99-N-NS-04-A-01

RTS(Real-Time Simulator) 기술 개발

RTS Technology Development

시뮬레이션 모델링 및 가시화

Computer Simulation Modeling and Visualization

포 항 공 과 대 학 교

과 학 기 술 부

제 출 문

과학기술부 장관 귀하

본 보고서를 "시뮬레이션 모델링 및 가시화" 과제(중과제명-RTS 기술개발)의 보고서로 제출합니다.

2000. 10. 14

주관연구기관명 : 포항공과대학교

주관연구책임자 : 강 교 철

연 구 원 : 박 찬 모

김 정 현

위탁연구기관명 : Univ. of California, Irvine

위탁연구책임자 : 김 광 회

요 약 문

Ⅰ. 제 목

시뮬레이션 모델링 및 가시화

Ⅱ. 연구개발의 목적 및 필요성

사회적으로 중요한 실시간 시스템들은 그 규모의 거대함과 복잡성에 의해 사용자의 요구사항을 정확하게 규명하기가 어렵고 개발하면서 당면하는 의사 결정 사항들에 대해서 결정하기가 어렵다. 특히 시간적 행위에 대한 예측과 증명이 매우 중요한데 이를 해결하는 방법으로 수학적인 방법이 수없이 연구되어 해결책이 제시되었고, 실제 응용해보려는 시도가 있어왔다.[Goldsack] 하지만, 이러한 방법은 매우 복잡하고 그 적용 범위 및 실용성에 한계가 많아 시스템의 개발 초기단계에서 실시간 시스템의 시간적 행위 등의 사용자의 요구사항을 규명, 증명하고 중요 설계관계 사항들에 대해서 결정을 내리는데 시뮬레이션이 유용하게 사용되어 왔다. [Luiq]

현재 실시간 시스템의 시뮬레이터에 대한 요구는 급속히 확산되고 있으며 시스템 개발에 있어 그 시스템의 실시간 시뮬레이터 개발에 드는 비용이 시스템 개발에 소요되는 경비를 능가할 정도로 그 중요성이 커지고 있다. 새로이 개발될 구축 지원 시스템을 활용하여 주어진 응용분야에 적합한 요구사항 명세 시뮬레이터를 신속하고 경제적으로 구축하여 응용시스템의 전 개발과정을 통하여 모든 중요 설계 결정 및 기술적 타당성, 시스템의 성능 및 신뢰성을 경제적으로 검증할 수 있고 시스템의 훈련기나 테스트베드 등으로 사용할 수 있다. 이는 안전하고 우수한 성능의 시스템이나 테스트베드를 경제적으로 생산할 수 있게 된다는 것을 의미한다.

본 연구는 실시간 시스템의 개발 초기 사용자 요구 사항의 명세 단계에서 상세 설계, 구현의 전 단계에 거쳐 사용할 수 있는 실시간 시스템 모델을 만들고 이 모델을 시뮬레이션할 수 있는 시뮬레이터를 만드는 것을 그 목적으로 한다. 또한, 실시간 시뮬레이션 디스플레이를 위해, VR(Virtual Reality)에 기반하여 실시간 시뮬레이션 실행 엔진과의 사용자 인터페이스를 개발한다. 이러한 것들을 기반으로 하여 중요한 응용 분야에 대한 실시간 시뮬레이션 모델을 구성하여 실제 사용 가능한 실시간 시뮬레이터를 구축함으로써, 실시간 시뮬레이터 프로토타입에 대한 방법론을 정립한다.

Ⅲ. 연구개발의 내용 및 범위

본 연구의 내용 및 범위는 다음과 같다.

구 분	연 구 개 발 목 표	연구개발내용 및 범위
1차년도 (1997)	• 실시간 객체 시뮬레이션 모델 개발 방법론 및 도구 시제품 개발	• 공간 정보를 포함한 실시간 시스템 명세 방법론 및 명세 도구 개발 • 시뮬레이터 운영자 및 사용자를 위한 VR 인터페이스의 개발 방법론 개발 및 예제 시스템 시제품 개발
	• 가시화 방법론 개발	• 기존 가시화 도구 고찰 및 목표 환경 선정 • 공간정보(Form) 명세 방법 개발 • 가시화 도구 설계 및 시제품 개발, 시뮬레이터와 연동하여 테스트

	• 목표 시스템의 시제품 개발	하여 테스트 • 목표 시스템을 선정하여 시스템 모델 작성, 객체지향적 행위, 기능 시뮬레이터 시제품 개발 • 목표 시스템에 대한 객체지향적 행위 및 기능과 시스템의 실시간적 요구사항에 대한 시스템 분석 • 목표 시스템의 가시화 시제품 개발
2차년도 (1998)	• 요구명세로부터 TMO 객체를 생성하는 생성기 개발 • 가시화 도구 및 VR(Virtual Reality) 인터페이스 저작도구 개발 • 실제 시스템 시뮬레이터의 시제품 개발	• 요구 명세로부터 TMO 객체를 자동으로 생성해내는 도구 개발 • VR(Virtual Reality) 인터페이스 저작도구 개발 • 가시화 도구 개발 완료 • 가시화, VR인터페이스와 시뮬레이터의 통합 • 목표 시스템의 명세 완성 및 실시간 시뮬레이터 개발
3차년도 (1999)	• 실시간 객체 모델의 실시간 시뮬레이터 개발 및 거대한 실제 시스템 시뮬레이터 구축	• 객체 지향형 실시간 명세 시뮬레이터, VR 인터페이스, 가시화 시스템, 실시간 시뮬레이션 엔진의 통합 • 목표 시스템의 행위, 기능 명세 실시간 시뮬레이터 구축 • 목표 시스템 시뮬레이터의 운영자 및 사용자 VR 인터페이스 저작 • 목표 시스템의 가시화 모델 구축 • 목표 시스템 테스트 및 성능 검증

Ⅳ. 연구개발결과

본 연구를 통해 공간 정보를 포함한 실시간 시스템의 명세 방법이 구축되었으며, 명세된 결과에 대한 시뮬레이션 도구가 완성 되었다. 또한 실시간 시스템의 수행 환경인 TMO로 디자인하여 코드 생성을 하는 도구가 개발되었다. 이러한 연구 결과가 실제 선박 시뮬레이터 시스템에 성공적으로 적용되었다.

본 연구 개발의 결과를 보다 자세히 열거하면 아래와 같다.

- 실시간 시스템의 명세 및 분석 방법 구축

  실시간 시스템의 기능과 행위를 명세하고, 이를 Validation할 수 있는 방법론 및 도구 개발

- 실시간 시스템의 가시화 도구 개발

  실시간 시스템의 외부 환경을 3차원 가시화하고 행위와 기능을 명세할 수 있는 객체 지향적이고 점증적인 방법론 및 도구 개발

- TMO 기반의 객체 지향적인 실시간 디자인 방법 및 코드 생성기 개발

  명세된 결과를 TMO 기반의 객체 지향적인 설계 모델로 만들고 이들을 분산 노드에 배치할 수 있는 디자인 방법 및 도구를 개발하고, 만들어진 디자인 모델을 바탕으로 자동으로 코드를 생성하는 방법 및 도구 개발

- 실제 시스템에 적용

  실제 거대 시스템인 선박 시뮬레이터 시스템을 상기의 방법과 도구를 사용하여 상용화 수준으로 제작

V. 연구개발결과의 활용계획

본 연구는 3차년도에 1, 2차 년도의 연구를 바탕으로 실제 거대 시스템인 선박 시뮬레이터에 적용하여 실효성을 검증하였다. 이번에 적용된 시뮬레이터의 경우, 선박 조정자들을 위한 훈련 프로그램으로서, 실시간 명세 방법을 통해 요구 사항을 정확히 분석하고, TMO를 통해 실시간 성질을 만족시키는 디자인과 구현을 하였으며, VR 인터페이스를 통해 현실감있는 외부 환경을 조성하여 성공적인 활용이 가능하였다. 이와 같은 모의 훈련 프로그램 외에도 원자력 발전소 시스템이나 교통 제어 시스템 등과 같은 위험도가 높고 복잡한 시스템에 적용하여, 실제 시스템을 구축하기 이전에 미리 문제점을 파악하여 신뢰성을 높이고, 개발 비용을 줄일 수 있게 된다.

현재 본 연구를 통해 개발된 방법은 ASADAL 이라는 이름의 통합 도구로 만들어졌으며, 연구 결과의 활용을 위해 관심 있는 업체들과의 Consortium 구성을 계획 중에 있다. 이러한 Consortium의 목표는 본 연구 과제를 통해 산출된 실시간 시스템의 체계적인 개발 방법을 소개하여 실시간 시스템 개발에 있어서 아직 낙후되어 있는 국내의 현실을 개선하는 것이다. 이를 위하여 2000년도에 Comdex Korea2000에 참가하여 연구 결과를 소개하고, Consortium의 취지를 설명하였다.

SUMMARY

Mission / Safety critical real time systems require thorough analysis of their dependability and correctness, however, due to their large scales and complexity, it is very difficult to verify such system characterisitcs. Particularly important is the prediction and verification of the system's temporal behavior, and many mathematical approaches have been developed to tackle this problem, only to find relatively little utility due to the difficulty in applying the method and its limited scope. Instead, simulation methods have been widely used to validate user and system requirements, and make decisions upon design and maintenence of real time systems.

In this aspect, there is an increasing need for real time system simulators, and sometimes, the effort required design the simulator often supersedes that of the actual system. However, once such a simulator is deployed, it can be effectively used to design and implement a more safe and reliable real time system, and even be used as a testbed or training facility.

The goal of this research is to propose a methodology for buildling a simulation system for large scale safety critical real time systems that can be used for the aforementioned purposes. In addition, to use the simulator as an effective training facility, we propose to develop a virtual reality based interface for it.

Our research has proceeded in the following manner. In the first year (1997), we have developed methodologies (and prototype supporting tools) for modeling and visualizing real time simulation objects and their form. The methodology was tested on a small scale real time system for demonstration. In the second year (1998), further design tools were developed that produces, from the abstract real time object models, actual executable TMO object codes. At the same time, further techniques were developed to consider system aspects for the virtual

training environment. The third year activities mostly involved integrating the overall methodologies (and supporting tool called the ASADAL) and applying it to buildling a large scale real time system (e.g. Ship Simulator / Generator System).

From the experience of applying our method and using our tool to develop an actual large cale system like the Ship Simulator, we are confident that the same method can be applied to other domains easily and due to making use of VR interfaces, the resulting simulation system has much higher learning effect by providing more realitic training environment.

CONTENTS

목 차

제 1 장 서론

제 1 절 연구 개발의 필요성

1. 연구개발의 기술적 필요성

실시간 시스템이란 일반적으로 컴퓨터 계산결과의 정확성 뿐만 아니라 그 계산의 결과가 나오는 시점이 중요한 시스템을 말하는데, 예를 들면 공장 제어 시스템, 항공기 시스템, 고속전철 시스템 등 산업 및 사회의 고속화에 따른 컴퓨터 제어 시스템들이다. 이러한 시스템은 특히 시간 제약성(timeliness), 신뢰성(reliability), 그리고 가용성(availability) 등의 측면이 매우 강조된다.

사회적으로 중요한 실시간 시스템들은 그 규모의 거대함과 복잡성에 의해 사용자의 요구사항을 정확하게 규명하기가 어렵고 개발하면서 당면하는 의사 결정 사항들에 대해서 결정하기가 어렵다. 특히 시간적 행위에 대한 예측과 검증이 매우 중요한데 이를 해결하는 방법으로 수학적인 방법은 매우 복잡하고 그 적용범위의 한계가 많아 선진국에서 보통 시뮬레이션을 필수적인 과정으로 취급하여 왔다. [Kang98a], [Kang98c] 따라서, 시스템의 개발 초기단계에서 실시간 시스템의 시간적 행위 등의 사용자의 요구사항을 규명, 증명하고 중요 설계관계 사항들에 대해서 결정을 내리는데 시뮬레이션을 유용하게 사용하여 왔다. 이러한 사용자 요구사항의 유효성 검증과 실시간 성질의 만족성 검증 뿐 아니라, 최근 실시간 시뮬레이션은 개발 과정상에서 비행기 시뮬레이션이나 핵발전소와 같이 제작된 컴포넌트를 목표 시스템에서 직접 테스트하기에는 너무나 막대한 비용이나 인명 피해가 예상되는 경우 소프트웨어 테스트베드의 구축이나 운영자나 사용자의 훈련용 시뮬레이터 등에 응용되는 등 사용 범위가 확대되고 점차로 그 중요성이 증가함에 따라 외국에서는

이미 제어 시스템 개발시 그 실시간 시뮬레이터를 만드는 일을 동시에 진행하는 등 최근 실시간 시뮬레이션 기술이 수없이 요구되는 실정이다.

실시간 시뮬레이션은 아래의 세 가지 필수조건을 만족시켜야 한다.

1. 실제 응용환경에서 발생하는 여러 사건들의 상대적인 시간성이 정확하게 시뮬레이터에 의하여 모방되어야 한다. 따라서 사건들의 순서가 실제 환경에서나 시뮬레이터상에서 같아야 함은 물론이고 실제 환경에서 일어난 두 사건 간의 시간간격과 상응하는 시뮬레이터상에서의 시간간격과의 비율이 어느 두 사건을 고려하더라도 항상 일정하거나 좁은 변동폭을 보여야 하는 것이다.

2. 시뮬레이션의 속도는 실제 응용시스템이나 환경이 진행하는 속도와 비슷하거나 보다 빨라야 한다.

3. 대량의 시뮬레이션의 결과를 효과적으로 분석해 낼 수 있어야 한다. 즉, 시뮬레이션 진행중 그 의미를 명확히 알 수 있어야 하며 시뮬레이션이 끝나고 시뮬레이션의 결과를 일목 요연하게 알 수 있어야 한다.

따라서 이러한 실시간 시뮬레이션의 목적은 응용환경에 존재하는 여러 물체간의 상호작용의 시간적인 면을 정확하게 모방하는데 있다. 두 번째 목적은 여러 개의 상호 협동적인 프로그램 모델 등의 시간적인 면을 정확하게 모방하는데 있다. 그리고 세 번째는 시뮬레이션의 결과를 최대한 쉽고 정확하게 사용자에게 전달해야 한다는 것이다.

정형적 사용자 요구사항 명세의 시뮬레이터는 대규모 응용시스템 전체를 적당한 수의 Stochastic 변수로 구성된 거시적인 모델 (macroscopic simulator)로 나타내어

이를 컴퓨터시스템이 수행하는 프로그램으로 구현하는 것을 의미한다. 이러한 시뮬레이터의 대표적인 모델로는 Queueing Model, Marcov Chain이나 Stochastic Petri-Net등이 있다. 이러한 시뮬레이터는 빠른 시스템 동작 예측에 쓰이며 사용자, 의뢰인, 개발자간의 이해 일치를 돕고 시스템 요구 사항 내의 문제점을 발견해 내는 것을 목표로 한다. 이 시뮬레이션은 실시간으로 이루어지지는 않지만 모든 실시간 성질에 대한 검증이 사용자 대화형이나 배치(Batch) 모드를 통해 이루어진다. 특히 배치 시뮬레이션 같은 경우 실시간 시뮬레이션에서는 불가능한 대량의 오랜 시간 시뮬레이션의 결과를 얻을 수 있어 유용하다.

일단 사용자의 요구사항과 전체적인 시스템의 중요기능 및 중요구조가 결정되면 시스템의 자세한 설계가 다단계를 거쳐 이루어지며 이 설계과정에서의 여러 가지 결정사항은 시스템의 실시간 성능에 직접적인 영향을 미치게 된다.

시스템의 분할과 분할된 시스템의 네트워크를 통한 연결방법, 태스크로의 분할 및 태스크의 스케줄링 방법 등 설계 과정에서 결정되어야 할 많은 사항들이 실시간 성능에 직접적인 영향을 미치는데 고정밀 실시간 시뮬레이션이 이와 같은 결정을 체계적으로 내리며 결정되는 사항이 시스템 성능에 미치는 영향을 측정할 수 있도록 도와준다. 따라서 고정밀 실시간 시뮬레이터는 목표 응용 시스템의 모델을 컴퓨터 시스템이 수행하는 것을 구현하는 것을 의미한다.

고정밀 실시간 시뮬레이션은 정형적 사용자 요구사항 명세의 시뮬레이션보다 비교적 모델설정에 많은 노력이 들고 비용이 많이 드나 시스템을 설계, 개발, 테스팅하는데 보다 정확한 자료를 제공한다. 즉, 실제 시스템과 완전히 같은 방식으로 실행이 되니까 가시화 시스템과 연동하여 가상 현실 시스템을 구축, 비행기 조종사 훈련 시스템이나 핵 발전소 운영자 훈련 시스템과 같은 훈련 시스템에 직접 사용가능하고 실제 시스템 개발중 만들어지는 각 시스템 부품(Component)이나 부분시스템

(Subsystem)등과 연결해 이들을 테스트할 수 있는 테스트베드로도 사용할 수 있다. 이에 비해, 정형적 사용자 요구사항 명세의 시뮬레이션은 시뮬레이션 모델을 설정하고 시뮬레이터를 준비하는 것이 비교적 용이하며 비용이 적게 들어 시스템 요구사항을 설정하기 위한 Rapid prototyping tool로 적합하다고 할 수 있다. 그러나 고정밀 실시간 모델의 Library가 확립됨에 따라 고정밀 실시간 시뮬레이션의 비용이 절감될 것이며 두 방법은 자연스럽게 통합될 것으로 예상된다.

결국, 정형적 사용자 요구사항 명세의 시뮬레이터는 구축시간이 짧아야 하며 선정된 변수들을 넓은 범위 안에서 여러모로 값을 바꾸어 가면서 빠르게 시뮬레이션을 수행할 수 있게 구성되어야 한다.

현재까지 국내에서 올바로 활용되지 못하였으나 교통제어시스템, 금융관리시스템, 에너지 유통시스템, 정보통신시스템 또는 대규모 공장 및 재해 관리 시스템의 설계단계에서 고성능 정형적 사용자 요구사항 명세의 시뮬레이터의 활용은 요구사항의 검증 및 유효성 검사에 필수적인 요소이다.

고정밀 실시간 모델은 목표 응용시스템의 세부적인 구성 및 동작들을 분명히 표현하는 것으로서 사회근간시설 및 산업기반 기술 같은 대규모 실시간 시스템에 대한 상세 설계 모델은 보편적으로 다음과 같은 성격을 갖는다.

- 방대한 양의 각종 소프트웨어, 하드웨어, 물리 객체
- 각 객체의 동작을 상세히 표현, 특히 각 동작의 시간적 요소 표현
- 목표 응용시스템의 계속적인 확장 및 개조에 따른 모델의 확장 및 개조

특히, 소프트웨어나 하드웨어 객체, 실재하는 물리 객체들은 디지털뿐 아니라 아날로그(Analog)적 특성을 가지므로 이들을 제대로 시뮬레이션하고 의미 있게 연결

하는 방법 및 시뮬레이션하는 기술이 중요하다.

- 대규모 실시간 시스템의 고정밀 시뮬레이터를 구축하는 데는 다음과 같은 난관을 극복하여야 한다.

- 정확하고 이해하기 쉽고 확장 및 개조가 용이한 객체 지향적 모델 구축

- 대규모의 고정밀 모델을 분산형, 병렬 처리형 등의 대용량 고성능 컴퓨터 시스템에서 빠른 시간 내에 수행되는 프로그램으로의 전환 자동화

- 고정밀 시뮬레이터를 구축하는 데 분산형, 병렬 처리형 등 고성능 컴퓨터 시스템을 사용하지 않으면 시뮬레이터 수행시간이 방대해져 효율성이 급감하는 문제가 있다.

- 엄격히 실시간 성질의 만족하는 시뮬레이터의 구축

대규모 실시간 시스템의 고정밀 실시간 모델을 바탕으로 한 시뮬레이터는 종래의 시뮬레이터로 얻을 수 없는 다음과 같은 결과를 가능케 한다.

- 신규 개발 대규모 시스템의 최적 설계 선정 또는 기존 실시간 시스템의 최적 운영 방식 선정

- 불규칙적인 동작을 하도록 설계된 시스템의 시간별 성능의 정확한 조사

- 목표 시스템의 새로운 설립이나 주요 확장 보수의 경우 시뮬레이터의 일부를 시스템 테스트에 사용 가능

이번 시뮬레이터에는 특히 가상 현실(Virtual Reality; VR) 사용자 인터페이스 및 실제 시스템 운영환경을 모두 실제와 같이 보여주는 가시화 시스템이 포함되어 실제와 같이 사용자가 가상 인터페이스를 이용해 시스템 개발 초기에서 개발 끝까지

시스템을 직접 사용해 볼 수 있게 될 것이다.

시뮬레이터 기술은 1950년대 이후 꾸준히 발전되어 왔으나 여지껏 실시간성이 실제 시스템의 테스팅에 이용될 수 있을 정도로 믿을 수 있는 고정밀 시뮬레이터를 구축할 수 있는 기술은 확보되지 않은 상태이다.

병렬처리 컴퓨터 시스템 기술 및 객체지향 소프트웨어 구성기술 등이 최근에 실용화 단계에 접어들어 고성능 고정밀 시뮬레이터 구축 활동이 급증할 시점에 도달하였다. 이러한 활동들은 국가산업 및 경제 활동에 원동력을 증가시키는 결과를 낳을 것이다.

2. 경제, 산업적 측면

현재 고정밀 실시간 시뮬레이터에 대한 요구는 급속히 확산되고 있으며 시스템 개발에 있어 그 시스템의 실시간 시뮬레이터 개발에 드는 비용이 시스템 기발에 소요되는 경비를 능가할 정도로 그 중요성이 커지고 있다. 새로이 개발될 구축지원 시스템을 활용하여 주어진 응용분야에 적합한 요구사항 명세의 시뮬레이터와 상세 설계 시뮬레이터를 신속하고 경제적으로 구축하여 응용시스템의 전 개발과정을 통하여 모든 중요 설계 결정 및 기술적 타당성, 시스템의 성능 및 신뢰성을 경제적으로 검증할 수 있고 시스템의 훈련기나 테스트베드 등으로 사용할 수 있다. 이는 안전하고 우수한 성능의 시스템이나 테스트베드를 경제적으로 생산할 수 있게 된다는 것을 의미한다. 이런 특징은 생산된 실시간 시스템에 경쟁력을 부여하여 내수 및 해외 수출에 크게 기여할 것이다.

실시간 시스템은 교통, 통신, 공장자동화 등 첨단산업의 근간을 이루는 분야에 핵심기술로 사용되고 있고 근년의 폭발적인 컴퓨터 관련 부품 기술의 향상으로 인

한 대규모 실시간 시스템의 폭증이 예상되기 때문에 고등 시뮬레이터 구축 지원 시스템의 등장은 첨단산업 발전에 크게 기여할 것이다. 따라서 고등 시뮬레이터 구축 지원 시스템의 개발은 한국의 경제 산업 발전의 현시점에서 아주 긴요한 적시의 과제이며 선진국에서도 아직 크게 기술이 정립되지 못한 분야이다. 고정밀 시뮬레이터 구축지원 시스템 개발에 대한 국가적 차원에서의 지원은 한국 소프트웨어 기술의 국제적 위상 제고에 큰 효과를 내리라 예상된다.

3. 사회, 문화적 측면

중요한 실시간 시스템들은 원자력 발전소, 교통 정보 시스템, 항공기 제어 시스템 등 안전성과 신뢰도가 중요시되는 분야에서 흔히 발견되는데 이러한 시스템들은 한번의 오류적 행동이라도 심각한 사회적 문제를 일으킨다. 따라서 실시간 시스템들은 개발시 충분한 검증 단계를 거쳐서 완성되어야 하는데, 이 때 시뮬레이션이 유용하게 사용된다. 시뮬레이션은 실시간 시스템의 여러 개발 단계에서 시스템의 여러 성질들을 경제적으로 검증한다. 또한, 이러한 실시간 시스템의 경제성이나 안전성 때문에 실제 환경에서 실시간 시스템을 직접 테스트한다는 것은, 예를 들어 비행기 제어 시스템을 항공기에 넣어 테스트한다거나, 핵발전소 제어 시스템을 핵발전소에서 테스트한다는 것들은 사회적으로 용납되지 않는 일이기 때문에 않는 일이기 때문에 실시간 시뮬레이션은 사회적 요구에 꼭 들어맞는 좋은 기술적인 해결책이 될 것이다.

시뮬레이션으로 인한 안전성과 경제성의 획득으로 사회 각 분야에 실시간 시스템들이 다양하게 개발되어 적용되면 여러 사회 운영이 전산화되어 고도로 편리한 문화적 환경을 이룰 수 있게 된다.

제 2 절 연구개발의 목적

연구 개발의 최종 목표는 객체 지향적인 실시간 시뮬레이션 모델링 기술, 가시화 및 VR 인터페이스 모델링 기술 개발 이다.

