DFCID while returning its processed results. This DFCID is hidden from the RTO.k application programmer.

After registering the SpM with the kernel, the user may create an MEP and bind the SpM to an MEP. This binding is typically done inside the AIP using DREAM library calls. The MEPs which are

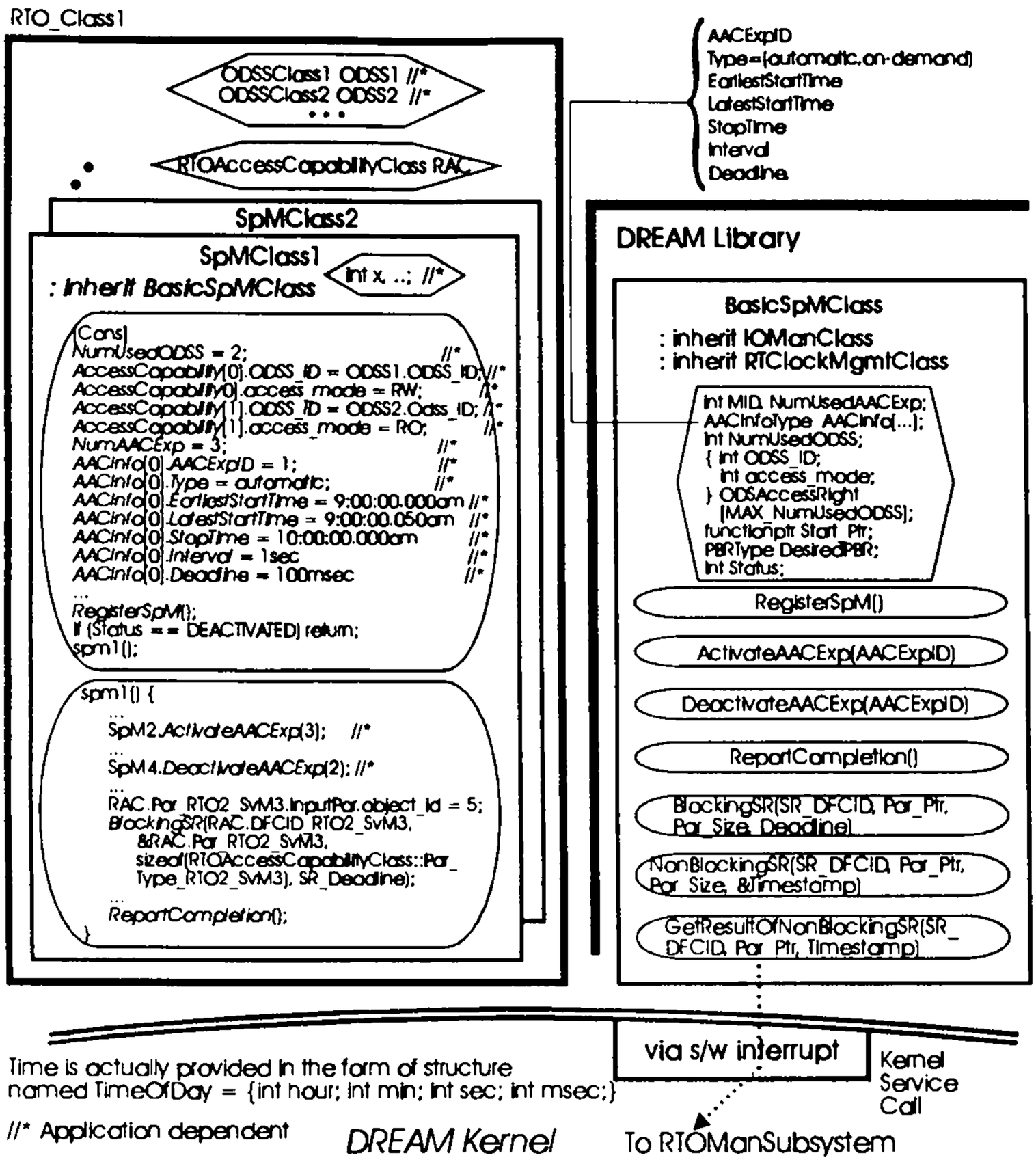


Fig. 4. The SpMClass and the DREAM library support for SpM management

bound to the SpMs may be generic or custom-tailored for the SpM. In the case of *generic MEPs*, the process priority, process stack size, etc. are fixed to some pre-defined standard values. On the other hand, these values may be custom-tailored for *custom MEPs*. If an SpM is left unbound, it is assigned by the DREAM kernel to a free MEP each time it is activated.

4.5 *Construction of SvMs*

Figure 5 shows the construction of an *SvMClass* through an inheritance of the *BasicSvMClass* of the DREAM library. The *BasicSvMClass* provides functions for registering the SvM with the kernel, receiving a service request, replying to a service request, making service requests to other SvMs in either a blocking or a non-blocking manner, retrieving the results of non-blocking service requests, informing the kernel about the completion of one execution run, etc.

When an SvM is registered with the kernel, the kernel automatically assigns a service-request DFCID to the SvM, broadcasts the DFCID of the SvM on a special global broadcast DFC, and then enables the SvM to receive service-requests via this DFC. Also, the kernel assigns a unique result-return DFCID to an SvM, so that the SvM may call for other SvMs and receive results via this DFC. Unlike in the SpM case, the assignment is done at each time of invoking the SvM, rather than at the SvM registration time, and thus if multiple invocations of the same SvM are concurrently active, each SvM invocation is assigned a unique result-return DFCID.

SvM itself may be a client of some other SvM(s), and an SvM client may make both blocking and non-blocking service requests. In addition, an SvM may also pass a client request to another SvM by using a *client-transfer* call, see (Kim, 1994a). The latter SvM may again pass the client request to another SvM. This chaining sequence may repeat until the last SvM in the chain returns the results to the client. The main motivation behind such a client

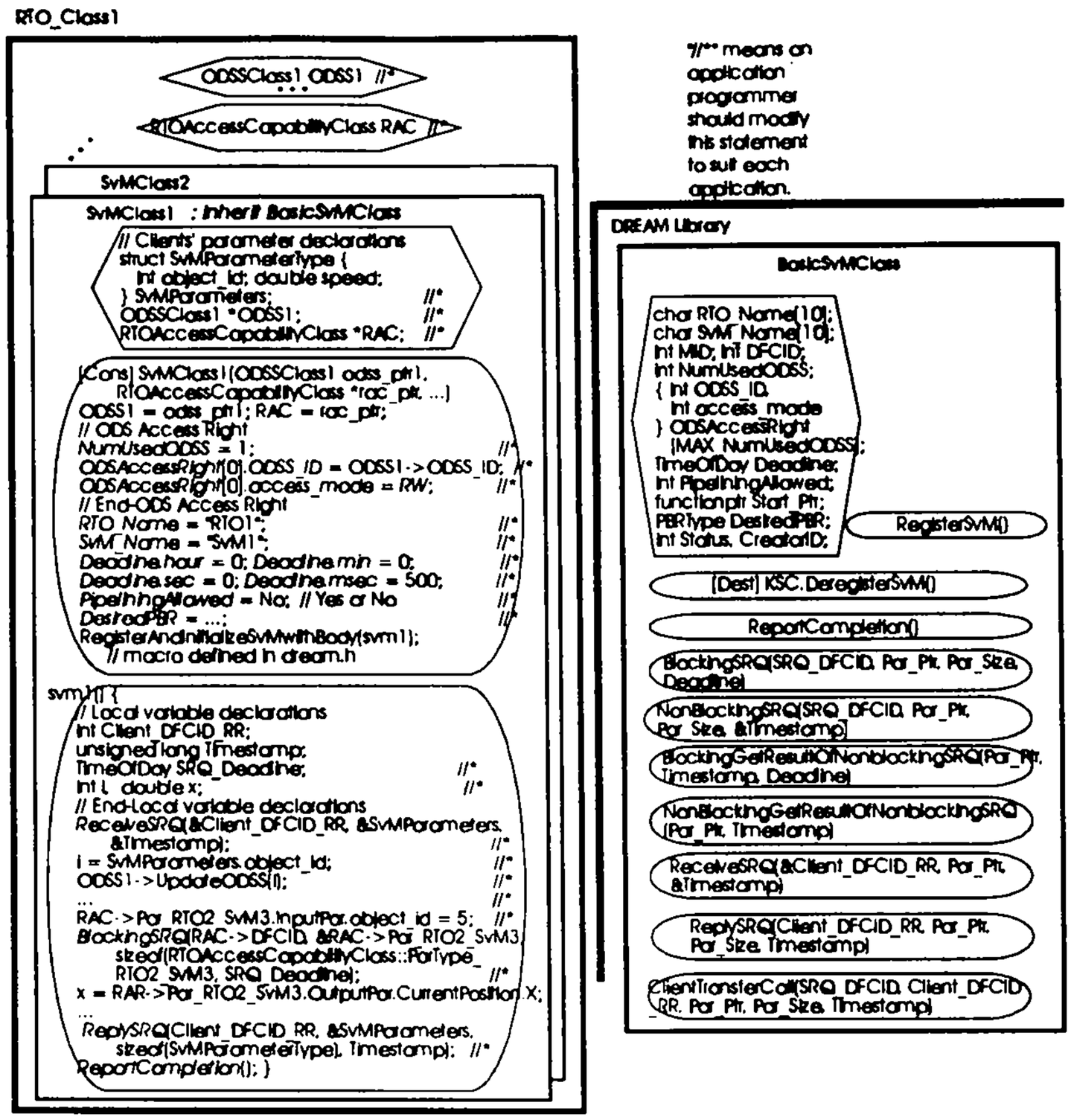


Fig. 5. The SvMClass and the DREAM library support for SvM management

transfer call stems from the basic concurrency constraint which requires an execution of an SvM to be made only if a sufficiently large time window between potentially conflicting SpMs opens up. Hence, in certain situations a highly complicated SvM may never be executed due to lack of a wide enough time window. One way to get around this problem is to divide the SvM into multiple smaller SvMs, SvM1, .. SvMx. A client can then call each smaller SvM each time. Calling each smaller SvM incurs communication overhead of transmitting a request to the smaller SvM and obtaining the results. Eliminating most of such communication overhead is the motivation behind an arrangement in which the client calls the first SvM and the latter passes on the client to another SvM and so on until the last SvM of the chain returns the results to the client.

5. SUMMARY OF MAJOR FEATURES OF THE DREAM LIBRARY

(1) The DREAM library hides various details of the passing of parameters from the application layer to the DREAM kernel. Each DREAM library class houses a pre-defined set of data items that just need to be initialized by the application program before calling an appropriate function in the class. This function will correctly retrieve the parameters from the data area of the class, package them, and make the correct KSC. Any result-returns from the kernel are saved in the data area of the class.

(2) To ease the application programmer's task of designing the application interface to the DREAM library, the data items in the DREAM library classes are often initialized with default values. An application programmer may thus choose to pass these default values to the DREAM kernel when calling for a DREAM kernel service. In such a case, the application programmer needs to just call the DREAM library function of the appropriate class without reinitializing any of the data items of the class.

The power of the DREAM library in easing the efforts of structuring C++ RTO.k objects has been positively tested through several sizable distributed RTO.k application program developments. The developed real-time applications include a military command and control prototype, a simple steel mill control prototype, and an advanced traffic management system (ATMS). The military comm-

and control prototype implementation consists of 9 RTOs and about 20,000 lines of C++ code, and other implemented applications are smaller.