실시간 시스템의 개발 초기 사용자 요구 사항의 명세 단계에서 상세 설계, 구현의 전 단계에 거쳐 사용할 수 있는 객체 지향적인 실시간 시스템 모델을 만들고 이 모델을 시뮬레이션할 수 있는 시뮬레이터를 만든다. 이를 위해 요구 사항 명세의 시뮬레이터를 만들고 상세 설계 모델의 실시간 시뮬레이터를 만들고, 요구 사항 명세로부터의 자연스러운 설계로의 전환을 위한 자동 변환기도 만든다. 그리고 초고속 네트워크로 병렬 연결된 다중 노드 PC에 다중 프로세스 기반 실시간 시뮬레이션 실행 엔진인 TMO를 기반한 다중 노드 실시간 시뮬레이션 실행 엔진을 개발한다. 그리고, 실시간 시뮬레이션 디스플레이를 위해, 멀티미디어 및VR(Virtual Reality)에 기반하여 실시간 시뮬레이션 실행 엔진과의 사용자 인터페이스를 개발한다.

이러한 것들을 기반으로 하여 중요한 응용 분야에 대한 실시간 시뮬레이션 모델을 구성하여 실제 사용 가능한 실시간 시뮬레이터를 구축함으로써, 실시간 시뮬레이터 프로토타입에 대한 방법론을 정립한다. 실시간 시스템은 특히 외부 세계와 복잡하게 연결되어 있으므로 내부적 관점에서의 명세나 설계만으로는 제대로 된 시뮬레이션을 해 내기 힘들기 때문에 각 모델들은 여러 실세계에서 일어날 수 있는 현상들을 모델링할 수 있어야 한다. 이들 중에 특히 공간적인 정보들은 복잡한 물리 현상들과 관련되어 표현하기 쉽지 않으므로 가능하면 이 정보들을 간단히 모델링할 수 있게 한다. 그리고 이렇게 표현된 공간 정보를 이용하여 3차원으로 실제 세계에서 일어나는 일들을 시뮬레이터 사용자에게 가시화하여 보여줌으로써 명세나

시뮬레이션 결과를 한눈에 알아볼 수 있게 한다. 또한, 실제 상황과 같게 만드는 VR 인터페이스는 사용자들이 개발할 시스템을 초기부터 사용해 보게 함으로써 요구사항을 조기에 알아낼 수 있다는 장점이 있고 나아가 사용자들을 훈련시키는 데에도 사용할 수 있다는 장점이 있다.

제 2 장 국내외 기술개발 현황

비록 대규모 시스템의 시뮬레이터 지원 기능이 선진국에서 1980년대부터 등장하여 현재 통신, 공장 자동화 및 유통 분야에서 정형적 명세 시뮬레이터의 개발 활동이 증가하고 있으나, 아직 지금까지의 시뮬레이션 방법 및 도구들은 대규모 실시간 시스템들에 대한 시뮬레이션을 수행하는데 많은 부족한 점을 갖고 있다[Aim85, Ave82, Com84, Cou91, Pet89]. 지금까지는 대부분 시뮬레이터를 일반적인 프로그래밍 언어를 이용하여 주먹구구식으로 개발, 개발 비용도 많이 들뿐 아니라 시뮬레이션 환경의 설정을 변화하는 것 등에 쉽게 대처하기 어렵다. 게다가, 그 시뮬레이터가 진짜로 실제 시스템과 시간적인 면에서 똑같이 동작하는지를 보장할 수는 없었다. 그리고 실시간 시뮬레이션의 개발 [Fuj90, Jef85, Phi93]과 가상 현실 인터페이스와의 결합 그리고 시뮬레이션 결과의 다각도적 분석 등에 대한 연구가 아직 미흡하다. 실시간 시뮬레이터의 늦은 또 느린 발전은 실시간 응용 시스템의 실시간 시스템 모델링 기법이 최근까지 제대로 개발되지 못한데 기인한다. 객체 지향형모델이 이러한 시뮬레이션을 위한 모델로 적합하리라는 믿음은 지난 28년간 꾸준히 축적되어 왔다[Boo91, Dah72, Dav90, Dou92, Pag89, Rum91, Zei90]. 이를 다시 실시간 시스템을 위한 모델로 확장하려는 노력은 근년에 와서 활발해 졌으나[Bih91, Ish90, Sch87, Tak92, Tok89] 큰 성공을 거두었다고 보기 어려운데 본과제의 국제 협동 Partner인 Kane Kim 교수와 Vienna 공대의 Kopetz 가 공동 개발 한 TMO(Time Triggered Message Triggered Object; 구명 RTO.k) 모델의 장점은 제어 프로그램 모듈 뿐 아니라 모든 응용 환경에 존재하는 물체의 변화 및 상호 작용을 정확하고 아주 정밀하게 또 자연스러운 형태로 나타낼 수 있다는 데 있다. 여기서 말하는 정밀성 및 정확성은 양적인 면 뿐 아니라 시간적인 면에도 적용되며 병렬 진행면이 TMO 모델에서

는 자연스러운 형태로 표현된다. 따라서 이에 대한 연구가 핵심소프트웨어 기술 개발 사업의 일환으로 올바로 이루어진다면 앞으로 선진 외국들에 비해서 기술적인 우위를 점하는데 결정적인 역할을 할 것이라 판단된다.

제 3 장 연구개발수행 내용 및 결과

본 장에서는 본 과제를 통하여 개발된 연구 내용과 그 결과에 대하여 1차(1997년), 2차(1998년), 3차(1999년)년도로 구분하여 기술하도록 하겠다.

제 1 절 1차년도 연구개발수행 내용 및 결과

1. 연구개발 목표

본 세부과제의 최종 목표는 공간 정보를 포함한 객체 지향적인 실시간 시스템 명세 기술 및 그 명세의 효율적인 분산 시뮬레이션 및 가시화 기술 개발 이다.

실시간 시스템의 개발 초기 사용자 요구 사항의 명세 단계에서 상세 설계, 구현의 전 단계에 거쳐 사용할 수 있는 객체 지향적인 실시간 시스템 모델을 만들고 이 모델을 시뮬레이션할 수 있는 시뮬레이터를 만든다. 이를 위해 요구 사항 명세의 시뮬레이터를 만들고 상세 설계 모델의 실시간 시뮬레이터를 만들고, 요구 사항 명세로부터의 자연스러운 설계로의 전환을 위한 자동 변환기도 만든다.

실시간 시뮬레이션 디스플레이를 위해, 멀티미디어 및VR(Virtual Reality)에 기반하여 실시간 시뮬레이션 실행 엔진과의 사용자 인터페이스를 개발한다. 이러한 것들을 기반으로 하여 중요한 응용 분야에 대한 실시간 시뮬레이션 모델을 구성하여 실제 사용 가능한 실시간 시뮬레이터를 구축함으로써, 실시간 시뮬레이터 프로토타입에 대한 방법론을 정립한다. 실시간 시스템은 특히 외부 세계와 복잡하게 연결되어 있으므로 내부적 관점에서의 명세나 설계만으로는 제대로 된 시뮬레이션을

해 내기 힘들기 때문에 각 모델들은 여러 실세계에서 일어날 수 있는 현상들을 모델링할 수 있어야 한다. 이들 중에 특히 공간적인 정보들은 복잡한 물리 현상들과 관련되어 표현하기 쉽지 않으므로 가능하면 이 정보들을 간단히 모델링할 수 있게 한다. 그리고 이렇게 표현된 공간 정보를 이용하여 3차원으로 실제 세계에서 일어나는 일들을 시뮬레이터 사용자에게 가시화하여 보여줌으로써 명세나 시뮬레이션 결과를 한눈에 알아볼 수 있게 한다. 또한, 실제 상황과 같게 만드는 VR 인터페이스는 사용자들이 개발할 시스템을 초기부터 사용해 보게 함으로써 요구사항을 조기에 알아낼 수 있다는 장점이 있고 나아가 사용자들을 훈련시키는 데에도 사용할 수 있다.

당해 년도에는 다음과 같은 연구 개발 목표들이 설정되어 있다.

1. 실시간 객체 시뮬레이션 모델 개발 방법론 및 도구 시제품 개발

   a. 공간 정보를 포함한 실시간 시스템 명세 방법론 및 명세 도구 개발

   b. 시뮬레이터 운영자 및 사용자를 위한 VR 인터페이스의 개발 방법론 개발 및 예제 시스템 시제품 개발

2. 가시화 방법론 개발

   a. 기존 가시화 도구 고찰 및 목표 환경 선정

   b. 공간정보(Form) 명세 방법 개발

   c. 가시화 도구 설계 및 시제품 개발, 시뮬레이터와 연동하여 테스트

3. 목표 시스템의 시제품 개발

   a. 목표 시스템을 선정하여 시스템 모델 작성, 객체지향적 행위, 기능 시

뮬레이터 시제품 개발

b. 목표 시스템에 대한 객체지향적 행위 및 기능과 시스템의 실시간적 요구사항에 대한 시스템 분석

c. 목표 시스템의 가시화 시제품 개발

2. 연구 수행의 이론적, 실험적 접근 방법

연구는 먼저 해당 분야에서 기존의 세계적인 연구들에 대한 검증으로 시작하였다. 현재의 과학, 연구계의 연구 결과들은 물론 개발되어 상용화되어 있는 도구들에 사용된 기술들까지 모아 우리가 나아갈 방향과 비슷한 점, 다른 점, 그들의 단점 및 우리만이 가질 수 있는 장점들을 파악하고 연구 방향을 설정하였다.

결정된 연구 방향을 따라 연구를 진행함과 동시에 연구 결과(실시간 시스템 명세, 설계 및 코드 생성 방법)를 소프트웨어로 구현하는 작업을 병행하였으며 이를 위해 가장 적절한(개발 편의성, 범용성, 사용 용이성 등) 개발 환경 및 개발도구를 선택하였다.

연구 도중 계속적으로 아이디어 및 그 결과물들을 세계 및 국내 유수 저널과 회의에 발표함으로써 우리의 연구에 대해 알리고, 이로부터 반응을 얻어 연구에 반영하였다.

당해 년도에는 연구 결과를 계속적으로 여러 가지 실시간 시스템들에 적용하였으며 가상의 시스템들뿐 아니라 실제 산업체에서 사용되고 있는 소프트웨어에 직접 적용함으로써 방법 및 도구를 계속적으로 검증하였다.

개발 도구는 Spiral Model을 이용하여 난이도가 큰 것부터 Incremental하게 개발

하였다. 도구 유지보수성을 좋게 하기 위해 객체 지향적인 설계를 하였으며 각 객체가 변화가능성 있는 시스템의 요소들을 적절히 숨길 수 있게 하였다. [Boehm]

3. 연구개발 결과

가. 연구내용

소개

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 시제 논리(Temporal Logic), 시뮬레이션, 스케줄링, 태스크 할당(Allocation) 등의 여러 접근 방법들이 고안되었다. [Kang95] 실시간 시스템은 주로 컴퓨터로 만들어지는 제어 시스템(Control System)과 센서와 엑츄에이터 등으로 구성된 제어 시스템에 의해 제어되는 시스템(Controlled System) 등으로 구성된다. 일반적으로, 실시간 시스템의 시간적 행위에 대한 유효성 검사는 제어 시스템에 의해 제어되는 시스템이 어떻게 반응, 동작하는지에 대한 추상적인 예상과 가정에 근거한다.

하지만, 대부분의 실시간 시스템들은 그것이 제어하는 동적으로 변화하고 예측하기 어려운 환경에 존재하는 객체들과 물리적으로 복잡하게 상호작용한다. 이러한 실시간 시스템의 의미있는 물리적 시뮬레이션을 위해, 제어 시스템과 관계된 실재 객체들은 어떤 모양을 가지고 어느 정도의 공간을 차지해야 한다. 사실, 객체들의 여러 다른 모양과 설정(Configuration)에 의해 시뮬레이션하면 서로 다른 결과를 만들어 낸다. 예를 들어 외형상 전투기는 여객기와는 다른 유체 역학적(Aerodynamic) 특성을 보인다. 우리는 이러한 무게, 모양, 물질 등의 물리적 성질과 위치, 방향, 속도 등의 설정을 "형태(Form)"라 부른다.

형태는 기능에 영향을 준다. 예를 들어 크기가 다른 두 개의 로봇 운반기계는 서로 다른 작업 영역과 능력을 가진다. 따라서, 형태 데이터가 없으면 시스템의 행위(Behavior)와 기능(Function)을 옳게 명세하고 시뮬레이션하기 어렵다. 시뮬레이션을 통한 가시화(Visualization)는 시스템의 유효성 검증의 좋은 방법인데, 형태 데이터를 이용해 시뮬레이션함과 동시에 실제 환경을 보여줌으로써 비행기들이 아주 가까이 다가간다거나 두 로봇이 작업하다가 부딪히는 등 발생하지 않아야 할 상황이나 위험 상황을 한 눈에 볼 수 있다.

관련 연구

다음 그림과 같이 전통적으로 소프트웨어 공학 분야에서는 내장형 제어 시스템(Embedded Control System)의 명세와 그것의 시뮬레이션에 대한 연구를 행위와 기능에 촛점을 맞추어 왔고, 그래픽스와 가상 현실 분야에서는 형태 데이터의 실제와 가까운 가시화에 중점을 두어 왔다.

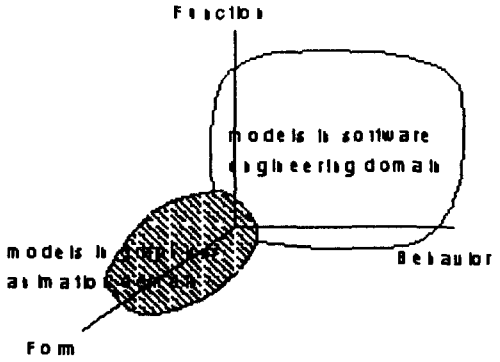


그림 1 실시간 요구사항 명세의 세 차원: 행위, 기능, 형태

이들 중 몇 가지의 독특한 관련 연구들에 대해 살펴보겠다. Gaskell과 Philips의 Executable Graphical Specification(EGS)는 DeMarco의 Data Flow Diagram(DFD) 표현 방법을 사용하여 기능 모델을 명세하고 프로세스를 자체 정의한 코드로 기술한 후 그것을 1대1 해석(Interpret) 방식으로 시뮬레이션함으로써 시스템의 기능을 살펴볼 수 있게 한다. 하지만, EGS는 데이터 처리 중심의 소프트웨어를 목표로 하여 만들어졌으므로 실시간 성질들을 명세할 수 없다. Harel은 그래피컬한 Computer-Aided Software Engineering (CASE) 도구인 STATEMATE를 만들었는데, 이 도구는 대형의 복잡한 실시간 시스템에 대한 명세, 분석, 설계, 다큐먼트 생성, 코드 생성 등을 한다. ASADAL은 실시간 시스템을 위한 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. 그러나, 이 접근 방법들은 제어 시스템 운영 환경 객체들의 행위엔 관심을 기울이고 있지 않다. 최근 Friesen 등이 실시간 제어 시스템과 그 작동 환경까지를 혼합적(Hybrid)으로 명세하는 객체 지향적인 방법을 제시하였으나 형태에 대한 것은 아직 대부분의 방식에서 간과되고 있다.

반면, 그래픽스와 Computer-Aided Design(CAD) 등의 분야에서는 명확하게 형태, 공간적 행위와 기능들을 표현하고, 이들을 모두 일관된 객체로 표현하는 것 등의 방법으로 ``움직이는" 세계를 만드는 방법이 연구되어왔고, 다년간 여러 도구들이 개발되었다. 그러나 이 분야에서는 그러한 것을 개발하기 위한 체계적인 기본환경(Systematic Framework)은 마련되어 있지 않다. 예를 들어 CAD로 모양을 만들고 일반적인 프로그래밍 언어 수준의 언어를 이용한 프로그래밍을 통해 행위를 넣고 있다. 이러한 구조화되지 않은 개발 방법은 개발물들을 유지, 보수하고 최적화(Tuning)를 어렵게 만들고 있다.

본 과제에서는 효과적인 유효성 검사 및 실제와 같은 실시간 시스템 명세의 시뮬레이션이 첫번째 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 방법인 DFD, Time-Enriched Statecharts(TES)와 새로 정의한 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) 행위 등을 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/PROTO는 DFD, TES와 VOS를 합친 새로운 통합 실시간 객체 모델(Unified Real-Time Object Model)을 제시한다.

개념

형태로의 확장을 하지 않은 ASADAL에서 주요 시스템 모델링 철학은 단계적인(Incremental) 개발 방법의 철학에 기반하며, 그에 적합한 명세 방법과 시뮬레이션 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[1] 등의 많은 생명 주기 모델에 의해 지지를 받고 있다.

물리적 성질들, 설정, 상호 작용하는 행위 등 형태에 관련된 속성들은 행위,

기능과 동시에 단계적으로 명세된다. 형태 명세가 제어되는 객체들의 모양과 물리적인 관계들 뿐 아니라 환경 객체들도 포함함에 주의해야 한다. 제어되는 시스템은 행위와 기능 명세가 만들어질 때 점차적으로 만들어지며1) 환경 객체의 명세는 대부분 미리 주어진다. 기본적인 생각은 각 차원의 명세들이 서로서로에게 영향을 준다는 것이다.

우리의 목표는 명세를 여러 상세화 단계에서 시뮬레이션하면서 제어되는 객체와 환경 객체들을 가시화함으로써 개발중인 시스템을 분석, 제어 시스템과 제어되는 시스템, 관련된 환경 객체들간의 상호 작용들을 관찰하는 것이다. 이렇게 하기 위해 제어 시스템 명세의 시뮬레이션은 외부 환경에 대한 시뮬레이션과 동시에 이루어져야 한다.

ASADAL/PROTO의 실시간 시스템에 대한 새로운 모델링 방법은 다음과 같은 세 가지 모델링 차원을 다룬다.

- 행위 명세는 사건(Event)에 대응하는 시스템의 상태 변화를 보여주며, 상태에 따라 적절한 기능들을 활성화시킨다.
- 기능 명세는 외부로부터의 입력에 대하여 데이터와 제어 신호의 계산 알고리즘들을 가진다.
- 형태 명세는 객체들의 여러 공간적 성질들을 정의한다. 행위와 기능 명세가 모든 객체에게 주어지는 반면, 형태 명세는 물리적 성질을 갖는 제어되는 시스템이나 환경 객체에 주어진다.

---

1 다른 방법은 이미 명세되어 있는 행위들과 기능들에 기반하여 내장된 라이브러리(built-in library)에서 형태를 고르는 것이다.

행위와 기능 모델들은 각각 ASADAL[6]의 명세 방법에서 제공하는 TES와 DFD의 정형적 명세 방법을 사용하여 만들어진다. DFD는 프로세스들을 명세하는데, 이들을 여러 개의 부분 프로세스와 그들의 입력/출력관계로 분할함으로써 시스템의 기능을 밝혀낸다. DFD는 그것을 제어하는 TES를 가질 수 있는데, TES는 어떻게 프로세스가 행동하는지와 시스템의 상태에 따라서 그 프로세스를 언제 실행할지를 명세한다. DFD와 TES를 이용한 정형적 명세 방법의 자세한 설명은 [6]을 참조하라. 형태는 VOS를 사용하여 표현한다. VOS의 주요 목적은 물리적 객체의 물리적 성질들과 설정을 기술하는 것이다(그림 2 참조).

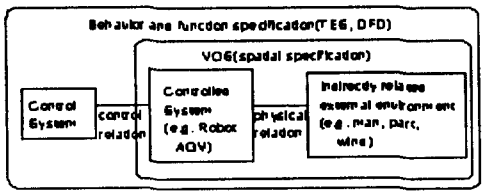


그림 2 제어 시스템, 물리 환경과 제어되는 시스템간의 상호 작용으로 나타나는 시스템의 행위

VOS는 제어되는 시스템이나 실세계 환경 객체들 사이의 물리적 관계나 제약조건들도 명세한다(예를 들면, Block의 위치는 일단 End-effector에 잡히면 그것의 위치를 따른다.).

모델링 과정과 시뮬레이션에 의한 유효성 검사

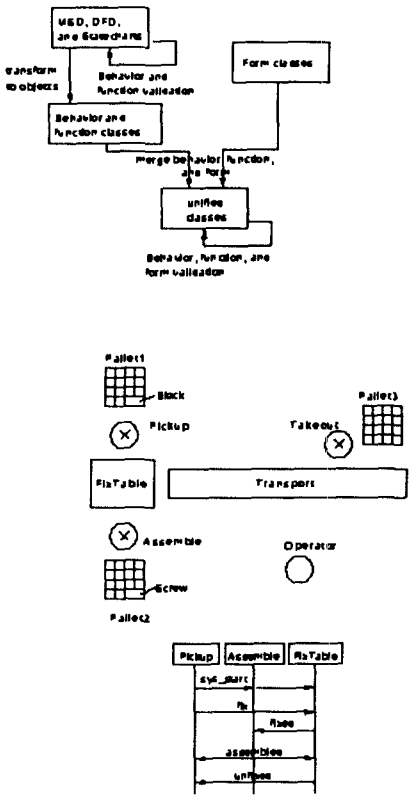


그림 3 한 개의 block 과 두 개의 screw 의 조립에 대한 생산 시스템의 설정

ASADAL에서 행위와 기능 명세들은 실제로 몇 장의 Message Sequence Diagram(MSD)을 그리는 것으로 시작된다. MSD는 시스템의 객체들 사이에 교환되는 데이타나 제어 신호들을 시간에 따라 나열함으로써 실시간 시스템의 외부 행동에 대한 전형적인 시나리오들을 기술한다(그림 3 참조). MSD는 처음부터 객체들을 찾을 수 있게 해 줌으로써 형태에 대한 고려와 함께 전체 모델링 과정에서 객체 지향성을 지원한다. 행위/기능 명세와 형태 디자인이 처음에는 각각 진행되다가 객체 단위로 나중에 결합되므로 이들이 객체들에 대한 일치된 견해를 보이는 것이 중요하다.

제어 시스템과 실세계 환경을 포함한 전체 시스템의 명세는 그래피컬한 도구인 TES와 DFD를 가지고 만든다. 실세계에 존재하는 객체들에 대한 형태의 명세는 VOS를 사용하여 병렬적으로 수행한다.1) 객체 지향적 환경을 만들기 위해서 TES를 클래스들의 병렬적인 행위들과 그와 연결된 DFD의 기능들로 나눈다. 이렇게 만든 클래스들 중 로봇같은 것들은 형태를 가질 것이고 제어 소프트웨어와 같은 클래스들은 형태를 갖지 않을 것이다. 그래서 로봇과 같이 행위, 기능, 형태를 모두 가진 클래스들은 행위와 기능 클래스들과 그 형태를 기술한 VOS 클래스들로부터 다중(Multiple) 상속함으로써 만들어진다. 실세계 객체들은 이러한 형태, 행위, 기능 클래스들로부터 설정 인자값(위치, 색깔)들을 새로 할당하고 객체화(Instantiation)함으로써 만들어지고, 형태가 없는 객체들은 행위와 기능 클래스들로부터 객체화된다. 명세가 어떻게 되는지에 대한 상세한 설명은 명세 방법에서 한다.

두 개의 시뮬레이션 루프(loop)가 수행되는데, 하나는 실시간 시스템의 행위와 기능 명세가 만들어 졌을 때, 다른 하나는 형태와 합쳐졌을 때 실제 환경의 가시화와 함께 이루어지며 상호 작용하는 시스템의 행위가 ASADAL[7]을 사용하여 관찰되고 분석될 수 있다. 이러한 명세와 시뮬레이션에 의한 유효성 검사 과정은 단계적인 방식으로 계속될 수 있다.

명세 방법

여기에서는 간단한 로봇을 이용한 생산 시스템을 예제로 사용하여 명세 방법에 대해 설명한다. 3.1절에서 중요한 시스템과 환경 객체를 인지하는데 이용되는 방

---

2 모양 자체에 대해서는 기하학적 디자인 도구를 사용하거나 주어진 도메인에 대한 라이브러리에서 정의되어 있는 모양 모델들중에서 선택함으로써 모델링한다.

법과 사용되는 요소들에 대해 설명하고, 그 후에 객체에 대한 상세화를 단계적인 방법으로 서로 다른 차원에서 수행하고 나중에 이들을 합치는데 그 과정에 이용되는 명세 방법을 3.2절과 3.3절에서 설명한다.

(ㄱ)시스템 명세

처음에 시스템이 수행해야 하는 ``두 부품을 조립하라"라는 기능에 대한 요구사항으로부터 시작하여, Block을 하나 가져와서 고정시키고 두 개의 Screw를 가져와서 조립한 다음 그것을 저장할 곳으로 수송하고 내려 놓는 세분화된 기능들로 명세한다. 시스템이 그러한 기능을 수행하기 위해서 어떻게 행동해야 할지에 대하여 고려함으로써 그림 3과 같은 초기의 설정을 만든다.

그 과정에서 객체는 컴퓨터에 의해 제어되어 활동적으로 일을 수행하는 로봇과 같은 객체와 활동적이지 않은 Screw와 같은 환경 객체로 구분된다.

실시간 시스템의 외부 행위에 대한 전형적인 시나리오를 기술하기 위해 사용되는 ASADAL에서 소개된 MSD를 시스템의 목적을 완수하기 위해 필요한 시스템 객체와 환경 객체를 더 명확하게 하기 위해 사용한다. 그림 3의 아래에 있는 MSD는 전형적인 시나리오를 가지고 객체 사이의 제어 신호의 흐름을 보여 준다. 예를 들어 그림 3의 MSD는 만약 Block을 FixTable에 옮긴 뒤에 Pickup이 FixTable에 fix 신호를 보내면, FixTable은 Block을 고정한 후에 fixed신호로 반응한다는 것을 보인다.

MSD를 사용하여 객체들 사이에 정보 교환에 대해 명확히 한 후에, 그림 4에 있는 Entity Relationship Diagram(ERD)을 사용하여 객체들 사이의 관계를 밝힌다.

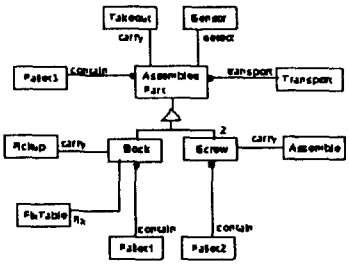


그림 4 객체들 사이의 관계와 관계되는 객체의 수(Connectivity)

ERD에서 사각형은 실세계 객체를 나타내고 객체들을 연결하는 선은 그들 사이의 관계를 보이며 삼각형에 연결된 선은 ``has-a" 관계로 연결된 객체들을 보이며 선 위의 숫자는 그 관계에 개입되는 객체의 수를 가리킨다. 예를 들면, Assembled Part는 한 개의 Block과 두 개의 Screw로 구성되고, Transport는 Assembled Part를 ``transport"한다. 이 정보들은 나중에 객체들 사이의 제약 조건들을 정의하는 데 이용된다.

시스템을 구성하는 객체들과 그들 사이에 흐르는 데이터 등으로 시스템의 설정을 명확히 한 후 각 객체에 대한 상세한 행위, 기능, 형태 명세를 시작할 수 있다. 그럼 다음에서 행위과 기능 명세에 대해 먼저 설명한다.

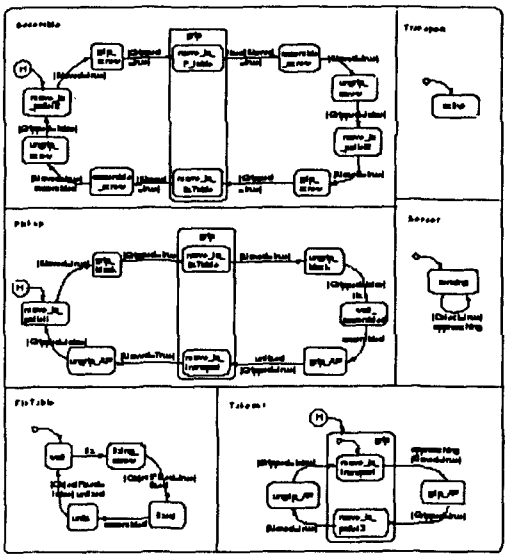


그림 6 생산 시스템의 TES 명세

(ㄴ)행위와 기능 명세

이제 ASADAL의 TES와 DFD를 사용하여 MSD의 각 객체의 기능 구조와 행위를 상세히 명세한다. 먼저, 각 객체의 행위를 따로 명세하지 않고 전체 시스템의 행위를 객체들의 동시 다발적인(Concurrent) 상태들로 명세한다. 그림 6의 TES는 예제 시스템의 행위를 보여준다. 예를 들면, 그림 6의 윗부분에서 보이는 Assemble의 행위는 여러 가지 사건들과 환경 조건들에 의해서 다른 객체들과 동기화되어서 pallet과 fix table사이에서 움직이는 상태, screw를 잡거나 놓는 상태, 조립 작업을 수행하는 상태 등을 가진다. 제어 시스템/제어되는 시스템과 환경 객체들 사이의 관계, 그리고 환경 객체들 사이의 관계는 물리적이므로 간접적인 환경 객체들의 행위는 TES를 사용하여 명세하지 않고, VOS에서 제약 조건으로서 표현될 것이다.

일단 행위 명세가 TES와 DFD를 사용하여 만들어지면, 명세의 논리적인 일치(Consistency)와 옳음(Correctness), 시간에 맞는 행위에 대한 분석을 할 수 있다. ASADAL/SIM은 DFD와 TES 명세들을 실행하고 데이타 흐름에 대한 통계학적 분석,

도달 가능성 분석, 비결정성에 대한 분석 등을 수행한다. 이러한 분석 결과는 (그림 11과 12 참조) 형태 명세를 수행하기 전에 행위와 기능 명세가 가진 문제점들을 발견하고 바르게 고치는 데 사용될 수 있다.

나중에 행위와 기능 명세들을 VOS라는 객체 지향적인 형태 명세와 결합하기 위해 TES와 DFD도 그림 9의 왼쪽 윗부분에서 보이는 것처럼 객체 지향적으로 변환해야 한다. 변환된 객체는 TES에 명세된 그것의 행위를 가지고, 어떤 일이 실행되어야 할 때 DFD에 있던 적당한 함수들을 부르며, 상태 전이에 의해 사건이 발생할 때 채널을 통해 신호를 보낸다. 그림 6의 TES에서 광역전파(Broadcast)되는 사건들은 그림 7에서처럼 객체들 사이의 동기화 채널로 할당된다. 할당된 통신 경로들은 동기화를 위하여 데이타나 제어 신호들을 교환하는 객체들 사이에 놓인다. 예를 들면, 그림 7의 Takeout의 명세를 보면 센서에 의해 발생하는 사건 approaching에 의해 move_to_transport에서 grip_AP상태로 전이하는데, 이 사건의 전달이 송신자인 Sensor객체와 수신자인 Takeout객체 사이의 채널로 표현된 것을 그림 7에서 볼 수 있다. 이와 같이 모든 다른 사건들의 전달도 적절히 객체들 사이의 채널로 표현된다.

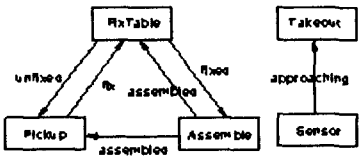


그림 7 객체들 사이의 동기화 채널들

형태가 없는 객체들은 행위와 기능 명세만으로 충분할 것이나 어떤 객체들은 행위와 기능뿐만 아니라 형태를 가진다. 이러한 객체들에 대해 형태는 VOS를 사용하여 명세하며 VOS는 객체들의 물리적인 특성들과 공간적인 행위들을 보인다. VOS

에 관한 세부 사항들은 다음 항에서 자세히 설명한다.

(ㄷ)VOS

형태 명세는 행위와 기능의 명세와는 독립적으로 진행된다. 예를 들면, 3.1절에서 설명된 시스템 명세로부터 그림 3의 생산 시스템 개요에서 객체들에 대한 대략의 모양/부피와 설정 정보를 잡아냄으로써 형태 명세를 시작한다. 행위나 기능처럼 형태도 VOS를 사용하여 단계적인 방식으로 점차 상세화한다. 다음 절에서 우리는 객체들과 객체들간의 물리적인 성질들, 반응 행위와 공간상의 제약 조건들에 대한 표현 능력 등의 관점에서 VOS에 대한 상세한 설명을 한다.

가. 물리적 성질들과 설정

VOS의 전형적인 속성(attribute)에는 모양, 재료, 위치, 속도, 가속도, 힘, 각속도, 각가속도, 토크 등이 있다(그림 9 참조). 속성 값들은 사용자로부터 주어지는 프로파일(Profile), 참조 테이블(Lookup Table), 미분 방적식 등 여러 가지 방법으로 표현된다.

VOS 지원 도구에 내장된 라이브러리에는 선가속도, 선속도, 각가속도, 각속도 등과 같은 전형적인 공간적인 움직임을 가진 객체들을 나타내는 메타 클래스들이 있다. 그림 8의 윗부분은 VOS 라이브러리의 여러 가지 메타 클래스들을 보여준다. 클래스들은 특정한 영역(Domain)이나 응용 시스템에서 사용되기 위해 *Domain*과 *Application*클래스들로 더 상세화된다.

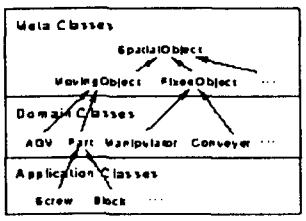


그림 8 VOS 클래스 계층

다음의 VOS 클래스들은 두 개의 메타 클래스 SpatialObject와 MovingObject, 그리고 도메인 클래스 AGV(Automated Guided Vehicle)를 보여 준다. 클래스 MovingObject에서 속도는 흐른 시간에 따라 가속도에 맞추어 새 값으로 바뀐다는 것을 보이고 있다(Eq velocity'...로 시작하는 줄 참조). 여기서 작은 따옴표가 붙은 것은 시간이나 계산이 수행된 후의 그 변수의 값을 의미한다. 이러한 시간 관련 방정식은 MovingObject를 상속한 클래스로부터 객체화된 객체의 위치를 시간과 속도에 따라 바꾸어 준다. 예를 들면, 아래에 명세된 AGV에 대해서 새로운 속도는 move메쏘드에 의해 설정되고, AGV의 위치는 시뮬레이션 동안에 시간이 지남에 따라 바뀌어진다.

```
Class SpatialObject
    Shape shape ;
    Coord location ;
EndClass
Class MovingObject
    Inherit SpatialObject ;
    Vect velocity ;
    Vect accel ;
    Vect angularVel ;
    Vect angularAccel ;
    ...
    Eq location' = location + velocity * time ;
```

```
    Eq velocity' = velocity + accel * time ;
    ...
EndClass
Class AGV
    Inherit MovingObject ;
    ...
    Method move (Vector newVelocity) :
        velocity' = newVelocity ;
    ...
EndClass
```

객체의 공간적 행위뿐만 아니라 부딛힘, 중력 등과 같이 객체들에 작용하는 공간적 관계와 그들이 따라야 할 물리 법칙들이 있으며 이를 표현하는 방법을 다음에서 설명한다.

나. 객체들 사이의 공간적 제약 조건들

실세계 객체들 사이에는 명확한 물리적 관계나 제약 조건들이 존재한다. 예를 들면, screw는 일단 end-effector에 의해 잡히면 그것의 위치를 따라가게 된다. 이같은 사항들은 VOS로 명세하는데, screw를 콘베이어 벨트에 붙이는(Attach) 것처럼 객체들이 서로의 공간적인 행위에 영향을 줄 수 있도록 객체들 간의 공간적 제약 조건들로 명세한다. 아래의 예제는 screw와 같은 움직이는 객체와 콘베이어 벨트와 같은 수송 도구 사이에 고려되어야 하는 제약 조건을 보여 준다. 그 제약 조건은 일단 움직이는 객체가 수송 도구 표면상에 있으면 그것의 속도는 수송 도구의 표면 속도와 같아진다는 것이다. 여기서 MovingObject는 MovingObject를조상으로 상속하는 아무 객체를 의미한다.

```
Class Transport

   Inherit FixedObject ;

   Vect surfaceVelocity ;

   Bool active ;

   Method on() { active' = True ; }

   Method off() { active' = False ; }

  Method off() { active' = False ; }
     Constraint On(MovingObject, Transport) and
        active=True ->
        MovingObject.velocity' =
              Transport.surfaceVelocity ;
  EndClass
```

앞에서 설명한 행위와 기능 클래스들과 VOS 클래스들은 세 가지 차원을 가진 통합 클래스들로 결합될 수 있다. 다음엔 어떻게 그것들이 통합 클래스들로 결합할 수 있는지를 보이고 어떻게 객체들이 시스템 모델을 형성하기 위해 클래스들로부터 객체화되는지 설명한다.

(ㄹ)행위, 기능, 형태를 가지는 통합 클래스

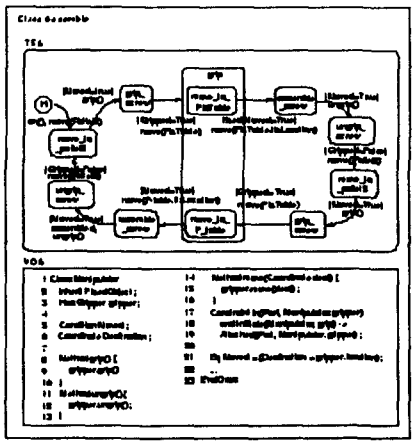


그림 9 행위, 기능, 형태를 명세하는 객체 모델

행위/기능 명세와 VOS를 만드는 것이 동시에 진행될 수 있지만, 가시화 시뮬레이션은 제어되는 시스템과 환경 객체들이 세 가지 차원의 명세를 모두 가지기를 요구한다. 그러므로, 제어되는 시스템과 환경 객체들에 대해 그림 9처럼 완전한 실시간 객체 표현을 만들기 위해 적절히 행위/기능 클래스와 VOS를 결합해야 한다. 우리는 이러한 통합 객체들을 행위/기능 클래스와 VOS 클래스를 다중 상속함으로써 만들 수 있다. 그러면 물리적 객체들은 새로운 통합 클래스들을 객체화함으로써 만들어지고, 제어 시스템처럼 형태가 없는 객체들은 행위/기능 클래스들로부터 객체화된다.

이런 방식으로 같은 행위를 갖지만 다른 물리적 성질을 갖거나 그 반대 경우의 클래스들을 쉽게 만들 수 있다. 예를 들어 Assemble, Pickup, Takeout 클래스들은 모두 팔과 End-Effector를 가지는 Manipulator VOS 클래스를 상속하지만 그림 6의 행위, 기능 명세에서 만들어진 클래스들로부터 서로 다른 행위와 기능들을 상속한다. Manipulator클래스는 물리적 세계의 Manipulator의 움직임을 보이는 move method를 갖는다. 객체들이 클래스의 객체화에 의해 만들어지며, 필요하다면 여러 개의 Takeout클래스 객체를 만들 수 있다는 것에 주목하라.

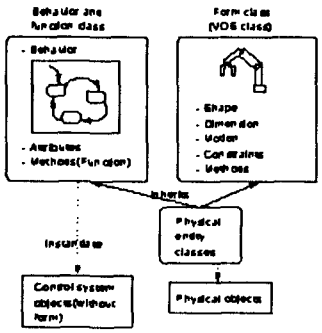


그림 10 물리적 클래스 Assemble

그림 10은 행위, 기능, 형태를 가진 Assemble클래스를 보여 준다. 이 클래스의

행위와 기능은 그림 6의 TES로부터 변환된 행위, 기능 객체 클래스로부터 상속하고, 그것의 형태 명세는 이미 정의된 Manipulator 클래스로부터 상속한다. Assemble은 VOS에 정의된 move(Pallet2) 메쏘드를 호출함으로써 Pallet2로 움직이고, grip()을 이용하여 screw를 잡는다. 움직여진 Screw의 상태는 VOS에 정의된 Moved의 조건 변수로 나타난다. 여기서 In, On, Above 등은 도구에서 제공하는 공간적 관계 조건들이라는 것에 주목하라.

시스템과 그것의 환경에 있는 모든 클래스들이 명세되면 시스템 모델을 만들어야 한다. 객체들은 객체화되면서 초기 설정과 같은 값들이 주어진다. 예를 들면, 그림 3에 있는 물리적 객체들은 위치, 방향 등에 대한 적절한 설정 인자 값들을 가진 Assemble, Transport, FixTable와 같은 통합 클래스로부터 객체화된다. 형태와 환경 객체 행위까지 고려함으로써 전체 시스템 명세의 가시화 시뮬레이션은 훨씬 더 현실적이 되고 개발 초기 단계부터 올바른 시스템 명세의 개발을 돕는다.

이러한 연구 흐름상에서 본 과제의 당해 년도 구체적인 연구 결과를 정리하면 다음과 같다.

1. 실시간 객체 시뮬레이션 모델 개발 방법론 및 도구 시제품 개발

기존의 ASADAL 실시간 시스템의 사용자 요구사항 명세 방법을 발전시켜 객체 지향적인 명세를 할 수 있게 했으며 이에 따라 명세된 Class의 여러 Instance들을 시스템 내에, 또는 다른 Class 내에 구성할 수 있게 되었다. 이러한 방법은 또한 객체 지향적인 디자인으로 쉽고 자연스럽게 나갈 수 있는 기틀을 마련한 것이다. 객체 지향 기술과 데이터 프로세싱 모델 및 상태 전이 모델을 결합시키고 있는 모델들은 여럿 있으나 ASADAL에서와 같이 명확하게 그들간의 관계를 명시하고 그들간

의 View를 분리(Separation)시켜 각각의 명세가 독립적으로 만들어 질 수 있게 하고 명세와 객체의 관계를 명확히 하고 상위 단계뿐 아니라 아주 상세한 명세까지 가능하게 하는 문법을 제공하는 방법은 없으며 또한 그 명세를 완전히 시뮬레이션할 수 있는 도구는 없다.

ASADAL 모델링 방법이 가지고 있는 완전히 새로운 아이디어는, 실시간 시스템의 모델링 및 시뮬레이션이 단순히 소프트웨어에 대한 명세와 시뮬레이션만으로 이루어질 수 없다는 것이다. 의미있고 효과적인 시뮬레이션이 되기 위해서는 실시간 소프트웨어와 관계를 갖는 실재하는 객체들에 대한 명세와 그들에 대한 시뮬레이션이 반드시 병행되어야 한다는 것이다. 그리고 많은 경우 이들은 공간적인 모델(Form;폼)을 가지기 마련이며 또 많은 경우 이것은 동적으로 변하게 된다. 따라서, ASADAL의 한 확장형인 ASADAL/FORM은 시스템에 존재하는 객체들의 행위, 기능뿐 아니라 폼(Form)도 모델링 할 수 있게 하고 있다. 이를 위해 폼을 모델링할 수 있는 저작도구 및 객체들의 움직임과 그들간의 공간적인 관계들을 표현하기 위한 Rule등을 묘사할 수 있는 언어인 VOS(Visual Object Specification)를 만들었다. 또한 행위, 기능의 시뮬레이션뿐이 아닌 이 Form도 같이 시뮬레이션하는 방법을 개발하였다.

결국, 이러한 방법을 사용하면 사용자는 도메인에 존재하는 객체들에 대한 명세를 기존에 존재하는 객체들을 Compose하고 이들을 다루기 위한 Behavior, Function을 첨가함으로써 만들어 낼 수 있으며 이 객체가 또 다른 객체를 만들기 위해 쓰일 수 있게 된다. 예를 들어 자동화 공장의 경우 먼저 로봇 도메인에 존재하는(없으면 새로 만들어 도메인에 첨가할 수 있다) Joint들과 Link들을 Visual하게 연결하고 이들이 가지고 있는 기능과 행위들을 이용하여 요구되는 수준 높은 서비스를 제공하는 로봇을 만들어 낼 수 있고, 이 로봇의 여러 객체들을 Instantiate하여 공장을 만들 수

있을 것이다. 이렇게 간단히 만들어진 모델은 바로 시뮬레이션되므로 공장이 어떤 식으로 동작하는 지, 문제는 발생하지 않는지, 로봇의 개수는 적당한지가 바로 검증될 수 있고 필요에 따라 공장의 구조를 바꾸거나 로봇의 개수를 바꾸는 등의 작업을 하고 또 다시 시뮬레이션을 해 보는 등의 일들이 가능하다.

새로이 개발중인 ASADAL은 이러한 변화를 모두 수용하고 있다. ASADAL은 개발의 편의성 및 사용자 편의성을 위해 Platform에 관계 없고 풍부한 라이브러리들을 제공하는 Java 언어로 만들어졌다.

첫 연도이므로 연구 결과가 모두 도구화되지는 않았으나 '시제품이 완성되고 예제 시스템에 적용하여 가능성이 검증되었다. 현재 객체 모델링 도구가 만들어졌고 행위, 기능 명세 도구가 만들어졌으며 행위, 기능에 대한 시뮬레이터는 완성되었다.

시제품에는 실시간 시뮬레이터의 사용자를 위한 VR 인터페이스가 들어 있으며 VR 인터페이스를 Stereoglasses등을 이용하여 실제 환경처럼 입체로 보이고, 사용자가 실제처럼 동작해 볼 수 있게 하는 기술을 개발되어 적용하고 있는 중이다.

2. 가시화 방법론 개발

ASADAL/PROTO는 객체들의 폼에 대한 명세를 포함하고 있으므로 별도의 가시화도구가 필요 없이 자연스럽게 객체들의 실제 모습 및 공간적인 행위를 명세하고 이를 시뮬레이션을 통해 가시화하는 방법을 택하였다. 예를 들어 Robot이 물건을 집어올리는 경우 주어진 시나리오대로 Robot의 움직임과 물건을 Synchronize하는 등의 방법을 사용하는 것이 아닌, Robot의 Gripper가 물건을 집으면(물론 "집는다"는 것도 명세되어야 한다) 물건은 Gripper가 움직이는 대로 따라가게 된다는 식의 규칙을 정함으로써 시뮬레이션을 통해 가시화하게 된다. 따라서 예기치 않았던

(그러나 실제적인) 공간적인 행위가 가시화될 수 있다.

이를 위해 우선 기존의 가시화 도구들을 고찰, Rule 및 Constraint Network등을 통한 표현 방법들을 보았으며 이들이 모두 우리가 연구하고 있는 수준이 아닌 아주 낮은 수준의, 프로그램에서 갓 벗어난 수준의 방법들을 제공하고 있다고 판단, 목표 환경을 Java를 이용한 다중 환경으로 잡고 가시화 라이브러리를 선정, 이용하여 개발을 시작하였다.

먼저 공간 정보를 명세하는 방법을 개발하였고 시제품에 적용, 행위와 기능의 시뮬레이터와 연동하여 시뮬레이션이 가능함을 검증하였다.

3. 실시간 시뮬레이터 소프트웨어의 의 자동 생성 방법 개발

명세를 시뮬레이션함으로써 논리적으로 검증된 후 실제로 실시간 시뮬레이터를 만들어야 한다. 이를 위해 ASADAL 명세로부터 실시간 시뮬레이터의 객체를 자동 생성하는 방법을 개발하였다. 이 방법은 일단 명세를 이용하여 우리가 새로 정의한 객체 지향적인 중간 코드를 자동 생성하고 이를 다시 TMO나 Java, C++등으로 필요에 따라 Translation함으로써 원하는 실시간 시뮬레이터를 만들어 낼 수 있게 된다.

UCI 위탁 과제의 연구 내용:

(1) Real-time simulation을 효과적으로 지원하는 근년에 개발된 객체지향 model인 TMO를 한단계 더 보강 하였다. 보강된 부분은 programmable data field channel (DFC)를 삽입한데 있다.

(2) TMO 의 execution engine을 Windows NT 바탕으로 두가지 개발 하였으며 건국대 와 외국어대 의 TMO execution engine 개발을 보좌였다.

(3) 건국대의 TMO형 전력 송출 system simulator 개발을 보좌 하였다.

(4) TMO형 real-time simulator 구축 기법의 다른 측면에서의 demo 로서 고속도로 및 도시교통망의 미시적 real-time simulator를 design 하였다.

나. 연구 결과

a. 실시간 객체 시뮬레이션 모델 개발 방법론 및 모델링 도구 시제품

b. 공간 정보를 포함한 실시간 시스템 명세 방법론 및 명세 도구 - 행위, 기능 명세 도구 완성, 폼 도구 시제품

c. 시뮬레이터 운영자 및 사용자를 위한 VR 인터페이스의 개발 방법 개발 및 시제품

d. 공간정보(Form) 명세 방법 및 명세 도구 시제품

e. 가시화 도구 시제품

f. 목표 시스템의 시제품

여러 목표 시스템들의 행위, 기능 명세 및 시뮬레이터

산업체가 요구한 목표 시스템의 3차원 실시간 시뮬레이터 인터페이스

산업체가 요구한 목표 시스템의 아이콘을 이용한 모델링 도구

g. 명세로부터 실시간 시뮬레이터의 자동 생성 방법

명세로부터 코드 자동생성을 위한 프로세스

중간 코드의 문법(Syntax) 및 의미(Semantics)

다음은 개발된 행위, 기능, 폼 모델링 및 시뮬레이션 도구이다.