However, the DREAM library still burdens the application designer with the chore of dealing with some low-level details in redundant forms which could be automatically deduced from a high-level abstract, yet accurate, representation of RTO.k objects. An extension of the C++ language to support abstract RTO.k programming is the most natural path to follow in this direction. The authors are currently developing one such extension named the C++T language, and RTO.k objects written in C++T will be processed by a combination of a C++T-to-C++ preprocessor and a commercial C++ compiler.

6. CONCLUSION

The DREAM kernel offers guaranteed timely services to real-time applications. The DREAM library considerably reduces the efforts required for programming applications in the form of an RTO.k object network. This suite of tools represents a significant step forward in realizing the GT design paradigms. Nevertheless, it represents a very small portion of the desirable software engineering environ-ment that can fully support GT design. The execu-table module of the DREAM kernel v3.0 along with the DREAM library class definitions will be available

in the world-wide web at http://www.eng.uci.edu/ dream/dream-lab.html, starting April 1997.

ACKNOWLEDGMENTS

The research work reported here was supported in part by US Navy, NSWC Dahlgren Division under Contract No. N60921-92-C-0204, in part by the California Transportation Department via the UCI Institute for Transportation Studies, in part by the UC MICRO program 96-169b, and in part by Hitachi, Ltd., IBM, Postech, ETRI, and LG Electronics.

REFERENCES

Attoui, A. (1991). An object oriented model for parallel and reactive systems, *Proc. IEEE CS 12th Real-Time Systems Symp.*, pp. 84-93.

Bihari, T., Gopinath, P. and Schwan, K. (1989). Object-oriented design of real-time software, *Proc. IEEE CS 10th Real-Time Systems Symp.*, pp.194-201.

Ishikawa, Y., Tokuda, H. and Mercer, C. W. (1992) An object-oriented real-time programming language, *IEEE Computer*, pp. 66-73.

Kim, K.H. et al. (1994a). Distinguishing features and potential roles of the RTO.k object model, *Proc. 1994 IEEE CS Workshop on Object-oriented Real-time Dependable Systems (WORDS)*, Dana Point, pp.36-45.

Kim, K.H. and Kopetz, H. (1994b). A Real-Time object model RTO.k and an experimental investigation of its potentials, *Proc. 1994 IEEE CS Computer Software and Applications Conf. (COMPSAC)*, Taipei, pp.392-402.

Kim, K.H et al. (1995a) Realization of autonomous decentralized computing with the RTO.k object structuring scheme and the HU-DF inter-process-group communication scheme, *Proc. IEEE CS Int'l Symp. on Autonomous Decentralized Systems (ISADS 95)*, Mar. 1995, Arizona, USA, pp.305-312.

Kim, K.H. et al. (1995b). A timeliness-guaranteed kernel model - DREAM kernel and implementation techniques, *Proc. 1995 Int'l Workshop on Real-Time Computing Systems and Applications (RTCSA 95)*, Tokyo, Japan, pp.80-87.

Kim, K.H. et al. (1996). Real-time simulation techniques based on the RTO.k object modeling, *Proc. 1996 IEEE CS Computer Software and Applications Conf. (COMPSAC)*, Seoul, Korea, pp.176-183.

Kopetz, H. and Kim, K.H. (1990). Temporal uncertainties in interactions among real-time objects, *Proc. IEEE CS 9th Symp. on Reliable Distributed Systems*, Huntsville, AL, pp.165-174.

Mori, K. (1993). Autonomous decentralized systems: Concept, data field architecture, and future trends, *Proc. IEEE CS Int'l Symp. on Autonomous Decentralized Systems (ISADS 93)*, Kawasaki, Japan, pp. 28-34.

Shrivastava, S.K. and Waterworth, A. (1991). Using objects and actions to provide fault tolerance in distributed, real-time applications, *Proc. IEEE CS 12th Real-Time Systems Symp.*, pp.276-285.

Takashio, K. and Tokoro, M. (1992). DROL: An object-oriented programming language for distributed real-time systems, *Proc. OOPSLA*, pp. 276-294.