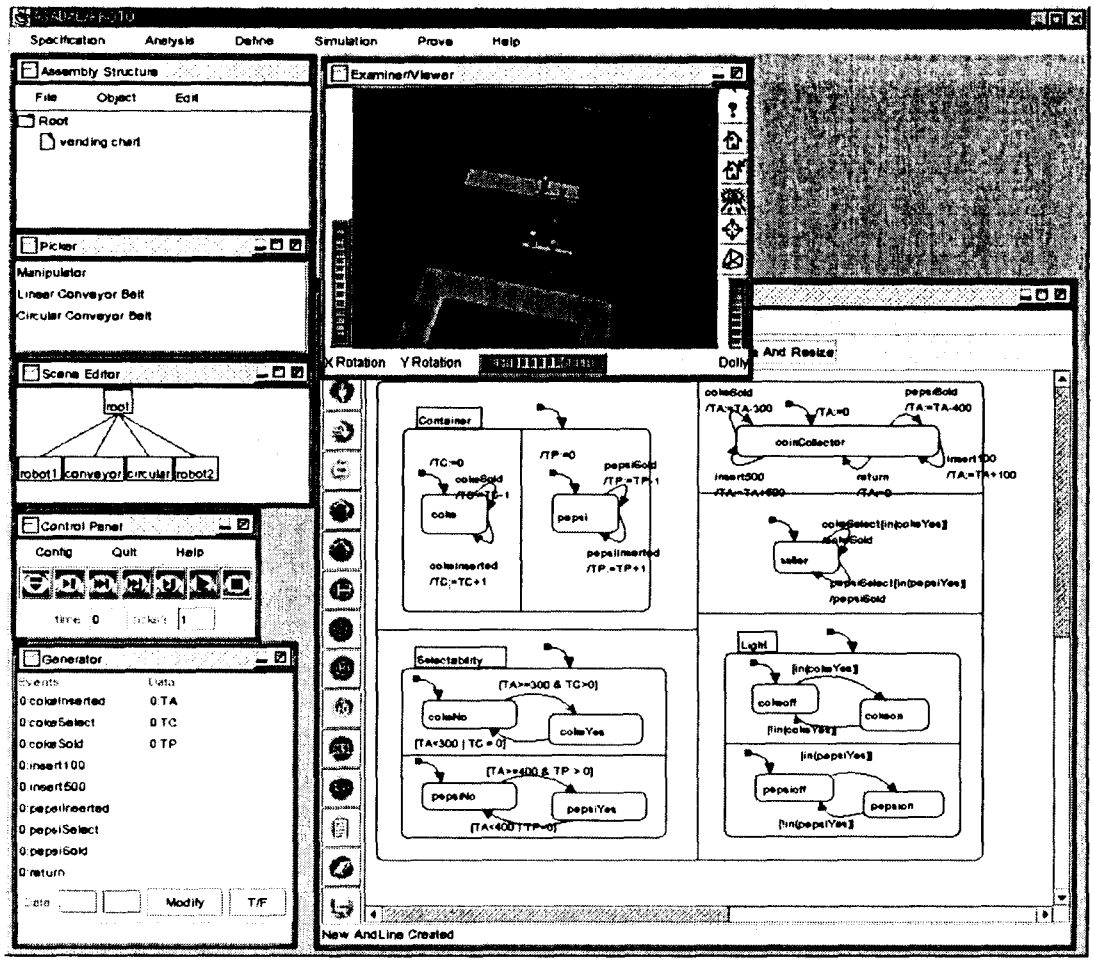


그림 11 ASADAL/PROTO

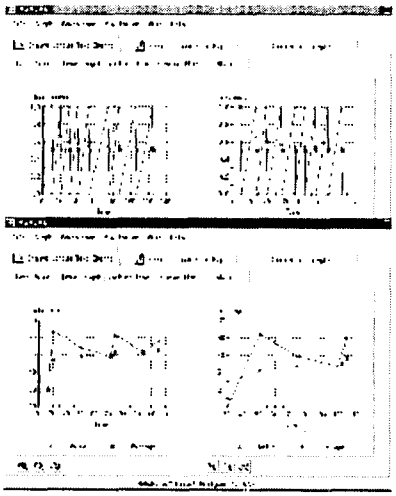


그림 12 ASADAL 데이터 분석기

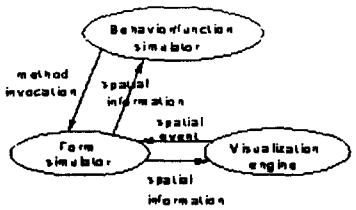


그림 13 시뮬레이션 시스템의 구조

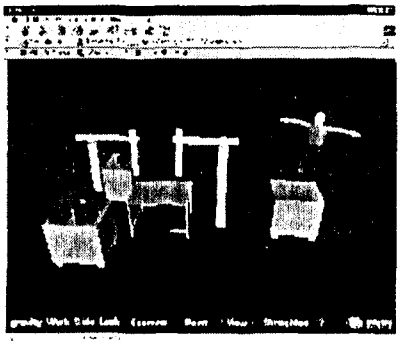


그림 14 가시화 도구 시제품

행위와 기능 명세의 시뮬레이션 환경은 완성되었으며 위의 그림은 ASADAL/PROTO와 분석 도구를 보여 준다.

ASADAL/PROTO는 ASADAL/SIM에 형태 표현과 물리적 시뮬레이션을 위한 명세 방법과 도구를 더해 확장한 것이다. 그림 13은 행위와 기능 시뮬레이터, 형태 시뮬레이터, 가시화 엔진을 가지는 ASADAL/PROTO의 구조를 보인다. 전체 구현이 1998년 초에 유효하게 될 Java3D에 기반하고 있지만 그것의 초기 프로토타입은 VRML(Virtual Reality Markup Language)과 자바 기술에 기반하여 구현되었다. 생산 공장 예제의 가시화 시뮬레이션에 대한 화면이 그림 14에 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 사람이 테이블에 부딪히면 Pickup Manipulator는 조립을 실패하게 된다.

4. UCI 위탁 과제의 연구 내용 및 결과

UCI 위탁 연구과제의 당해 년도 연구 개발 목표는 다음과 같다.

* RTS modeling 기법을 실시간 객체지향형으로 더욱 공고히 확립

* 대규모 TMO형 real-time simulator를 더욱 효율적으로 신속히 execute 할 수 있는 분산 병렬 처리 engine 구축

UCI 위탁 연구 과제의 연구 결과는 다음과 같다.

(1) TMO 에 programmable data field channel (DFC) 를 삽입 보강한 것은 자연스럽고 효율적으로 multicast facility를 활용하게 하여 Real-time simulation 의 효율을 증대시키는 효과가 있다.

(2) TMO 의 execution engine을 Windows NT 바탕으로 두가지 개발 하였는 바 하나는 CORBA communication 환경에 적합한 것이며 현재 동작하고 있고 다른 하나는 NT 의 UDP communication facility를 활용하는 것으로서 수행 속도가 보다 빠른데 현재 final validation 중이다. 건국대 와 외국어대 의 TMO execution engine 개발을 보좌하였고 이는 현재 동작하고 있다..

(3) 건국대의 TMO형 전력 송출 system simulator 개발을 보좌 하였다.

(4) TMO형 real-time simulator 구축 기법의 다른 측면에서의 demo 로서 고속도로 및 도시교통망의 미시적 real-time simulator를 design 하였다. .2차년도

중반쯤 구현이 완성 될 것이다.

제 2 절 2차년도 연구개발수행 내용 및 결과

1. 연구개발 목표

당해 년도의 연구 목표는 공간 정보를 포함한 객체 지향적인 실시간 시스템 명세 기술 및 그 명세의 효율적인 가시화 기술 개발 이다.

실시간 시스템의 개발 초기 사용자 요구 사항의 명세 단계에서 상세 설계, 구현의 전 단계에 거쳐 사용할 수 있는 객체 지향적인 실시간 시스템 모델을 만들고 이 모델을 시뮬레이션할 수 있는 시뮬레이터를 만든다. 이를 위해 요구 사항 명세의 시뮬레이터를 만들고 상세 설계 모델의 실시간 시뮬레이터를 만들고, 요구 사항 명세로부터의 자연스러운 설계로의 전환을 위한 자동 변환기도 만든다.

실시간 시뮬레이션 디스플레이를 위해, 멀티미디어 및VR(Virtual Reality)에 기반하여 실시간 시뮬레이션 실행 엔진과의 사용자 인터페이스를 개발한다. 이러한 것들을 기반으로 하여 중요한 응용 분야에 대한 실시간 시뮬레이션 모델을 구성하여 실제 사용 가능한 실시간 시뮬레이터를 구축함으로써, 실시간 시뮬레이터 프로토타입에 대한 방법론을 정립한다. 실시간 시스템은 특히 외부 세계와 복잡하게 연결되어 있으므로 내부적 관점에서의 명세나 설계만으로는 제대로 된 시뮬레이션을 해 내기 힘들기 때문에 각 모델들은 여러 실세계에서 일어날 수 있는 현상들을 모델링할 수 있어야 한다. 이들 중에 특히 공간적인 정보들은 복잡한 물리 현상들과 관련되어 표현하기 쉽지 않으므로 가능하면 이 정보들을 간단히 모델링할 수 있게

한다. 그리고 이렇게 표현된 공간 정보를 이용하여 3차원으로 실제 세계에서 일어나는 일들을 시뮬레이터 사용자에게 가시화하여 보여줌으로써 명세나 시뮬레이션 결과를 한눈에 알아볼 수 있게 한다. 또한, 실제 상황과 같게 만드는 VR 인터페이스는 사용자들이 개발할 시스템을 초기부터 사용해 보게 함으로써 요구사항을 조기에 알아낼 수 있다는 장점이 있고 나아가 사용자들을 훈련시키는 데에도 사용할 수 있다.[Lee98], [Kang98b]

당해 년도에는 다음과 같은 연구 개발 목표들이 설정되어 있다.

1. ASADAL 명세로부터 설계를 거쳐 최종적으로 TMO객체를 자동으로 생성해 내는 생성기 개발 (Automatic Code Generator)

   - ASADAL 명세로부터 자연스럽게 객체 지향적인 설계를 할 수 있는 설계 도구를 만든다음 ASADAL 명세와 설계 결과로부터 1차년도에 정의한 중간 코드를 자동 생성하는 생성기를 만든다.

   - 생성시 요구되는 Library들도 중간 코드의 형태로 만든다.

2. 가시화 도구 및 VR(Virtual Reality) 인터페이스 저작도구 개발

   - 1차년도에 연구, 개발된 폼 모델링 언어를 모델링할 수 있는 폼 모델러와 작성된 폼 모델의 자동 해석을 통한 가시화 시뮬레이션 도구를 만들어 행위, 기능의 시뮬레이터와 연동되도록 한다.

   - 실시간 시뮬레이터의 사용자에게 실제와 같은 영상을 제공하는 VR 인터페이스를 저작할 수 있는 저작 도구를 만든다.

3. 실제 시스템 시뮬레이터의 시제품 개발

- 목표 시스템의 명세를 완성하고 TMO 프로그램을 자동 생성한다.

4. 전력 시스템 모델링 도구 개발

- 목표 시스템에 대한 시뮬레이터 모델링 도구를 개발한다.

2. 연구 수행의 이론적, 실험적 접근 방법

연 구 범 위	연구수행방법 (이론적·실험적 접근방법)	구체적인 내용
ASADAL 명세로부터 설계를 거쳐 최종적으로 TMO객체를 자동으로 생성해 내는 생성기 개발 (Automatic Code Generator)	객체 지향적인 접근 방법 (Object-Oriented Development Method)	코드 생성기는 실제 코드를 작성하는 만들어 내는 부분과, 생성된 코드가 사용하는 라이브러리의 두 가지 부분으로 구현되어 있다. 이 두부분 모두 Java, C++를 각각 이용하여 객체 지향적인 디자인과 접근 방법을 취하였다.
	사건 기반의 아키텍처 (Event-Based Architecture) 를 통한 접근 방법	생성된 코드와 사용자 정의 코드와의 연결을 위하여 사건 기반의 아키텍처를 참조하여, 사용자가 생성된 코드의 구조와 무관하게 손쉽게 프로그램 작성이 가능하도록 하였다.

	점진적인 시스템 개발 방법	코드 생성기는 코드 생성 전에, 실제로 어떤 코드가 생성되어야 하는 가를 모든 경우에 대해서 테스트를 완료하였다. 이 테스트를 할 때, 점진적으로 상황의 복잡성을 증가시키면서 테스트를 진행하였다.
가시화 도구 및 VR(Virtual Reality) 인터페이스 저작도구 개발	객체 지향적인 접근 방법 (Object-Oriented Development Method)	가시화 도구 및 VR 인터페이스 저작도구는 실세계에 존재하는 객체들의 형태, 위치, 방향, 공간적인 관계 등을 명세할 수 있게 되어 있다. 이 때 객체 지향적인 접근 방법에서 사용하고 있는 객체들간의 관계인 상속(Inheritance), 모음(Aggregation), 조립(Assembly) 등을 그대로 정의하여 이용할 수 있게 하였다.
	인공 지능의 자동 논리 추론 방식	인공 지능에서 사용하는 논리 언어인 Prolog를 기반으로 한 논리 추린 엔진을 탑재, 공간상에 존재하는 객체들간의 관계 및 관계에 따른 공간적인 행위를 자동 추론하여 시뮬레이션에 반영하게 하였다.

	공간에 대한 추상적인 모델	공간상에 존재하는 객체들에 대한 공통적인 모델을 시스템에 만들어 놓고, 또 사용자가 필요에 따라 정의할 수 있게 함으로써 가시화 도구 및 VR 인터페이스를 빠르게 만들 수 있도록 하였다.
실제 시스템 시뮬레이터의 시제품 개발	또아리 (Spiral) 방식의 접근 방법	실제 시스템의 시뮬레이터를 간단한 것에서부터 점차로 기능을 붙여가며 계속적으로 개발, 자동 코드 생성기를 테스트하는 데 사용하였고 최종적으로 시연품으로서 사용하였다.
전력 시스템 모델링 도구 개발	점진적인 시스템 개발 방법	모델링 도구는 점진적으로 기술적으로 어렵거나 성능을 크게 좌우하는 곳을 먼저 개발해 나갔으며 계속적으로 만들어진 부분들에 대한 부분 테스트, 그리고 다른 곳과의 연동성 테스트를 병행하여 개발해 나갔다. 또한, 테스트에는 실제 사용할 전력 연구원의 연구원들이 참여, 도구의 유효성을 최대화 하였다.

3. 연구개발 결과

가. ASADAL 명세로부터 설계를 거쳐 최종적으로 TMO 객체를 자동으로 생성해 내는 생성기 개발

이 부분은 ASADAL 명세로부터 TMO 모델을 설계하도록 유도하고 설계된 모델을 바탕으로 TMO기반의 실행 가능한 코드를 자동으로 생성하는 코드 생성기에 대한 설명이다.

연구의 목표

복잡성이 높은 실시간 시스템은 요구 분석 단계에서부터 그 행위나 기능을 엄밀히 검증하는 일이 매우 중요하다. 이를 위해서 ASADAL 에서는 실시간 요구 분석 단계 언어인 DFD, Statecharts 등과, 이들간의 연결을 제시하여 사용자가 개발하고자 하는 시스템을 명세할 수 있는 방법을 제공한다. 이렇게 명세된 결과물은 실행 가능한 것으로서, 사용자는 중요한 행위 성질을 증명할 수도 있고, 시스템의 행위를 시뮬레이션해 볼 수도 있다. [Kang98c] 그러나, 이렇게 명세를 완벽히 수행했다고 하더라도, 그 이후 개발 과정인 디자인이나 코딩 단계에서 이렇게 증명된 명세대로 프로그램을 작성할 수 있는가는 별개의 문제이며 매우 어려운 문제이기도 하다. 이 연구는 아사달로 명세된 결과를 바탕으로 실행 가능한 코드를 바로 생성해 낸다면 명세 단계에서 검증된 내용이 코드안에서도 자동적으로 보장될 뿐 아니라, 개발 시간이 매우 단축시킬 수 있다는데 착안하여 출발하였다. 그러나, 이렇게 바로 코드 생성이 이루어질 경우, 성능 향상이나 재사용성을 위해서 사용자가 수행하는 지적인 과정인 디자인 과정이 생략되어 쓸모 없는 코드가 생성될 확률이 매우 높다. 따라서, 본 연구에서는 실시간 시스템의 디자인 모델인 TMO로 사용자가 아사달로 명세한 결과물이 맵핑될 수 있는 관계를 제공하고, 이 결과를 바탕으로 코드 생성을

하는 것을 그 목표로 하였다.

디자인 모델 구축

디자인 모델은 실시간 객체 지향 방법은 TMO를 기반으로 하였다. 한 객체는 데이터와 메쏘드 그리고 행위를 제어하는 스테이트로 구성되어 있다고 생각할 수 있다. 여기에서 TMO가 덧붙여 가지는 성질은 메쏘드가 일정 주기로 수행되는 Spontaneous Method(이하 SpM)와 외부 요청에 의해 동작하는 Service Method(이하 SvM)으로 구성된다는 것이다. 명세된 결과물에서 DFD의 프로세스는 메쏘드로 연결되고, DFD의 데이터 스토어 및 Statecharts의 임시 데이터들은 객체의 데이터로 연결이 된다. DFD의 프로세스의 수행 관계를 컨트롤하는 Statecharts는 객체의 행위를 제어하는 컨트럴 역할을 하게 된다. 아래에서는 프로세스, Statecharts, 데이터가 각각 어떻게 연결될 수 있는가를 자세히 밝혔다.

a. 프로세스

아사달 명세 결과물과의 연결에서 있어서 가장 중요한 역할을 하는 것은 DFD의 프로세스들이다. 사용자는 원하는 객체를 생성하고, 이 객체에 DFD의 프로세스의 일부를 할당한다. 할당된 프로세스는 해당 객체의 메쏘드가 된다. 이 메쏘드들은 그 특성에 따라서, SpM이나 SvM으로 사용자가 결정을 한다. SpM이나 SvM을 결정하는 가이드라인은, 해당 프로세스의 활동 성향이다. 만일 해당 프로세스가 매우 빈번히 사용되는 것이고, 일정 주기를 가지고 행동한다면 SpM으로 할당하는 것이 바람직할 것이고, 해당 프로세스가 실행 주기가 일정치 하고 빈번하지 않게 실행 된다면 SvM으로 할당되는 것이 바람직하다. 프로세스의 주기성은 Statecharts에서 나타

나기도 한다. 스테이트 명세에서 어떤 프로세스의 실행을 주관하는데, 이 때 주기를 할당할 수 있고, 혹은 "tm(time out)" 이벤트를 사용하여 프로세스의 주기를 함축적으로 표현할 수 있다. 이렇게 명시적으로 주기성이 명세에 포함된 프로세스의 경우에는 SpM으로 할당하는 것이 성능을 향상시킬 수 있다. 이 때 주의 해야 하는 것은 SpM의 주기를 설정하는 것이다. 만일 해당 프로세스가 서로 다른 스테이트 명세에서 서로 다른 주기로서 실행이 표현되어 있다면, 이들 주기의 최소 공약수의 크기로 주기가 할당되어야지 바른 실행 결과를 기대할 수 있다. 그렇지 않은 경우에는 실행이 기대되는 순간에 실행하지 않을 수 있음을 명심해야 한다.

b. Statecharts

Statecharts는 사용자가 직접 객체의 할당할 필요가 없다. 데이터의 프로세스를 각 객체에 할당하고 나면, 전체 Statecharts에서 해당 프로세스를 컨트롤 하는 부분을 골라내어 해당 객체에 자동적으로 연결을 시켜준다. 이를 위해서 명세된 Statecharts 모델을 재해석하여야 하는데, 그 단계는 아래와 같다.

■ DFD 연결 관계를 통해 Statecharts의 부모-자식 관계 설정

원래 아사달 명세 결과물은 DFD의 분할로 모델이 확장되어 가고, Statecharts는 각 DFD에 대한 컨트롤러의 작용을 할 뿐, Statecharts각각의 부모-자식 관계를 표현되어 나가지는 않는다. 그러나, 부모-자식 관계가 DFD의 분할을 통해 함축되어 있다. 만일 어떤 DFD D가 있고 이 D가 컨트롤러 S를 가지고 있고, D안에 프로세스 A가 있다고 하자. 그리고, 이 프로세스 A가 분할되어 그 안에서 다시 컨트롤러 S1을 가지고 있다고 하자. 이 경우, S와 S1은

실제로 모델링 결과물 상에서 눈에 보이는 부모-자식 관계를 형성하고 있지는 않지만, S에서 프로세스 A를 실행시키는 스테이트가 S1의 부모 스테이트가 된다. 프로세스 A의 컨트롤러 S1은 프로세스 A가 실행중일 때만 활성화되게 되는데, 곧 이 의미가 S에서 프로세스 A를 실행시키는 스테이트가 활성화될 때만 S1이 활성화된다는 의미이므로 부모-자식 관계가 형성되는 것이다. 이런 연결을 쫓아가다 보면, 명세 결과물에서 표현된 모든 Statecharts가 하나의 트리로 표현되게 된다.

■ 프로세스의 실행 여부를 주관하는 스테이트를 최하위 노드로 변경

모든 Statecharts를 하나의 트리로 표현하고 나면, 프로세스의 실행과 연결된 스테이트를 최하위 노드로 변경하는 작업을 한다. 이렇게 하는 이유는 나중에, 몇개의 프로세스들과 연관된 Statecharts의 그룹핑을 할 때 중복성이 생길 위험을 제거하기 위함이다. 만일 어떤 프로세스 A, B가 있는데, A와 연결된 스테이트가 B와 연결된 스테이트의 부모 스테이트라면, 서브 트리를 구성하기가 어려워진다. 이 경우에, A에 AND관계의 2개의 자식 노드를 만들어, 하나의 노드에 프로세스를 할당하고 또 하나의 노드에 기존의 하부 트리를 연결하는 방식으로 처리한다. 이 과정에서 덧붙여 하는 일은 프로세스 스펙에 있는 컨디션 조건을 스테이트 트렌지션 레벨로 끌어 올리는 일이다. 이렇게 함으로써, 프로세스는 연속적으로 수행하는 것이 아니라, 해당 조건이 맞을 때만 한번씩 수행하도록 효율적인 제어를 할 수가 있다.

■ 선택된 프로세스를 제어하는 하위 트리 추출

위의 과정이 끝난 후, 객체와 객체와 연결된 프로세스 정보가 추출이 되면, 각 프로세스들을 제어하는 하위 트리를 추출해 낸다. 만일 이 하위트리가 연결된 형태가 아니면 별도의 루트노드를 만들어, 하나의 트리로 재구성한

다. 이 과정에서 하위 트리로 떼어낸 부분과 그 상위 스테이트간에 이벤트를 할당하는데, 이 이벤트는 객체 자체의 활성화/비활성화를 결정하게 된다.

이러한 일련의 과정을 거치고 나면, 본래의 Statecharts의 루트로부터 시작하는 상위 레벨 노드들은 할당이 되지 않은 채로 남게 된다. 이 Statecharts의 남는 부분은 새로운 객체를 생성하여 할당하거나 기존의 객체에 할당시켜야 한다. 이 가장 상위 레벨 Statecharts가 할당된 객체는 다른 모든 객체들의 시작 시점을 결정하는 객체가 된다. 이 객체는 다른 객체가 시작 준비 상태일 때 시작하여(즉, 가장 마지막에 활성화되어) 다른 객체의 시작을 제어하도록 하여야 한다.

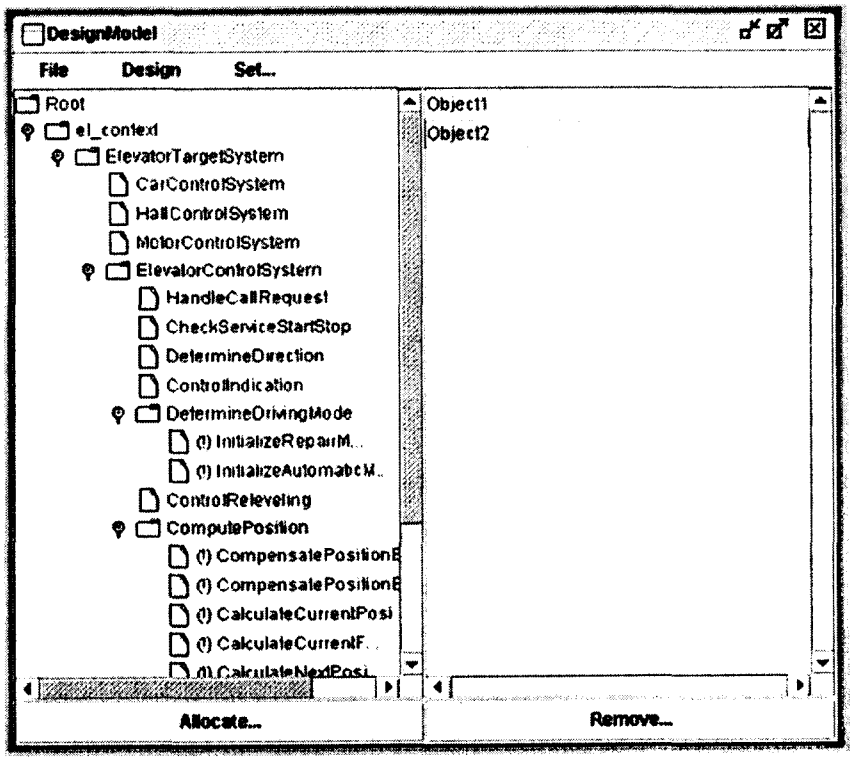
이런 과정의 장점은, 사용자는 그룹핑이 편리한 프로세스를 중심으로 객체를 구성하기만 하면, 객체의 행위 모델이 자동적으로 연결되어 복잡성이 줄어든다는 데 있다. 그러나, 단점은 객체를 구성할 때, 기능적인 측면에만 너무 천착하고, 행위적인 독립성은 염두에 두지 않을 수 있다는데 있다. 그러나, 자동 생성된 코드는 그 결과를 자동으로 분석할 수 있는 잠재력을 가지고 있으므로, 행위적인 특성(실행 주기, 데이터 통신량)은 차츰 디자인 결과에 다시 반영할 수 있을 것이다.

c. 데이타

데이타는 프로세스와 마찬가지로 사용자가 객체로 할당을 해 주어야 한다. 객체간의 데이터 교환은 TMO의 채널을 사용하여야 하고, 객체 안에서의 데이터 사용은 단순 함수 부름이라고 생각하면, 최대한 데이타의 통신량을 줄이는 것이 성능 향상의 주요 요인이 된다는 것을 알 수 있다. 따라서 데이터를 할당할 때는 객체간

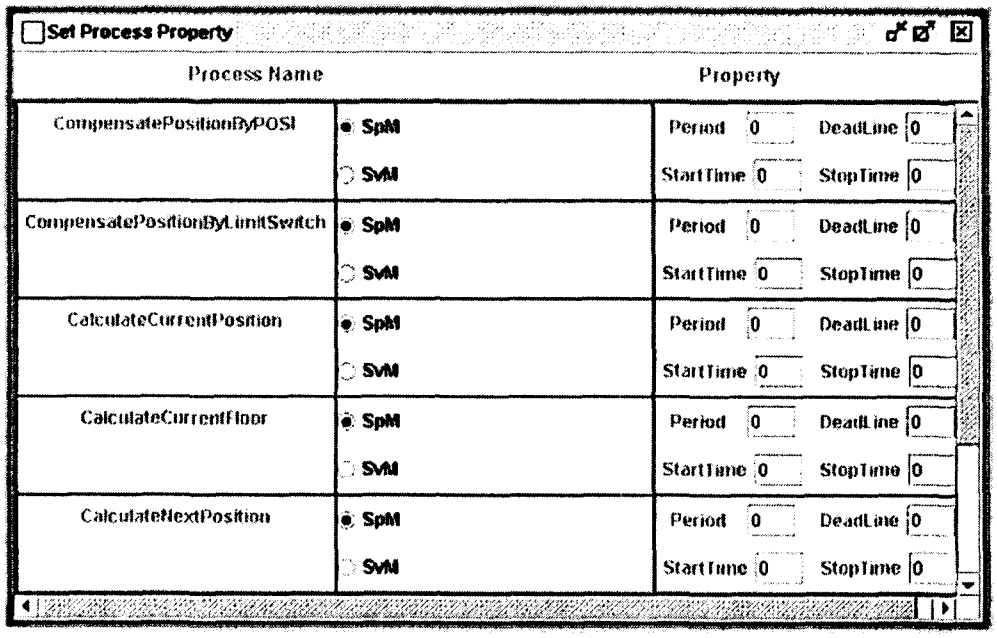
의 움직여야 하는 데이터량을 줄이는 방향으로 진행되어야 한다.

이러한 일련의 과정을 지원하는 툴을 개발하였다. 이 툴은 사용자가 원하는 상위레벨 Context Diagram을 로드하면, 이하 하위 레벨 DFD들과 관련된 Statecharts, Data들을 해석하여 정보를 구성하고, DFD의 프로세스들을 트리 형태로 보여주어, 객체를 생성하고 할당할 수 있도록 해 준다.



위 그림은 아사달에 포함되어 있는 TMO 디자인 모델을 지원하는 툴의 메인 윈도우 화면이다. 왼쪽에 보이는 것이 DFD의 프로세스를 트리 형태로 보여주는 것이고, 오른쪽에 보이는 것이 생성된 객체들의 리스트이다. 원하는 프로세스를 선택하고 밑에 있는 "Allocate"라는 버튼을 누르면 선택된 노드의 하위 프로세스들이 모두 객체의 메쏘드로 할당이된다. 이미 할당된 프로세스에 대해서는 (!) 표시를 해

주어, 사용자가 프로세스를 중복하여 할당하는 실수를 미연에 방지하도록 하였다. 또한 이러한 디자인 모델을 다 만들고 난 후에, 할당이 안 된 프로세스가 있을 경우 에러를 발생하고, 중복 할당이 있을 경우에도 에러를 발생한다.



프로세스를 객체에 할당하면 앞서 언급했듯이 그 프로세스가 어떤 특성의 메쏘드로 할당될 것인지가 결정되어야 한다. 위의 그림은 프로세스의 특성을 결정하도록 돕는 윈도우로서 사용자는 SpM, SvM중 속성을 결정하게 되고, 결정된 사항에 따라서, SpM인 경우는 Period, Deadline, SvM인 경우는 Deadline을 결정한다. 필요에 의해서 Start time과 Stop time(SpM인 경우)을 결정할 수도 있고, 결정하지 않을 경우는 자동적으로 0과 FOREVER로 셋팅이 된다.

코드 생성을 위한 최적화 단계

디자인 모델을 만드는 단계가 끝나면, 코드 생성을 위해서 내부적으로 최적화 단계를 거친다. 아래에서는 최적화 단계 중에 일어나는 여러가지 사항을 정리한 것이다.

a. 생성-소비 관계 파악

각 개체간에 데이터, 이벤트 통신을 알기 위해서 모든 데이터와 이벤트에 대해서 생성-소비 관계를 파악한다. 만일 이 관계를 파악하지 않고, 브로드캐스트 방식을 채택하게 되면, 신뢰성이 떨어지고, 필요없는 데이터나 이벤트를 계속 처리해야 하므로 효율이 떨어진다. 따라서 코드 생성 전에 데이터와 이벤트 전달 관계를 파악함으로써 각 객체간의 통신의 효율을 극대화시킨다.

b. 컨디션 커넥터 제거

컨디션 커넥터는 사용자의 편의를 의해서 만들어진 것으로, 사실상 필요없는 스테이트이다. 이러한 컨디션 커넥터들은 동일한 의미의 트렌지션으로 확장한 후 제거한다.

c. 내부 생성 이벤트 수요 파악

아사달 모델에는 내부에서 자동적으로 생성되는 많은 이벤트들이 있다. 예를 들어, 스테이트에 들어오고 나갈 때 발생되는 en(S), ex(S) 이벤트, 데이터 발생시 발생되는 gen(D) 이벤트, 데이터 변화시 발생되는 ch(D) 이벤트, 프로세스 종료시 발생되는 tmd(P)이벤트 등이 그것이다. 만일 이러한 내부 생성 이벤트가 모두 발생 및 처리된다면 성능에 악영향을 끼칠 것이다. 따라서, 내부적으로 발생되는 이러한 이벤트가 사용되는 것만을 조사하여, 수요로 파악된 것들에 대해서만 자동적으로 발

생토록 하고 나머지 경우엔 발생하지 않도록 한다.

d. SpM에 대한 Time Out 관계 제거

SpM은 그 자체로서 주기를 가지고 있기 때문에, 사용자가 명세 단계에서 명시한 해당 프로세스의 시간적인 특성은 무시되어야 한다. 그렇지 않으면, 시간적인 특성을 두번 중복해서 가지게 되어, 프로세스 활성화 시점을 놓칠 수 있고, 성능을 저하시킬 수 있다. 따라서 객체에 할당된 Statecharts에서 SpM과 관련된 스테이트를 찾아, 이로부터 시작되는 TimeOut 관계는 제거시킨다. 그러나 만일 SpM의 주기와 Statecharts에서 표현된 시간 주기가 다를 경우에는 이를 제거시키지 않고, Warning을 발생시킨다.

생성된 코드를 위한 라이브러리

생성되는 코드는 라이브러리를 기반하여 만들어진다. 이 라이브러리는 ASADALTMOCodeLIB.lib 이라는 정적 라이브러리로 만들어져 있다. 이 라이브러리는 생성될 코드가 상속받을 기본적인 베이스 클래스들이 정의되어 있다. 이 라이브러리의 핵심 부분은 Statecharts를 실행시키는 StateTree라는 클래스와 Time Out 이벤트를 관장하는 Timer 클래스이다. 이 두가지에 대해서 아래에서 알아보도록 한다.

a. StateTree

StateTree는 객체에 할당된 Statecharts를 실시간으로 수행시키는 엔진이다. Statecharts의 문법대로 수행되도록 작성되어 있고, 성능 향상을 위해 필요한 이벤트를 미리 예측하는 테이블을 관리한다. 예측 테이블이란 현재의 활성화된 상태로부터 시작되는 트렌지션이 필요로 하는 이벤트가 e1, e2 두 개가 있다면, 이 2개의 이

벤트와 이 이벤트를 필요로 하는 스테이트를 관리하는 테이블을 말한다. 이 예측 테이블을 통해, 예측 테이블에 포함되어 있지 않은 이벤트는 바로 돌려 보내고, 포함된 이벤트가 온 경우, 바로 이를 필요로 하는 스테이트로 부터 트랜지션이 일어나도록 한다.

b. Timer

Timer class 는 Statecharts에서 사용하는 모든 Time Out이벤트를 관리하기 위해 만들어졌다. 만일 Timer를 내부 Thread나 Signal로 객체내에서 사용한다면, TMO 커널을 사용하지 않기 때문에 시간이 늦춰질 수 있고 신뢰성이 낮아진다. 따라서, Timer 객체는 TMO를 상속받아서, 사용자가 원하는 시간 간격으로 SpM이 활성화되면서 요청된 시간이 지나면, 요청을 한 채널로 Time Out 이벤트를 발생시킨다. 이 Timer는 6번 채널을 사용하고 있다.

생성되는 코드

생성되는 코드는 각 객체에 대해서 5개의 파일이 만들어진다. 이들 각각은 아래와 같다.

- States.h : 해당 객체에 할당된 Statecharts에 포함된 State의 정보들이 포함된다. 각 State는 ASADALTMOCodeLIB.lib에 포함된 State class를 상속받는다.
- Transitions.h : 해당 객체에 할당된 Statecharts에 포함된 Transition의 정보들이 포함된다. 각 Transition은 Transition class를 상속받는다.
- Code.h : StateTree 를 상속받는 것으로, States.h와 Transitions.h 에 정의된 클래스들의 연결 관계를 가지고 있어서 StateTree가 구동가능하도록 하는 클

래스가 정의되어 있다. 또한 해당 객체에 연결된 데이터들의 정보를 가지고 있다.

- Proxy.h : 해당 객체로부터 외부 객체로 보내지는 이벤트들을 관리하고, 이들이 어느 채널로 보내져야 하는가를 알고 있는 클래스이다.

- TMO.h / TMO.cpp : TMO를 상속받는 클래스로, Code class 타입이 객체를 멤버로 가지고 있다. 해당 객체에 연결된 메쏘드들이 주어진 특성에 따라 SpM혹은 SvM으로 정의되어 있다. 그리고 외부로부터 이벤트를 받아들이는 SvM을 추가로 가지고 있다.

이렇게 생성되는 객체는 aip.cpp 안에서 모두 전역 포인터로 정의된다. aip.cpp 에서는 TMOMain과 TMOExit을 정의하여, 이들 객체들을 활성화/비활성화 시키는 함수를 포함한다.

코드 사용법

생성된 코드를 사용하기 위해서, 사용자는 생성된 코드로 이벤트나 데이터를 전송할 수 있어야 하고, 또한 관심있는 이벤트나 데이터를 받아볼 수 있어야 한다.

이벤트를 전송할 때는 생성된 TMO class 가 멤버로 가지고 있는 stm(Code type 의 class)의 generated 함수를 호출하면 되고, 데이터 전송의 경우에는 generatedData 를 호출하면 된다. 이벤트를 받기 위해서는 먼저, 자신이 관심 있는 이벤트를 registerCallback이라는 함수를 사용하여 stm 에 알려주어야 한다. stm에 알려줄 때는

CBObj를 상속받은 클래스를 등록해 놓아야 한다. 이렇게 하면 stm 은 해당 이벤트의 발생시에 등록된 CBObj를 상속받은 클래스의 eventGenerated 함수를 호출한다. (이것은 Java 스타일의 Listener관계와 비슷하다. Callback 함수 형태가 아니라 Listener 형태로 작성된 이유는 Java 로 생성되는 코드와 일관성을 유지하기 위함이다.) 아래에서는 생성된 코드를 사용한 예제 프로그램을 보여주고 주석으로서 설명을 하였다.

코드 생성 시제품

코드 생성 시제품으로 간단한 덧셈 계산기 예제를 사용하였다. 계산기 예제는 입력을 받아들이는 프로세스와 덧셈 연산을 수행하는 2개의 프로세스로 구성되어 있고, 계산 수행을 주관하는 Statecharts가 포함되어 있다. 예제는 두 프로세스를 각각 서로 다른 객체에 할당하고, 입력을 받아들이는 프로세스는 SpM으로, 덧셈 연산을 수행하는 프로세스는 SvM으로 할당한다. 생성된 코드와 연결될 사용자 코드는 Visual C++로 계산기 GUI를 작성하였다.

이 시제품 개발 과정에서, 이러한 일련의 프로세스의 장점이 모두 드러났다. 먼저 명세 기법을 사용하여 개발 전에 잠재된 불일치 등의 문제들을 해결하고, 완벽한 모델을 명세할 수 있었으며, 명세 단계가 끝나면 코드 생성기를 통해서 손쉽게 프로그램이 완성되었다. GUI 프로그램의 경우 GUI 개발 도구인 Visual C++를 사용하여 개발 시간이 매우 짧았고, GUI 개발 도구로 생성된 코드와 ASADAL로 생성된 코드의 결합도 매우 쉬웠다. 특히, 명세에 문제가 있을 경우에는 단지 코드 생성만 다시 하고 GUI 코드는 그대로 두면 되므로, 여러가지 테스트가 손쉽게 이루어졌다.

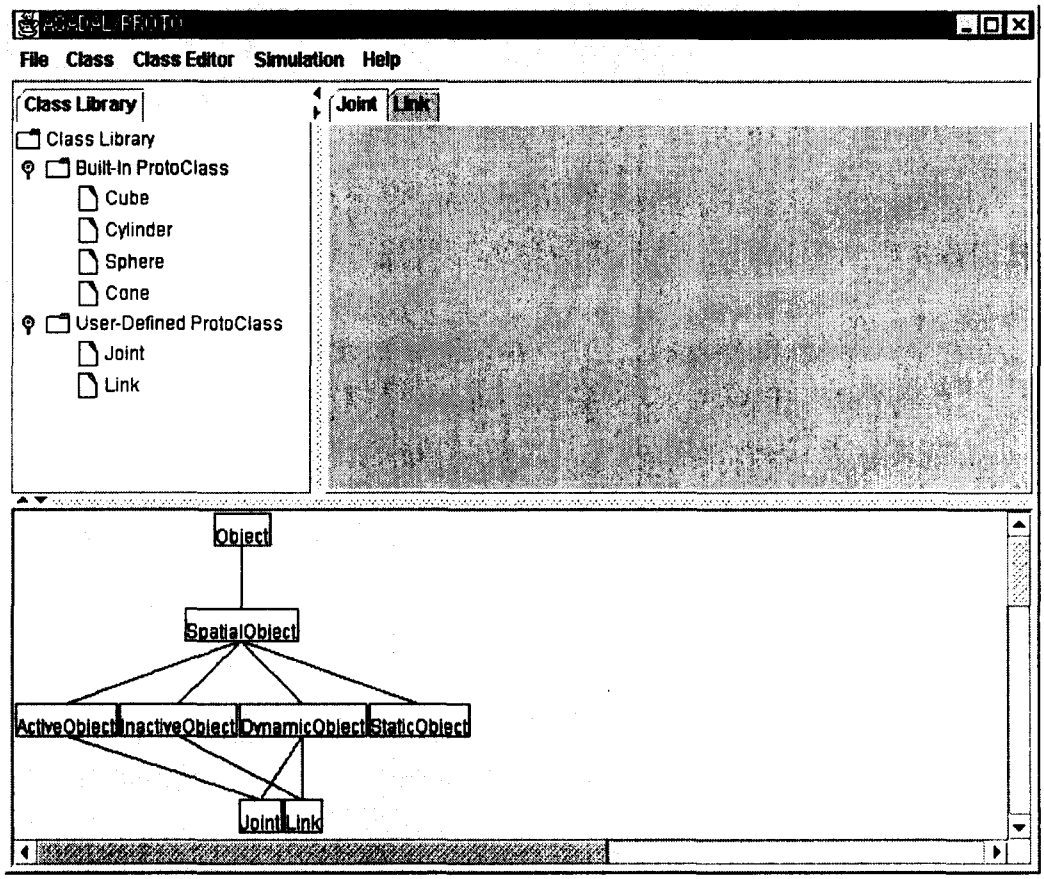
```
#include <tfx.h>
#include "MyTmo2.h"
#include "MyTmo.h"

class CB : public CBObj        // CBObj를 상속받은 클래스
{
public:
        void eventGenerated( string evt ) // event발생시 불리는 함수
        {
                cout << "== Event == : " << evt << endl;
        }
};

void main()
{
        extern void TMOMain();          // aip.cpp에 정의된 기동 함수
        extern void TMOExit();          // aip.cpp에 정의된 소멸 함수
        extern MyTMO* pMyTMO;           // 생성된 코드 포인터
        extern MyTMO2* pMyTMO2;

        CB *cb = new CB;
        tfxInitWTmos(TMOMain, TMOExit);      // TMO 구동

        pMyTMO->stm->registerCallback( cb );   // stm에 Callback 등록
                        // cb만 등록할 때는 모든 이벤트에 대해서 다 알려줌
        pMyTMO2->stm->registerCallback( cb, "e1" );

        pMyTMO->stm->initialize();          // stm 구동
        pMyTMO2->stm->initialize();

        pMyTMO->stm->generated( "e" );              // stm으로 이벤트 전송
        pMyTMO2->stm->generatedData( "a", 10 );   // stm으로 데이터 전송
}
```

나. 가시화 도구 및 VR(Virtual Reality) 인터페이스 저작도구

여기서는 가시화 도구 및 가상 현실 인터페이스 저작에 대해 설명한다.

먼저 ASADAL/Proto의 실행 화면과 메뉴들의 기능에 대해 간단히 설명한다.

Main Window



ASADAL/PROTO를 실행시키면 위와 같은 윈도우 창이 생성된다. 화면은 menu(위쪽), library(왼쪽 위), inheritance diagram(아래), 작업창(오른쪽 위)으로 분할되어 있다.

Library

사용자가 정의한 클래스들은 라이브러리에 저장되며 User-Defined ProtoClass 아

래에 나타난다. 라이브러리에 저장되어 있는 클래스들은 다른 클래스를 정의할 때 그것을 구성하는 subobject로 사용될 수 있다.

Inheritance diagram

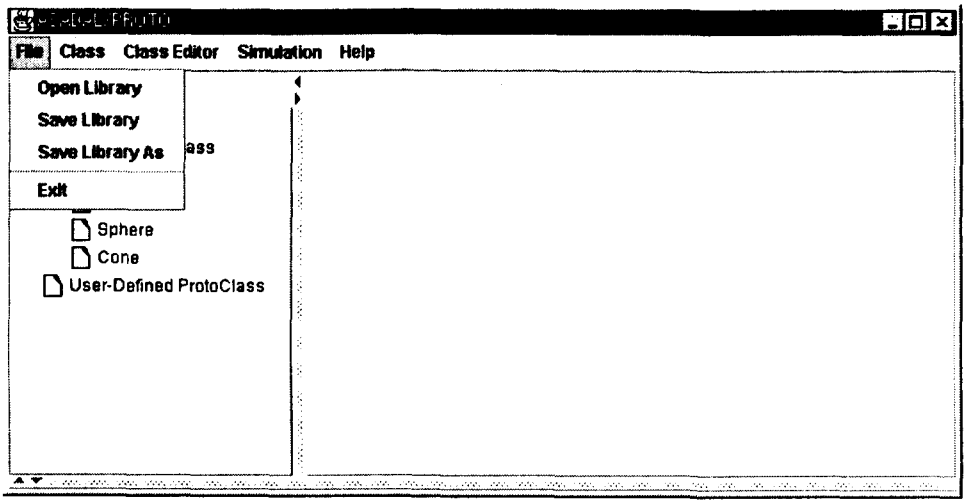
클래스 사이의 상속 관계를 보여 주는 diagram이다. 기본적으로 multiple inheritance를 허용하며, root에는 "Object"라는 클래스가 있으므로 모든 ProtoClass는 "Object" 클래스의 후손으로 정의된다. Object를 상속하는 클래스 중에서 3차원 spatial 정보를 가지는 클래스를 "SpatialObject"라고 하고, 모든 SpatialObject는 자신에게 붙어 있는 passive children을 움직일 수 있는 객체인가에 따라 "ActiveObject"와 "InactiveObject"로 나누어지며 움직일 수 있는 객체인가에 따라 "DynamicObject"와 "StaticObject"로 나누어진다. "Joint" 클래스는 자신에게 붙어 있는 link를 움직일 수 있고 자신도 움직일 수 있으므로 ActiveObject와 DynamicObject로부터 상속받으며, "Link" 클래스는 다른 객체를 움직일 수는 없지만 자신이 움직일 수 있으므로 InactiveObject와 DynamicObject로부터 상속받는다.

작업창

사용자가 클래스를 정의하기 위해서 사용하는 공간으로서 tab pane을 이용하여 구현되어 있다. 각 tab에서 하나의 클래스를 editing할 수 있으므로 여러 개의 tab에서 동시에 여러 클래스를 editing하는 것이 가능하다.

Menu

1) File Menu



- Open Library

저장되어 있는 라이브러리를 불러온다. 라이브러리에 저장되어 있는 file들이 Library창에 나타나며, 그들 사이의 inheritance관계는 inheritance diagram에 나타난다.

- Save Library

라이브러리를 현재 열어놓은 file에 저장한다.

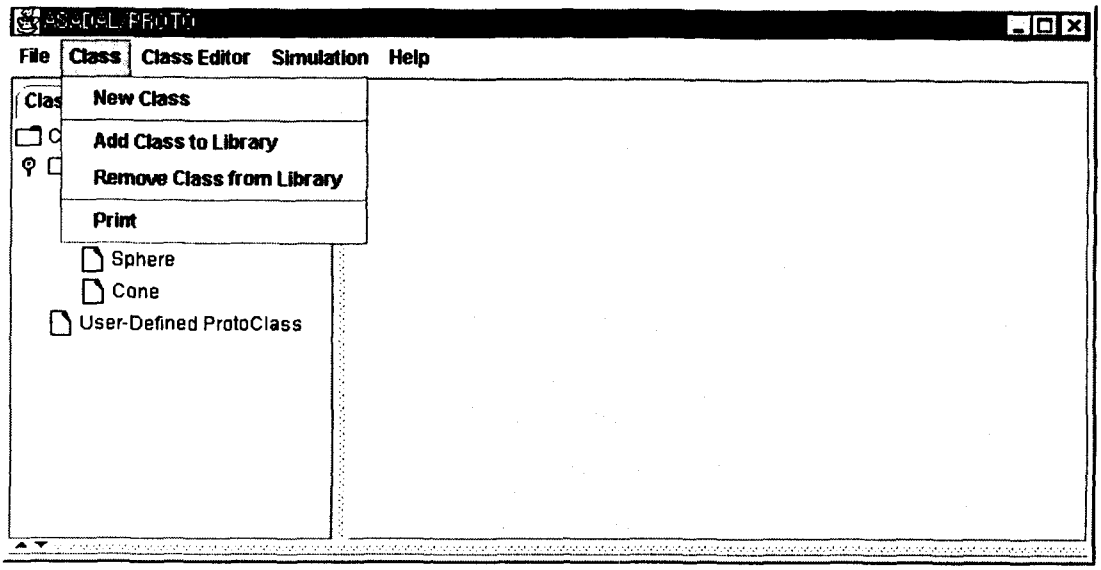
- Save Library As

라이브러리를 사용자가 지정하는 file에 저장한다.

- Exit

ASADAL/PROTO를 끝낸다.

2) Class Menu



- New Class

새로운 클래스를 정의하기 위한 메뉴로서 새로 정의하려는 클래스의 이름과 inheritance 관계를 정의하게 한다. 클래스 이름은 나중에 변경할 수도 있다.

- Add Class to Library

클래스에 대해 필요한 명세를 다 하고 나면 라이브러리에 등록하여 다음에 재사용될 수 있게 한다.

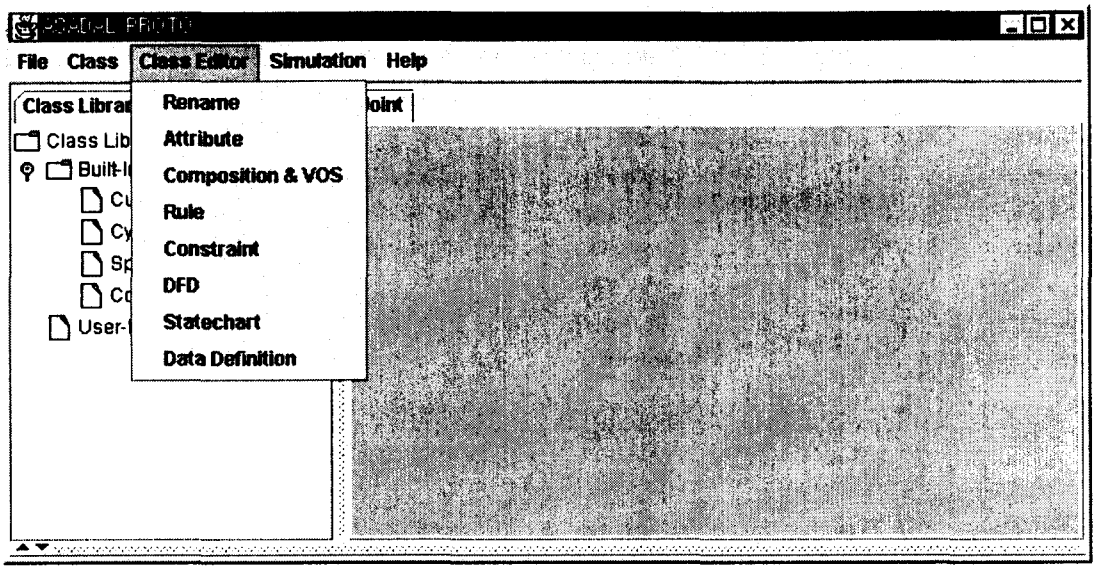
- Remove Class from Library

라이브러리에서 불필요한 클래스를 제거한다.

- Print

Not available yet.

3) Class Editor



- Rename

클래스의 이름을 바꾼다.

- Attribute

클래스가 가지는 attribute들을 정의할 수 있는 UI를 제공한다.

- Composition & VOS

이미 정의되어 있는 ProtoClass를 instantiation하여 composition 관계를 정의할 수 있는 composition editor와 sub object들 사이의 Motion hierarchy를 정의할 수 있는 MH editor를 제공한다. 그리고, 각 sub object의 attributes을 editing하여 원하는 configuration을 가지게 하고 그것들이 가시화되어 보이는 viewer를 제공한다.

- Rule

클래스의 sub object들 사이에 존재하는 rule들을 정의할 수 있는 UI를 제공한다.

- Constraint

클래스에서 지켜져야 될 constraint들을 정의할 수 있는 UI를 제공한다.

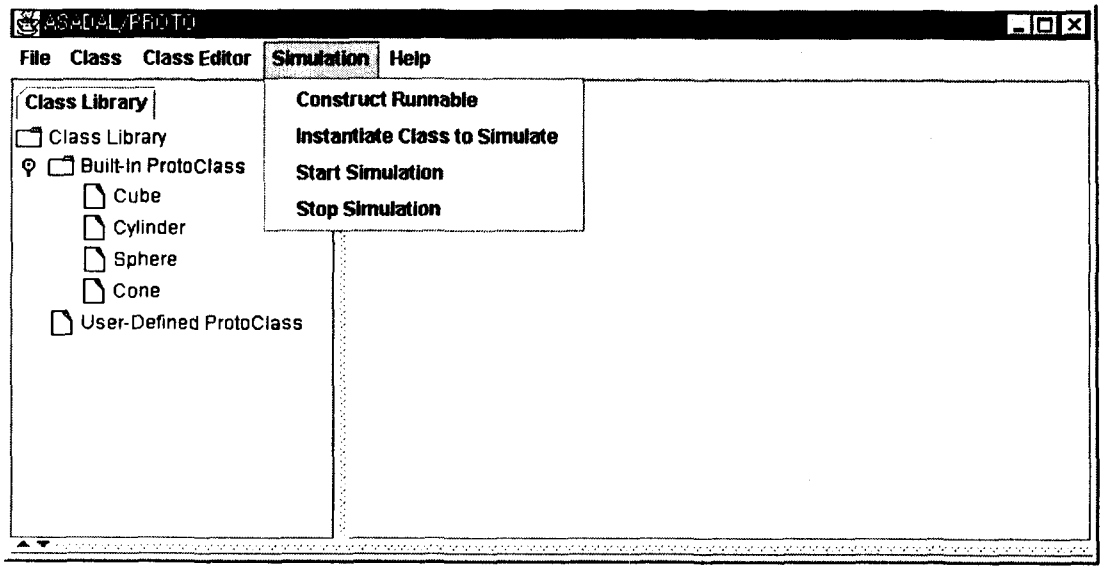
- DFD

클래스가 가지는 DFD를 명세한다.

Statechart

클래스가 가지는 Statechart를 명세한다.

Data Definition



4) Simulation

- Construct Runnable

Statechart와 DFD 명세로부터 code를 만든다.

- Instantiate Class to Simulate

어떤 클래스를 시뮬레이션할 것인지 결정한다.

- Start Simulation

시뮬레이션을 시작한다.

- Stop Simulation

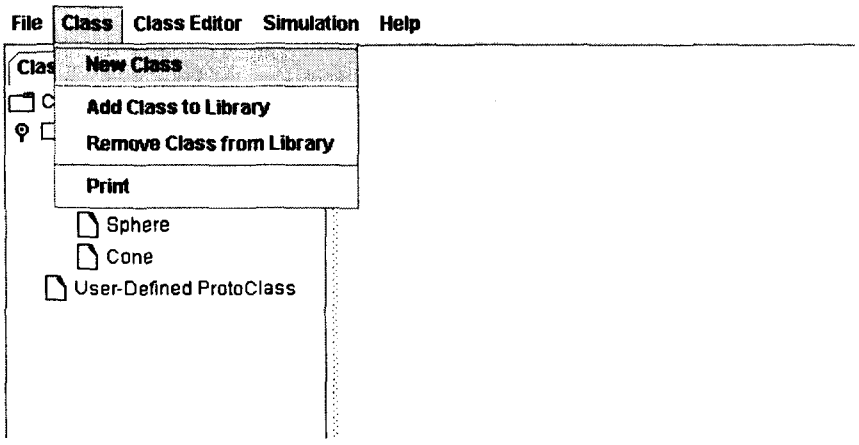
시뮬레이션을 끝낸다.

이제 ASADAL/PROTO를 이용하여 객체를 만드는 방법에 대해서 설명한다.

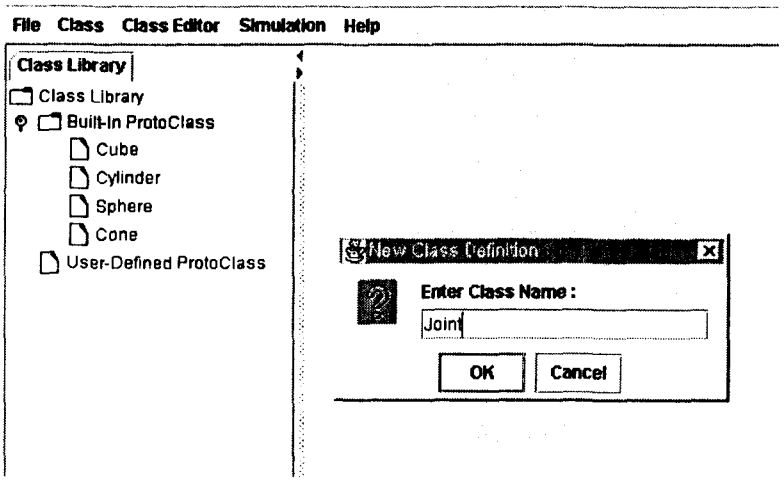
Joint와 Link로 구성된 간단한 Robot 만들기 예제를 보면서 ASADAL/PROTO tool의 사용법을 설명할 것이다.

Joint class 정의하기

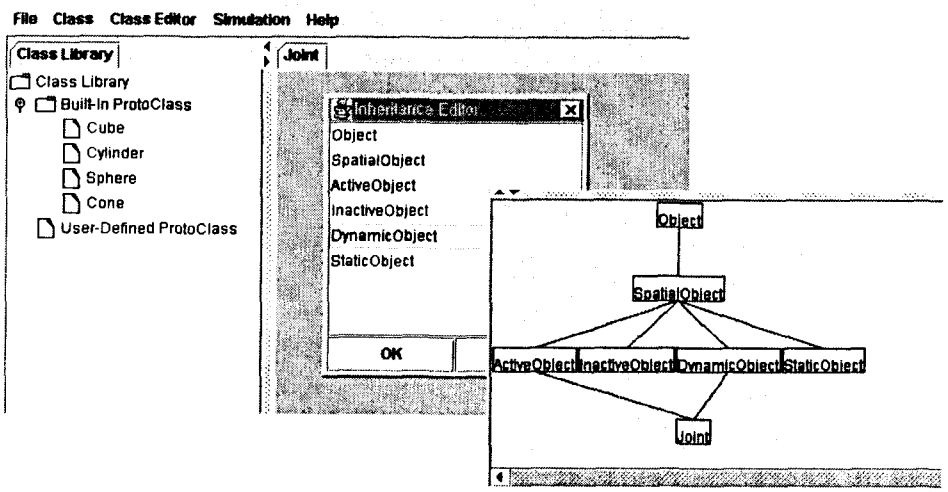
1) Class menu에서 "New Class"를 선택하고 새로 정의할 클래스의 이름을 입력한다.



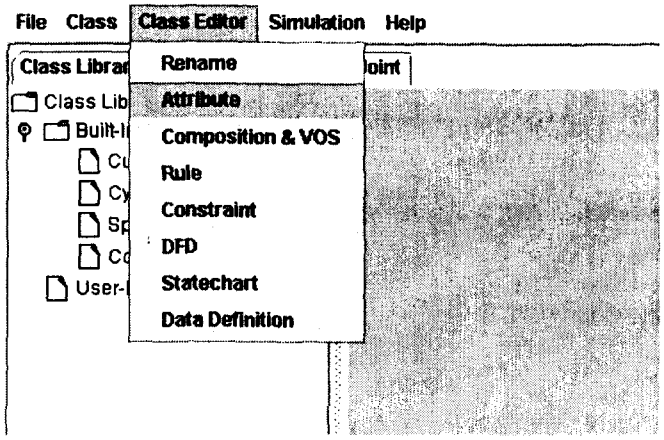
2) 이름을 입력하면 작업창에 새로 정의하는 클래스의 tab(tab의 title을 class의 이름이 된다)이 나타나고, 클래스의 inheritance관계를 정의하기 위한 창이 나타난다. 창에 나타난 클래스 중에서 Parent class들을 선택하고(ctrl key로 multiple class를 선택할 수 있다) "OK"를 누르면 그 관계가 inheritance diagram에 update된다.



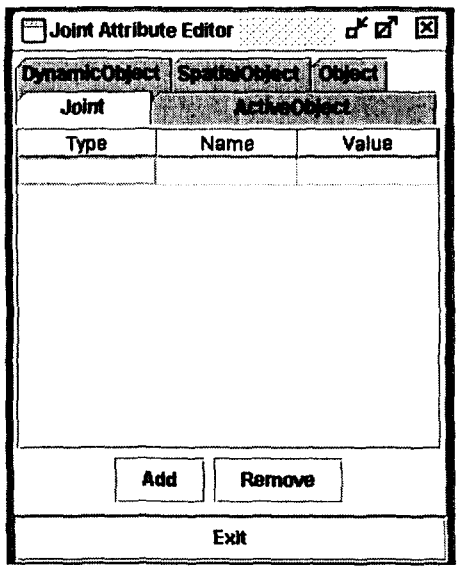
3) 클래스에 대한 여러 가지 정보를 editing하기 위해서 "Class Editor" 메뉴를 사용한다. 어떤 editor를 선택했을 때 그 대상이 되는 클래스는 현재 선택되어 있는 tab의 title을 이름으로 갖는 클래스이며, editor들은 필요한 대로 순서에 상관없이 사용될 수 있다.

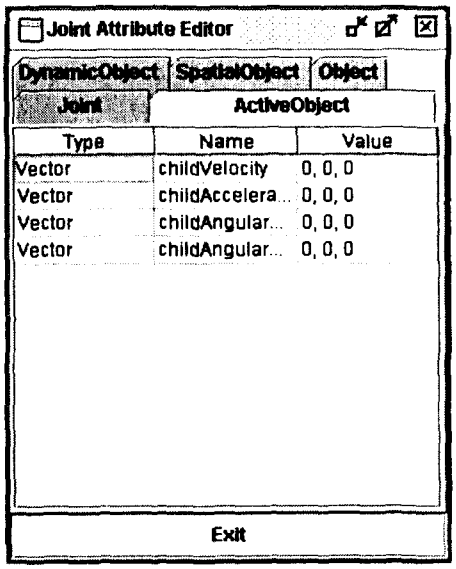


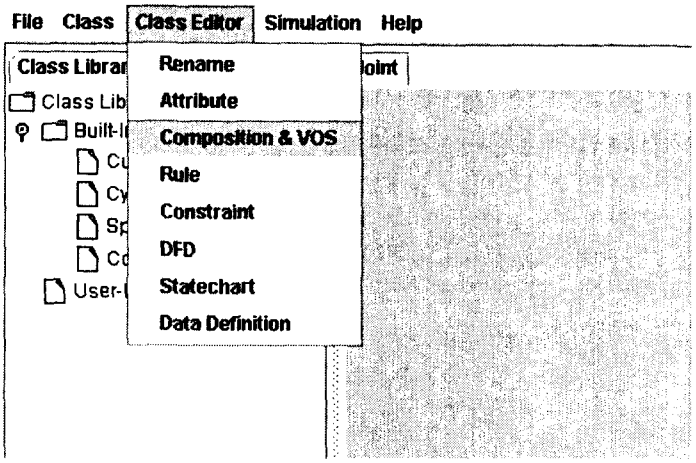
클래스의 attribute를 editing하기 위해서는 "Attriute"을 선택한다.



Attribute Editor는 inheritance 관계에 의해 상속받고 있는 모든 클래스들의 attribute들을 tab pane을 이용하여 보여 준다. 새로 정의하는 "Joint" 클래스는 처음에는 attribute이 하나도 없지만 Joint 클래스만이 가지는 attribute들을 add/remove 버튼을 이용하여 editing할 수 있다.

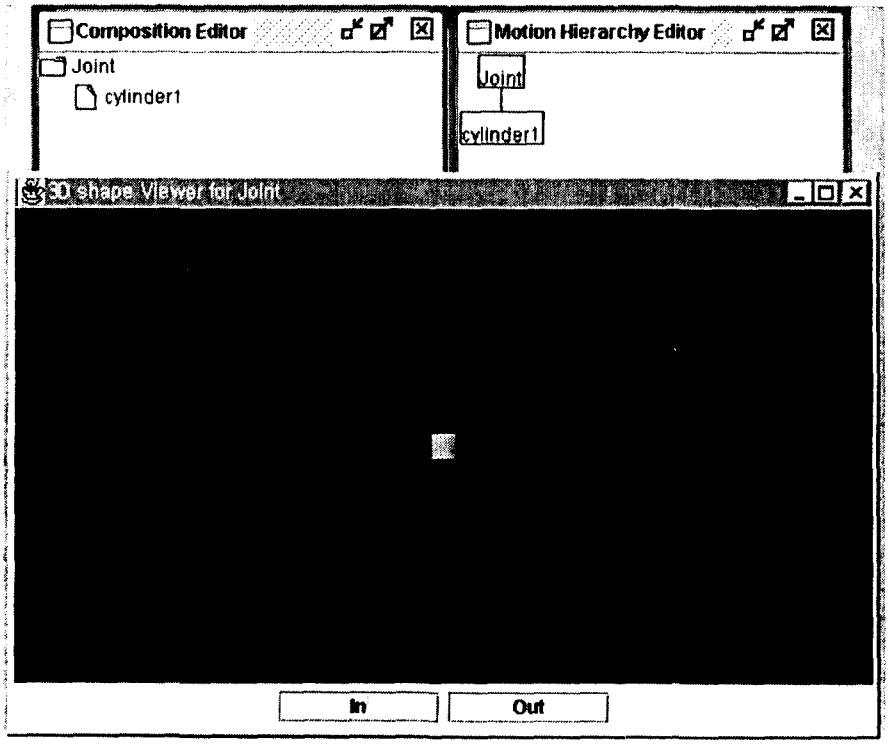






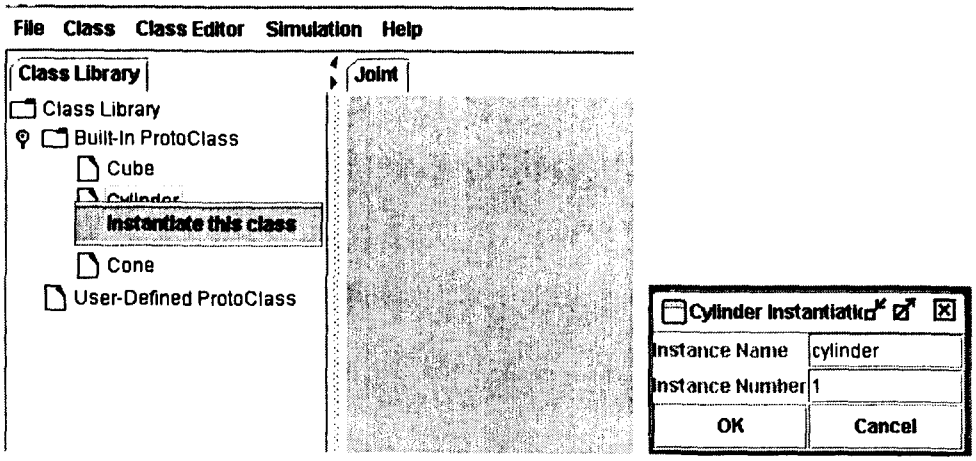
4) Composition & Rule

이 sub menu는 클래스의 form specification 을 위해 필요로 하는 editor들을 제공한다. 이 메뉴를 선택하면 다음의 세 개의 창이 생성된다.

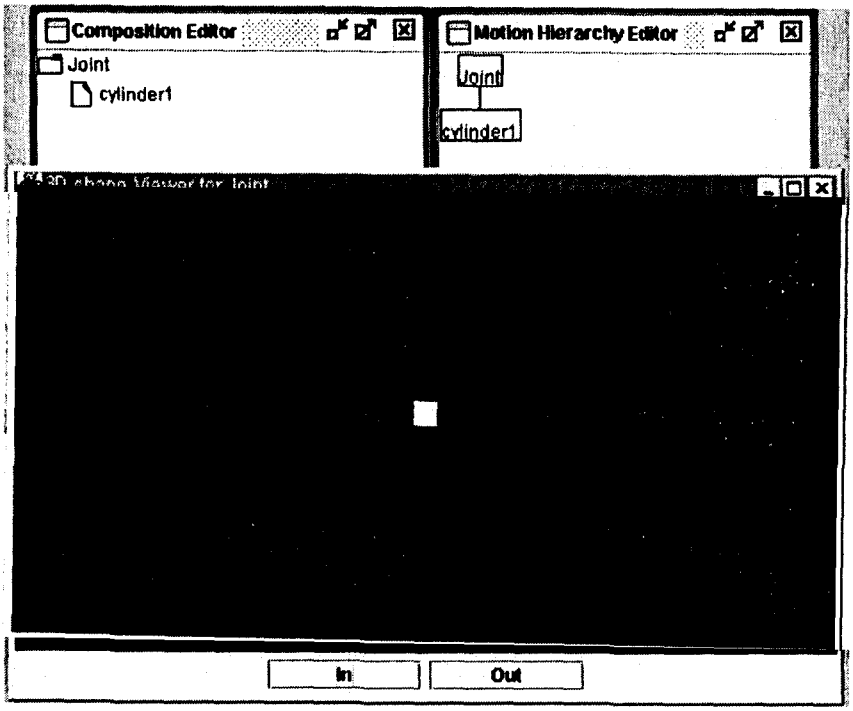


Class Library로부터 한 클래스를 선택하면(왼쪽 마우스 버튼으로 class 이름을 누르면 선택이 된다) popup menu가 뜬다. 그 중에서 "Instantiate this class"를 선택하면

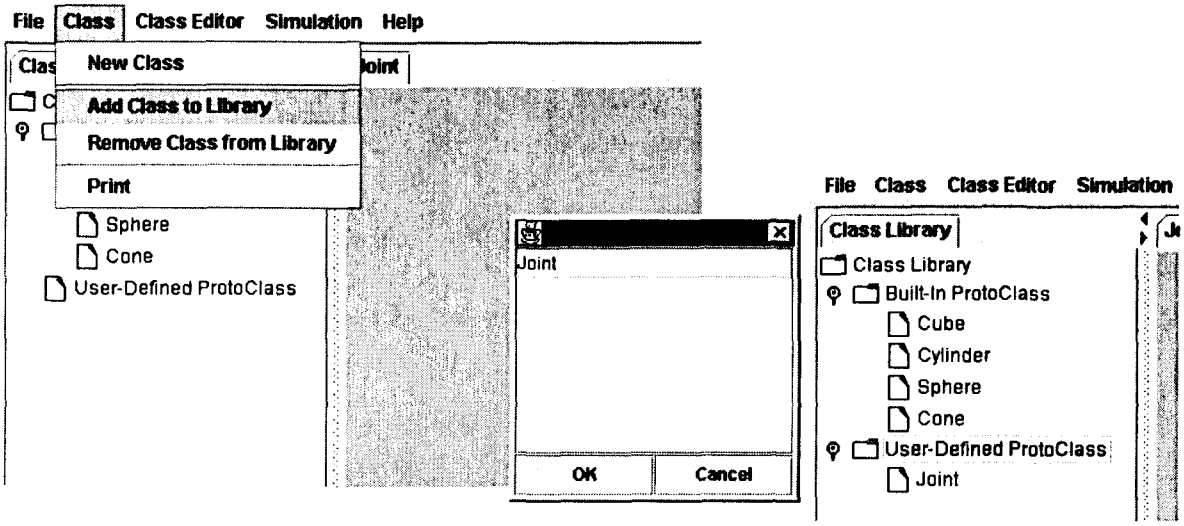
그 클래스의 인스턴스를 생성하기 위해 필요한 인스턴스의 이름과 갯수를 입력할 수 있는 창이 뜨고, 입력된 갯수대로 만들어지는 인스턴스들은 새로 만들어지는 클래스의 sub object가 된다.



위의 예제에서는 Cylinder 클래스의 인스턴스 1개를 만들고 있다. Instance Name이 cylinder이고, Instance Name이 1이면 그 인스턴스의 이름은 자동으로 cylinder1이 된다.



5) 여기서 사용되는 "Joint" 클래스는 Statechart와 DFD가 없어도 되므로, form 명세만으로 끝내고 완성된 클래스를 class library에 등록해 둔다. 라이브러리에 등록해 둠으로써 built-in 클래스들을 instantiation하듯이 Joint 클래스도 instantiation될 수 있으며, Robot을 만들 때 subobject로 이용할 수 있다.



Link class 정의하기

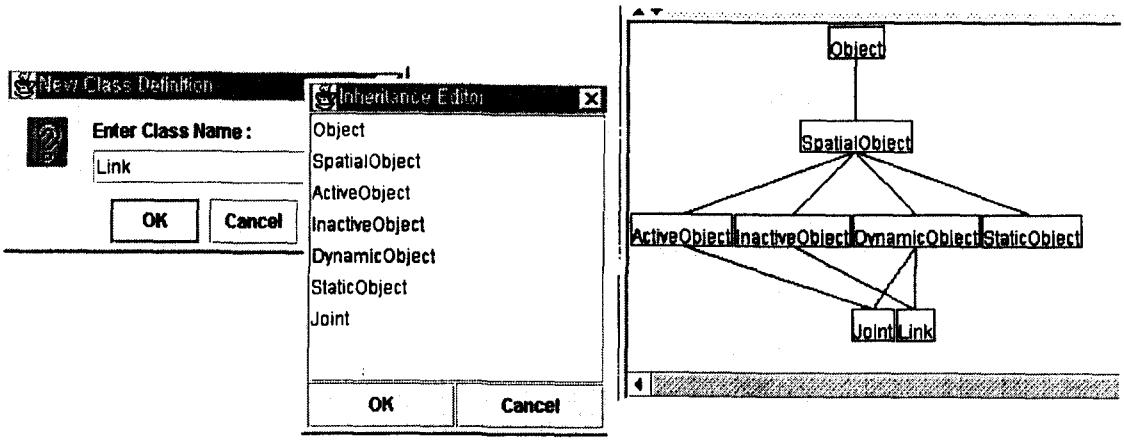
Joint class를 정의하는 방법과 똑같은 방법으로 Link class도 정의할 수 있다.

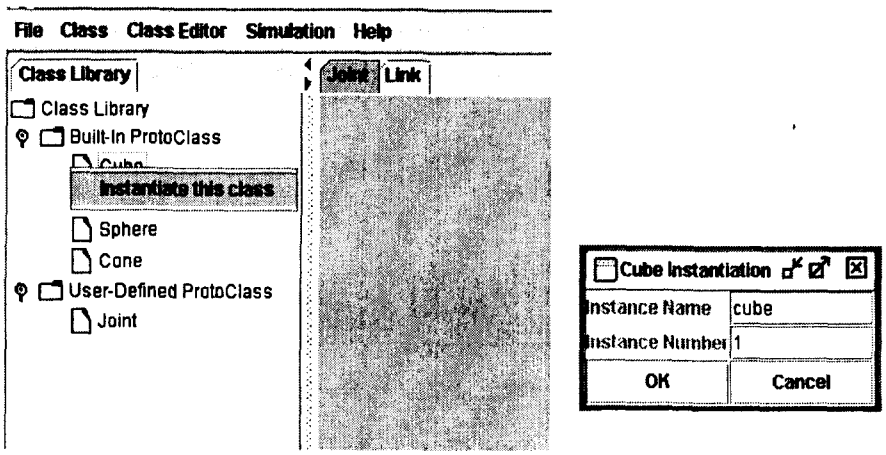
1) 클래스의 이름과 inheritance 관계를 정의한다.

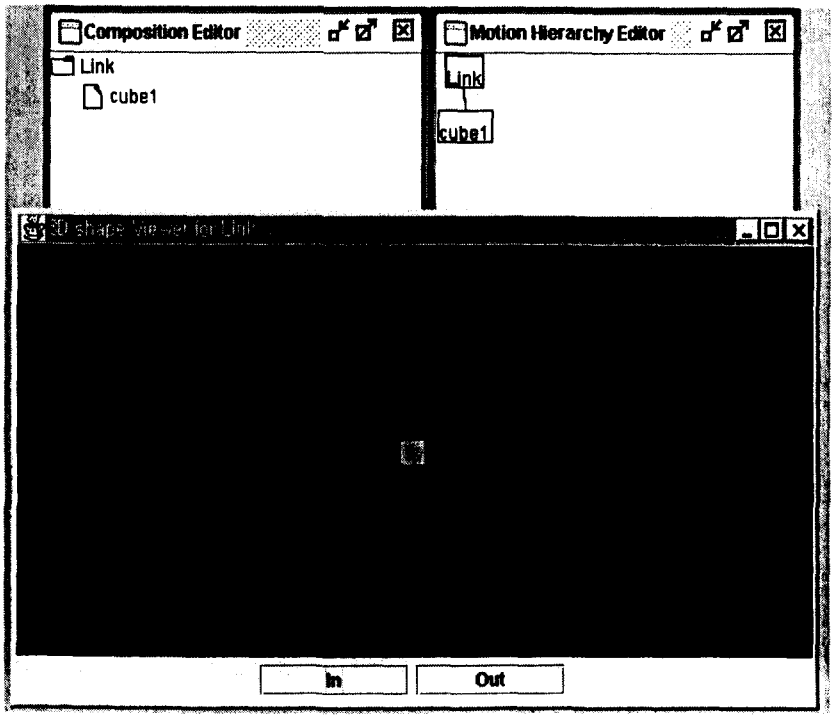
2) Class Editor 메뉴에서 "Composition & VOS"를 선택하고 세가지 창이 만들어지면 Class Library로부터 원하는 클래스의 인스턴스를 생성시킨다.

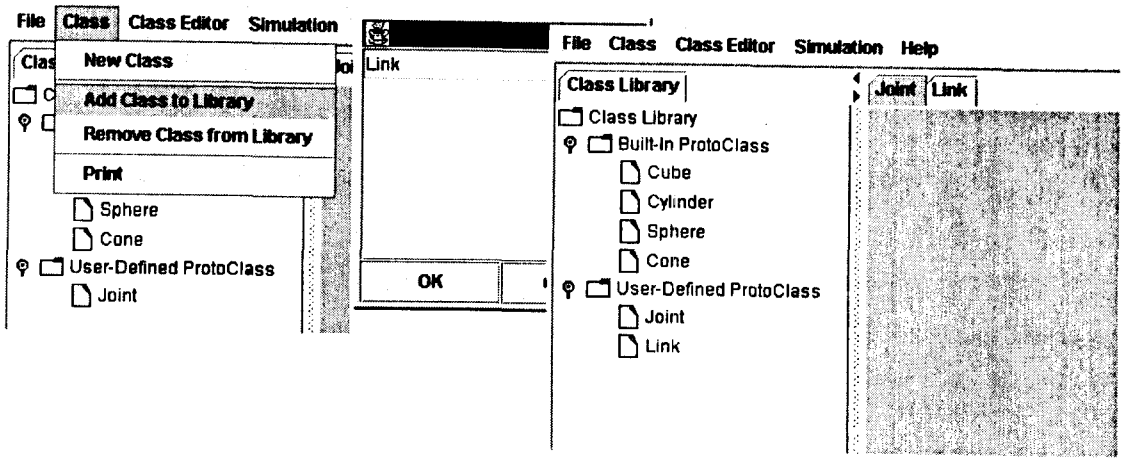
3) Cube 클래스의 인스턴스 1개를 만들면 cube1이 세 개의 창에 나타나는 것이 보인다.

4) Link 클래스도 Class Library에 등록해 둔다.


New Class Definition
Enter Class Name :
Link
OK
Cancel
Inheritance Editor
Object
SpatialObject
ActiveObject
InactiveObject
DynamicObject
StaticObject
Joint
OK
Cancel
Object
SpatialObject
ActiveObject
InactiveObject
DynamicObject
StaticObject
Joint
Link

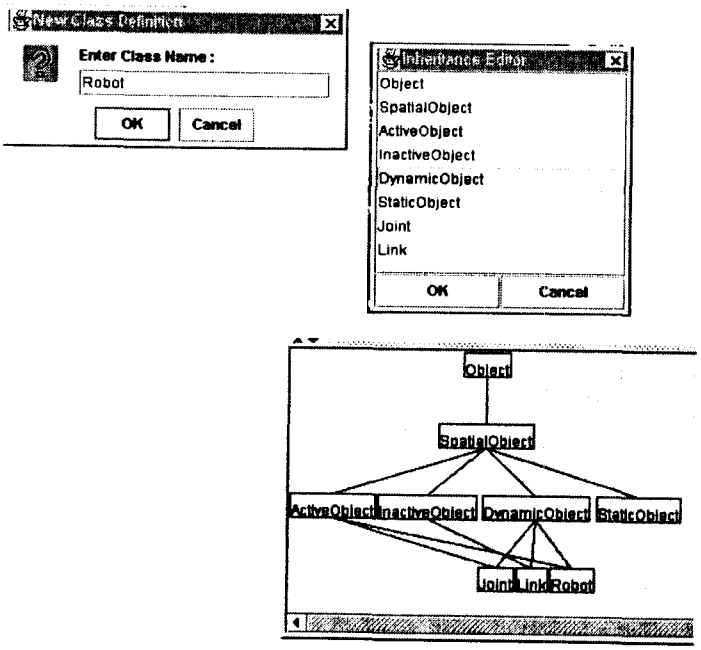

File Class Class Editor Simulation Help
Class Library
Class Library
Built-In ProtoClass
Instantiate this class
Sphere
Cone
User-Defined ProtoClass
Joint
Joint
Link
Cube Instantiation
Instance Name
cube
Instance Number
1
OK
Cancel


Composition Editor
Link
cube1
Motion Hierarchy Editor
Link
cube1
In
Out



Robot class 정의하기

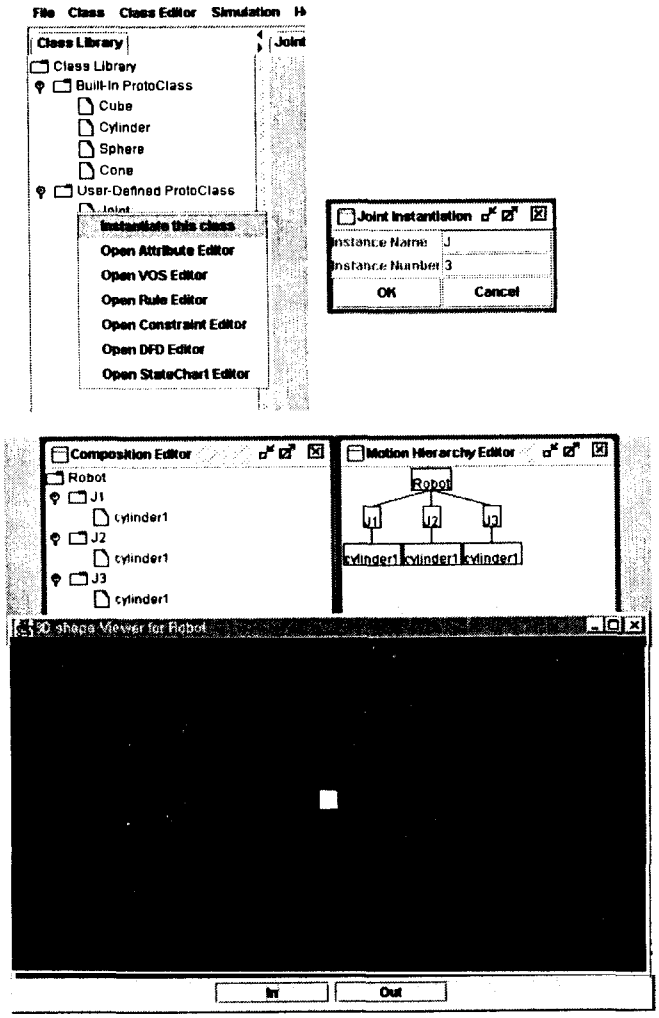
1) 클래스의 이름과 inheritance 관계를 정의한다.



2) "Composition & VOS"를 선택하고 세가지 창이 만들어지면 Class Library로부터 원하는 클래스의 인스턴스를 생성시킨다.

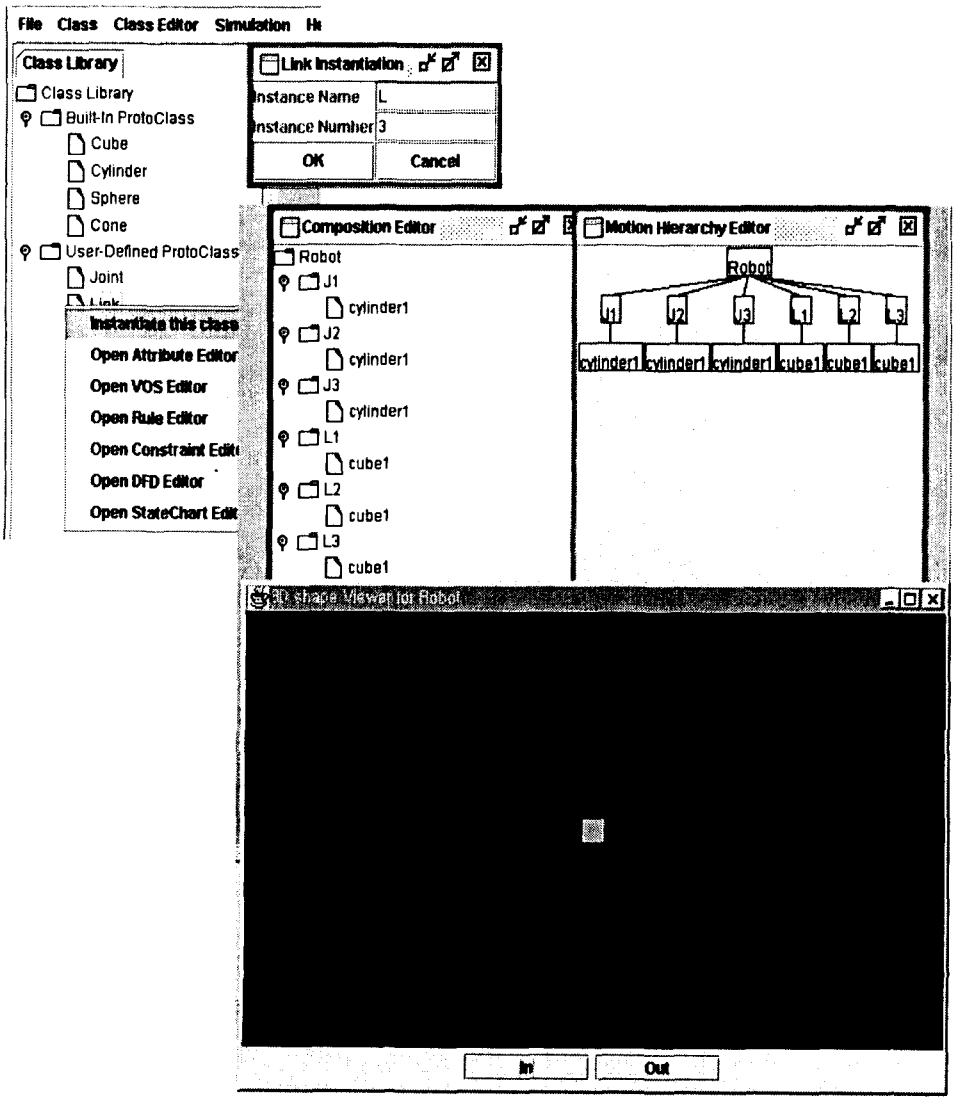
2.1) Joint instance 만들기

Joint 클래스의 인스턴스 3개를 만들면 J1, J2, J3가 만들어지고 Joint class가 가지는 default position(원점)과 orientation, 색상(gray), shape(cylinder) 등이 적용되어 위의 Shape Viewer에 가시화된다. 세 인스턴스는 같은 default 값을 가지기 때문에 하나로 보이고 있는데, 후에 instance 정보를 editing하여 원하는 configuration으로 바꿀 수 있다.



2.2) Link instance 만들기

Link 클래스의 인스턴스 3개 L1, L2, L3도 똑같은 방법으로 만들어진다. 3D shape viewer에 정사면체 하나만 보이고 있지만 사실은 여섯 개의 instance가 같은 위치에 있는 것이다. 각 인스턴스의 attribute value을 바꿈으로써 원하는 configuration으로 만들 것이다.

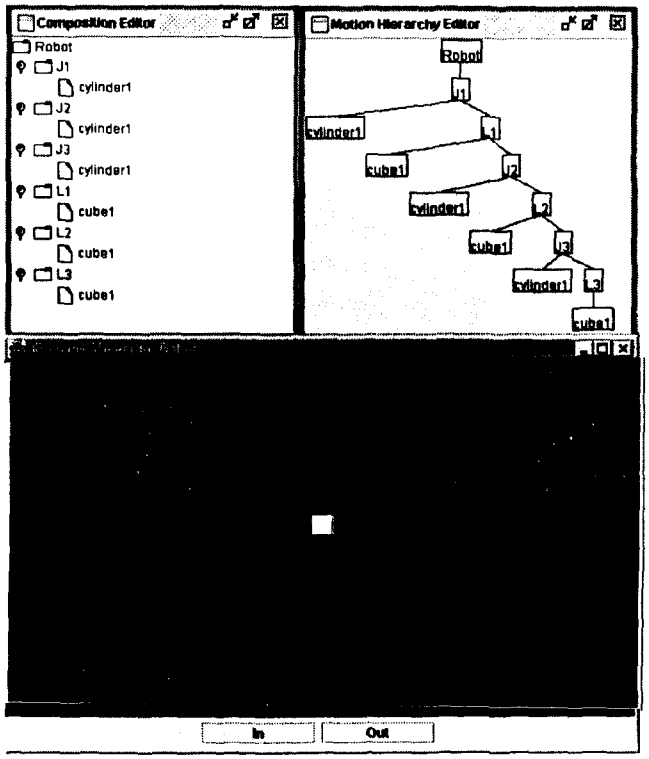


3) Motion hierarchy

Subobject들 사이의 motion dependency를 결정한다. Motion hierarchy tree상의 node들은 transform matrix를 가지고 있고 각 node들에 적용되는 transform(translation과 rotation) 값은 root의 transform matrix로부터 accumulate되기 때문에 tree상의 parent에게 적용되는 움직임은 children들에게도 똑같이 적용된다. 이러한 것을 고려하여 subobject들 사이에 존재하는 motion dependency를 명세한다.

4) Instance attribute editing하기

모든 subobject들은 그 default postion과 orientation에 의해 초기에는 원점에 나타나는 것을 위에서 보았다.이제부터 각 subobject의 attribute의 값을 변경하여 적당한 위치에 옮기고 dimension을 바꾸어 줌으로써 원하는 모양을 쉽게 만들어 나가는 과정을 설명할 것이다.



Motion hierarchy editor에서 각 node를 마우스로 누르면 그 subobject가 가지는 attribute을 editing할 수 있는 창이 생성된다. 창에서 각 attribute의 value를 바꾼 후 return을 누르면 그것의 효과가 바로 3D shape viewer에서 보인다. Viewer에 보이는 모습을 monitoring하면서 원하는 모양이 되도록 attribute을 바꾸어 나가면 된다.

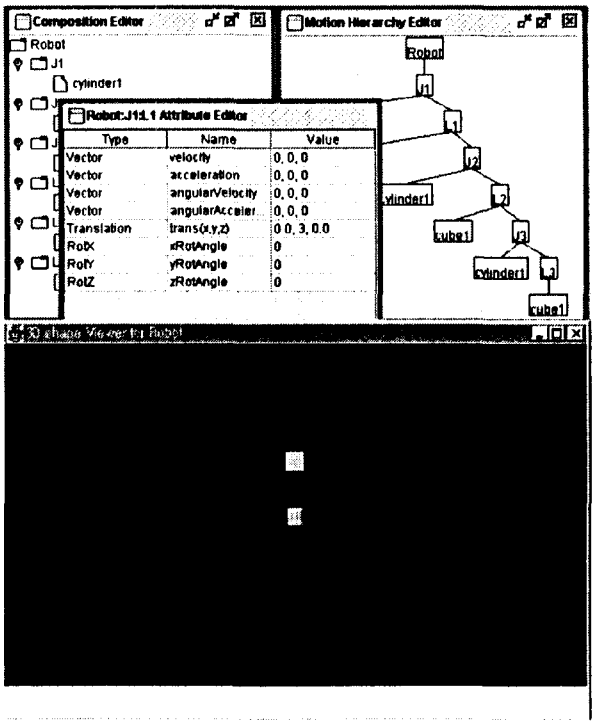
L1의 attribute중에 Translation vector를 (0, 0, 0)에서 (0, 3, 0)으로 바꿈으로써 motion hierarchy tree상의 "L1"아래의 모든 객체를 y축으로 3만큼 이동시켰다.

J2의 attribute중에 Translation vector를 (0, 0, 0)에서 (0, 3, 0)으로 바꿈으로써 motion hierarchy tree상의 "J2"아래의 모든 객체를 y축으로 3만큼 이동시켰다.

L2의 attribute중에 Translation vector를 (0, 0, 0)에서 (2, 0, 0)으로 바꿈으로써 motion hierarchy tree상의 "L2"아래의 모든 객체를 x축으로 2만큼 이동시켰다.

J3의 attribute중에 Translation vector를 (0, 0, 0)에서 (3, 0, 0)으로 바꿈으로써 motion hierarchy tree상의 "J3"아래의 모든 객체를 x축으로 3만큼 이동시켰다.

L3의 attribute중에 Translation vector를 (0, 0, 0)에서 (-3, 0, 0)으로 바꿈으로써 motion hierarchy tree상의 "L3"아래의 모든 객체를 y축으로 -3만큼 이동시켰다. 이렇게 해서 모든 subobjet들의 대강의 위치를 정할 수 있는데, 각 subobject들의 위치는 다시 attribute의 value를 바꾸어 주면 언제라도 변경될 수 있다.

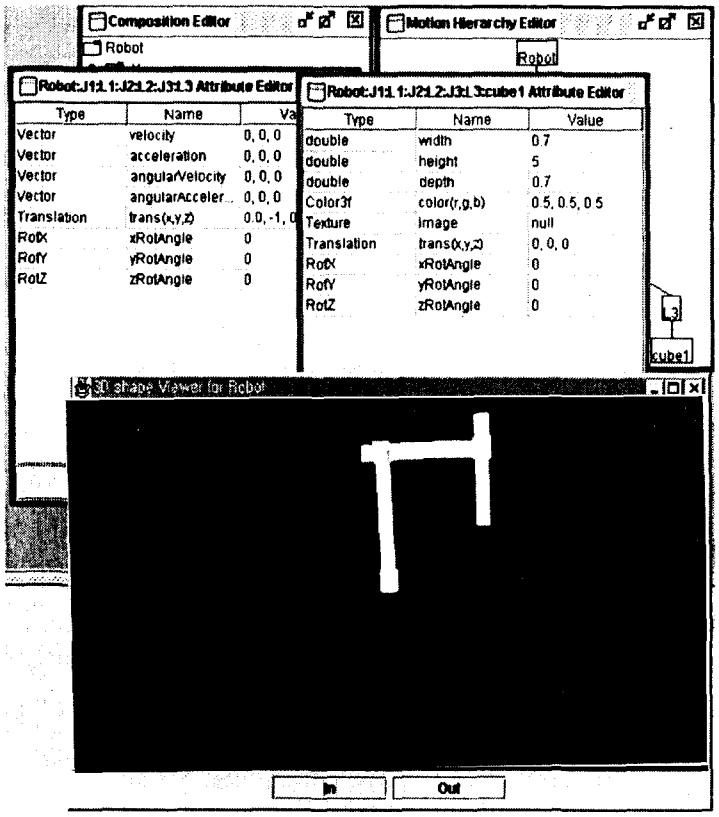


대강의 위치를 잡은 다음 각 link의 사이즈를 조절해 주면 쉽게 Robot의 3차원 모양을 만들 수 있게 된다. 물론 필요에 따라 지금하고 있는 예제처럼 단순한 모양으로 만들 수도 있고 더 복잡하고 섬세하게 만들 수도 있다.

L1의 subobject "cube1"의 attribute editor에서 width(0.7), height(6), depth(0.7)를 조절하여 J1과 J2를 잇는 link를 만든다.

L2의 subobject "cube1"의 attribute editor에서 width(6), height(0.7), depth(0.7)를 조절하여 J2과 J3를 잇는 link를 만든다.

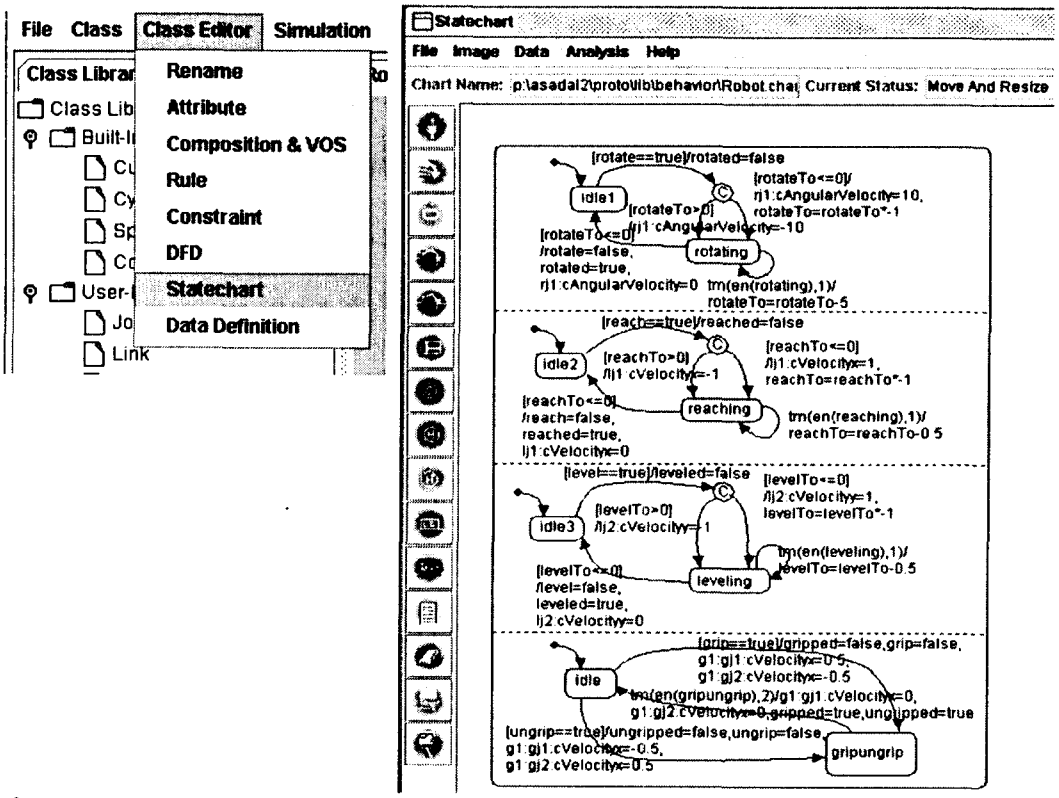
L3의 subobject "cube1"의 attribute editor에서 width(0.7), height(5), depth(0.7)를 조절하고, "L3"의 Translation vector를 (0, -3, 0)에서 (0, -1, 0)으로 바꾸어 J3과 Gripper(아직 만들지 않았음)를 잇는 link를 위와 같이 만든다.



이런 VOS명세를 통해 클래스의 form specification을 마치고, 클래스의 behavior/function specification도 만든다.

5) Statechart 명세

Statechart의 자세한 방법론과 사용법은 asadal user's manual을 참조하기 바란다. 이것은 이전의 보고서의 부록으로 제출한 바 있다.



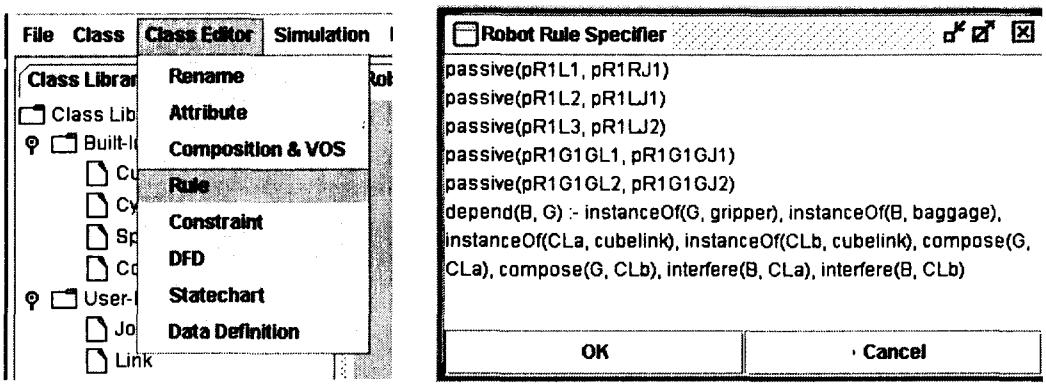
6) DFD와 Data definition

DFD의 자세한 방법론과 사용법 역시 asadal user's manual을 참조하기 바란다.

7) Rule 명세

로봇에 적용되어야 할 물리 규칙 같은 것을 rule로 명세함으로써 복잡한 물리

수식을 사용한 시뮬레이션부터 아주 추상화 단계(abstract level)에서의 시뮬레이션 등 여러 단계의 시뮬레이션이 가능하도록 해 준다. Rule을 명세할 수 있는 text editor를 제공한다.

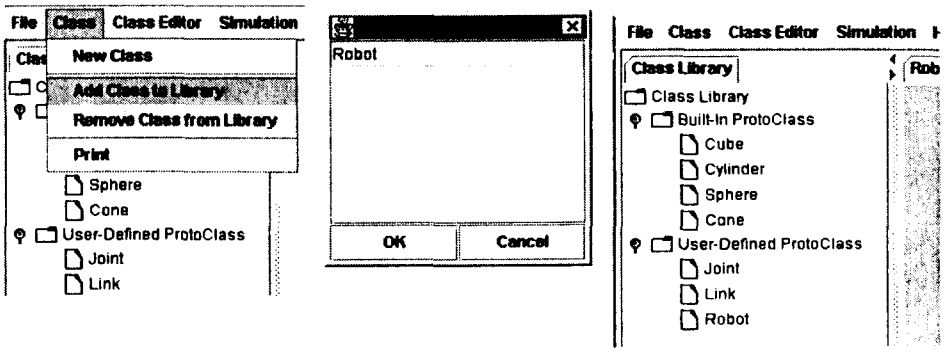


8) Constraint 명세

Robot 클래스가 위반해서는 안되는 규칙을 constraint로 명세한다. 이러한 constraint들은 시뮬레이션 도중에 계속 검사해서 위반되는 경우에 사용자에게 알려준다. 역시 text editor가 제공된다.

9) Class Library에 등록

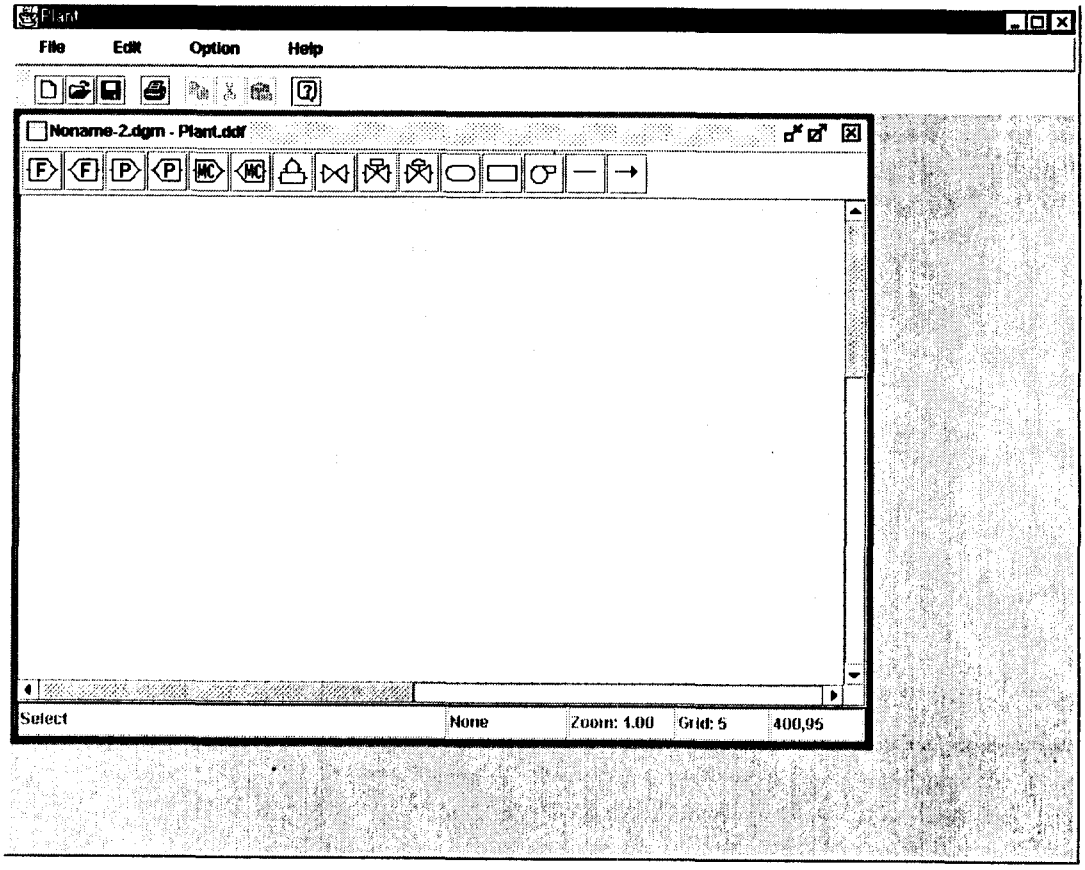
명세가 끝나면 class library에 등록해 둔다.



다. 전력 시스템 모델링 도구 개발

이 부분은 전력 시스템 모델링 도구의 사용설명서로서 어떻게 이 도구를 사용해서 모델링을 하는 방법을, 구체적으로는 실행에서부터, 컴포넌트들을 이용하여 모델링을 하는 방법들에 대해서 설명을 한다.

1. 실행하기



이 도구를 실행하기 위해서는 단지 plant.bat 라는 batch file 을 실행하는 것에서 시작이 된다. 실행이 되면 다음과 같은 화면이 나타나면서, plant diagram 을 그리기 위한 준비가 된다.

위와 같은 Window 가 나타나고, 그 다음부터는 다음 순서대로 이 도구를 사용하면 된다.

2. 각 Menu 들의 기능 설명

이 도구에서는 다음과 같은 메뉴들이 제공된다.

File 메뉴

New : New를 하면, 또 다른 그림을 그릴 수 있는 Internal Window 가 하나 더 생성되고, 다른 그림을 그릴 수 있다.

Open : 저장된 모델을 불러올 수 있다.

Save : 현재 모델을 저장할 수 있다.

Save As.. : 현재 모델을 다른 이름으로 저장할 수 있다.

Export to JPEG : 현재 모델을 jpg 파일로 저장을 할 수 있다.

Print : 현재 모델을 Print 할 수 있다.

Close : 현재 Internal Window 를 Close 한다.

Exit : 현재 프로그램을 종료 시킨다.

Edit menu

Copy, Cut, Paste 기능은 아직 안된다.

Delete: 현재 Select 되어진 Component 를 지워준다.

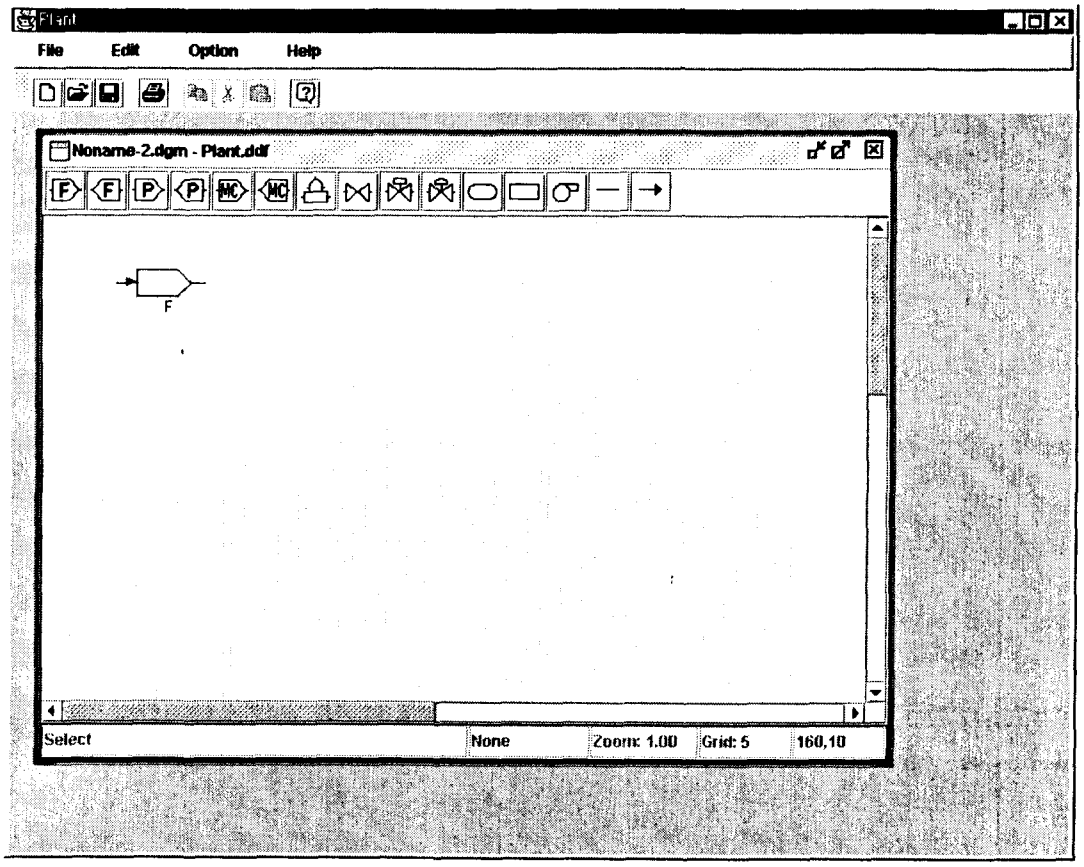
Select All : 현재 window 상에 있는 모든 Component 를 Select 해 준다.

Option Menu

Grid : 현재 Window 의 Grid 를 셋팅할 수 있다.

Zoom : Zoom in, Zoom out 을 할 수 있다.

3. 사용 방법



1. 각각의 내부 윈도우 안에 있는 메뉴바에서 그리고 싶은 하나의 Component 를 왼쪽 버튼을 클릭함으로써 선택을 한다.

2. 이렇게 선택을 하고 나면, 그 Component의 외형이 파란 점선으로 보이면서, 마우스를 움직일 때 따라 다닌다.

3. 자신이 원하는 곳에 왼쪽 버튼을 클릭함으로써 Component 를 완성시킨다. (단, 그리는 것을 취소시킬려면, 오른쪽 버튼을 클릭하시면 된다.)

4. 이렇게 자신이 원하는 Component 들을 적당한 위치에 놓음으로써 그림을 그리게 된다.

5. 그리다가 그림을 move, resize 하고 싶으실 때에는 왼쪽 버튼을 이용해서 이미 그려진 하나의 Component 를 선택하게 됩니다. 그러면 그 선택되어진 Component 는 Control point 를 보여주게 된다.

6. 십자가 표시의 커서가 나타날 때에는 move 가 되고, Control Point 위에서는 손모양의 커서가 나타나게 되는데, 이 때에는 resize가 된다.

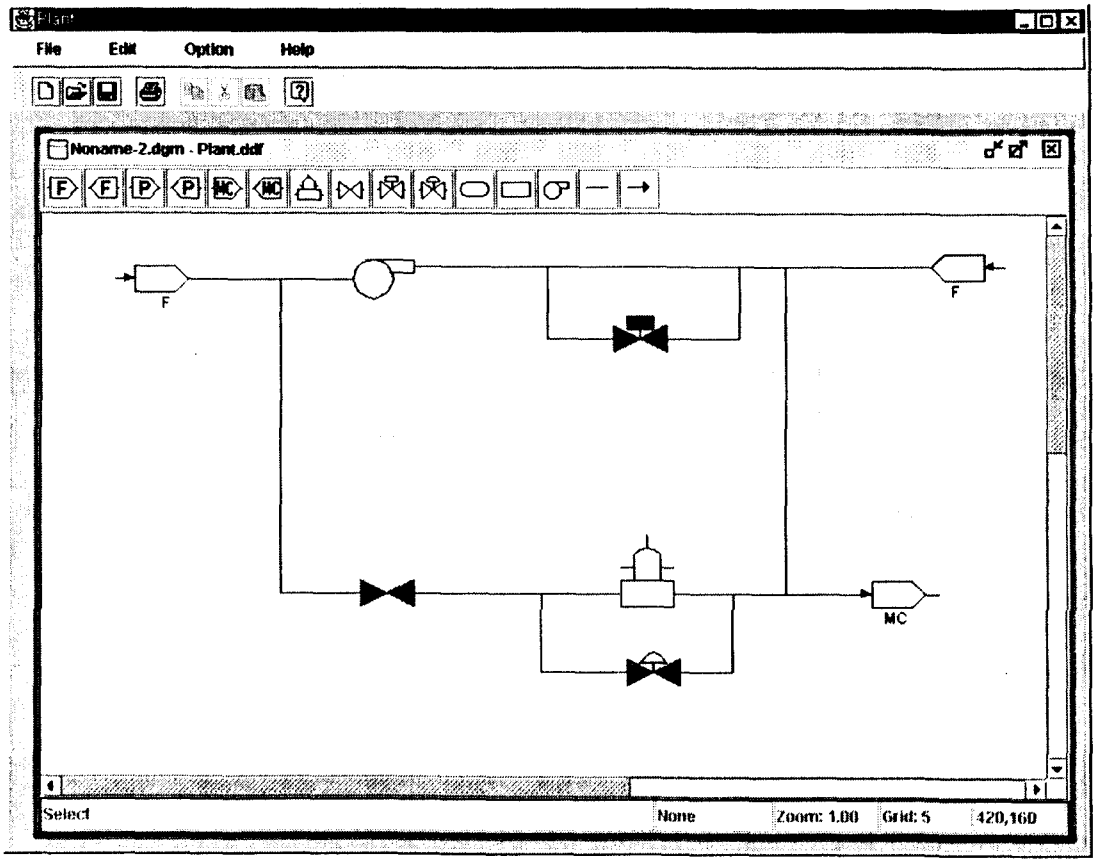
7. move, resize 둘다 왼쪽 버튼을 이용해서 완료가 되고, 취소하시고 싶으면, 오른쪽 버튼을 누르시면 된다.

8. 그리고 Canvas 상에서 오른쪽 버튼을 누르시면, System main menu 가 나타나게 된다. 여기서 Canvas 의 기본 색, 바탕 색등을 선택할 수 있고, 각각의 Component들도 선택하실 수 있다.

9. Select 되어진 Component 위에서 오른쪽 버튼을 누르면, Component 메뉴가 나타나게 된다. 여기서는 Component 의 line color, line width, line type, fill color 등을 선택할 수 있다. 그리고 각각의 Component 들의 Attribute 등을 셋팅해서 그

Component 가 가질 수 있는 값들을 정해서 밖으로 Display 할 수도 있다.

10. 이렇게 만들어진 것은 모든 Component 들을 다 그리고 각각의 Attribute 등을 셋팅하면, 다음과 같은 그림을 볼 수 있다.



4. UCI 위탁 연구 내용 및 결과

당해 년도 UCI의 연구 개발 목표는 다음과 같다.

- RTS modeling 기법을 실시간 객체지향형으로 더욱 공고히 확립
- 대규모 TMO형 real-time simulator를 더욱 효율적으로 신속히 execute 할 수 있는 분산 병렬 처리 engine 구축

연 구 내 용	연 구 결 과
Real-time simulation을 효과적으로 지원하는 근년에 개발된 객체지향 model 인 TMO를 한단계 더 보강 하였다. 보강된 부분은 programmable data field channel (DFC) 를 삽입한데 있고 필요한 execution engine 확장을 이룩 하였다.	TMO 에 programmable data field channel (DFC) 를 삽입 보강한 것은 자연스럽고 효율적으로 multicast facility를 활용하게 하여 Real-time simulation 의 효율을 증대시키는 효과가 있다.
TMO network 형태로 이루어진 real-time simulator에서 인접 TMO간에 data sharing을 효율적으로 하여 분산 처리의 효과를 극대화 하는 방법을 확립하였다.	Real-time simulator 의 분산 처리에 있어서 가장 어려운 문제는 분산 객체간에 message exchange 나 data sharing을 극소화하고 효율적으로 수행하는데 있다. 이 분야에서 TMO에 근간을 둔 방법은 획기적인 향상을 약속한다. 이번에 확립된 인접 TMO간의 data sharing을 효율적으로 하여 분산 처리의 효과를 극대화 하는 방법은 학문적 및 실용적 의의가 매우 크다 하겠다.

TMO 의 execution engine을 Windows NT 바탕으로 개발 하였으며 건국대 와 외국어대 의 TMO execution engine 개발을 보좌하였다.	TMO 의 execution engine을 Windows NT 바탕으로 두가지 개발하였는 바 하나는 CORBA communication 환경에 적합한 것이며 현재 동작하고 있고 다른 하나는 NT 의 UDP communication facility를 활용하는 것으로서 수행속도가 보다 빠르다. 건국대 와 외국어대 의 TMO execution engine 개발을 보좌하였고 이는 현재 동작하고 있다.
TMO형 real-time simulator 구축 기법의 다른 측면에서의 demo 로서 고속도로 및 도시교통망의 미시적 real-time simulator를 Windows NT machine network 상에서 동작 하도록 구현 하였다.	TMO형 real-time simulator 구축 기법의 다른 측면에서의 demo 와 Windows NT 바탕으로한 execution engine의 demo 로서 고속도로 및 도시교통망의 미시적 real-time simulator (DOFS -- Distributed Object-based Freeway Simulator) 를 구현 하였다.

제 3 절 3차년도 연구개발수행 내용 및 결과

1. 연구개발 목표

당해 년도의 연구개발 목표는 **ASADAL** 방법론과 **CASE Tool**을 이용하여 선박 레이다 시스템을 개발하는 것이다. 목표 시스템인 선박 레이다 시스템은 선박 운전자들을 위한 훈련 프로그램으로서, 훈련자가 실제 상황과 거의 유사한 현실감을 느끼게 하기 위하여 VR 인터페이스를 필요로 하고, 실시간 제약이 있으며, 시스템이 거대하여 효율적인 성능을 위해 분산 배치를 필요로 하는 등 본 과제를 통해 얻은 연구 결과를 적용하기에 적절한 목표 시스템이다. 당해 년도에서는 이 선박 레이다 시스템을 구축하기 위하여 특히 객체 지향적인 설계에 초점을 맞추어 방법론과 도구를 강화하였다. 당해 년도의 연구 개발 목표를 세분화하여 정리하면 다음과 같다.

1. 선박 레이다 시스템의 사용자 요구 사항 명세

   - 선박 레이더 시스템에 대한 다양한 요구 사항을 파악하고 ASADAL의 명세 도구인 DFD와 Statechart 를 이용하여 명세한다.
   - 명세된 결과를 Graphical Simulator 를 이용하여 분석한다.

2. 검증된 명세를 기반으로 TMO 기반의 객체 지향적 설계

   - 명세된 결과를 바탕으로 Performance, Complexity, Real-time property 등을 고려하여 TMO 기반의 객체 지향적인 설계를 한다.
   - 설계 객체들을 분산 환경에 배치한다.

3. 객체 지향적 설계를 바탕으로 자동 코드 생성

- 객체 지향적인 설계를 토대로 TMO 기반의 코드를 자동 생성한다.

4. 선박 레이다 시스템의 가시화

- VR(Virtual Reality) Interface를 구축하고, 자동 생성된 코드와 연동하여 가시화한다.

5. 병렬성을 고려한 디자인 모델 개발

- 시스템의 병렬성을 고려한 디자인 모델(즉, 태스크 기반의 디자인 모델)을 명세로부터 구축하는 방법을 개발한다.
- 태스크 기반의 디자인 결과를 바탕으로 실시간 OS 위에서 실행되는 코드를 자동 생성한다.
- 자동 생성된 결과를 수행함으로써 디자인 결과를 분석할 수 있는 도구를 개발한다.

6. ASADAL 도구 보완

- 1,2차년도를 통해 만들어진 ASADAL 도구의 사용자 인터페이스를 개선한다.
- 효과적인 시뮬레이션을 위하여 외부 사용자 인터페이스를 빠르게 제작하고 명세와 연결하여 시뮬레이션 할 수 있는 프로토타이핑 도구를 개발한다.

7. 전력 시스템 모델링 도구 보완

- 2차년도에 제작한 전력 시스템 모델링 도구를 보완한다.

전년도 계획과 비교하여 변동 사항

전년도 계획과 비교하였을 때, ASADAL 방법론과 CASE Tool 을 이용하여 선박 레이다 시스템을 개발한다는 큰 목표에는 변함이 없지만, 세부적으로 몇몇 내용이 변경 및 추가되었다. 아래에 변경 및 추가된 내용과 그 사유에 대하여 기술하였다.

* 변경 사항

- 객체 지향적 설계 모델에 대한 평가 도구

  당초 계획은 설계된 모델을 시뮬레이션 하면서 반응 시간, 노드 부하 등을 검사하여 디자인 결과를 평가하는 도구를 만들 계획에 있었으나, 건국대학교에서 전년도에 개발한 TMO의 수행 평가 도구와 대부분의 내용이 동일하기 때문에 연구의 중복성을 피하기 위하여 계획을 수정하였다. 수정된 내용은 위의 당해년도 목표의 5번 항목으로, TMO 기반의 디자인 모델이 아닌 병렬성을 고려한 태스크 기반의 디자인 모델을 만들고 이를 실시간 OS 위에서 수행한 결과에 대한 디자인 평가 도구를 개발하였다.

* 추가 사항

- ASADAL 도구 보완(6번 항목)

  개발되어온 도구의 사용상의 불편함이 지적되어 사용자 인터페이스를 개선하는 것과, 외부 사용자 관점의 시뮬레이션의 필요성이 지적되어 프로토타이핑 도구를 개발하는 내용을 추가하였다.

- 전력 시스템 모델링 도구 보완(7번 항목)

  2차년도에 개발된 모델링 도구로부터 전력 연구원의 요청에따라 필요한 사항을 수정 및 보완하였다.

2. 연구 수행의 이론적, 실험적 접근 방법

선박 시뮬레이터의 개발은 내부 컨트럴러의 개발은 Linear Sequential Model(Waterfall Model)을 사용하여 ASADAL 방법론으로 명세 및 설계, 구현 과정을 거쳤으며, VR 인터페이스의 경우 개발을 요청한 측에서도 정확한 요구 사항을 파악하지 못하고 있었기 때문에 Prototyping Model을 사용하여 피드백을 받으며 진행하였다. 그리고 선박 시뮬레이터에 포함되는 지도의 데이터베이스의 경우, 따로 분리해 내서 개발이 가능하였으므로, 재사용을 위한 Component의 개념을 활용하여 컨트럴러의 개발과 인터페이스의 개발과 별도로 Component화하여 개발하였다.

TMO 기반의 자동 코드 생성기와 태스크 기반의 디자인 모델은 기존의 객체지향 방법론, Architecture 등의 관련된 연구 결과를 전반적으로 검토하고, 방법이 추구하는 바의 차별성을 확인해 나갔으며, 구현에 있어서는 Spiral Model 을 사용하여 중요한 부분에 비중을 두어 개발해 나갔다. 현재 이들의 연구 결과는 논문으로 준비 중에 있고, 논문으로 발표하여 피드백을 받을 예정이다.

3. 연구 내용 및 결과

여기서는 당해 년도 연구개발 목표에 따른 자세한 연구 내용과 그 결과를 기술하도록 하겠다.

가. 선박 레이다 시스템의 개발

여기서는 효성 데이터 시스템과 함께 개발한 선박 레이다 시스템의 개발내용과 그 결과를 소개하도록 하겠다.

선박 레이다 시스템은 먼저 요구 사항을 개념적으로 분석하고, 내부 컨트릴 부분을 ASADAL 명세 방법으로 명세하였으며, 데이터베이스와 관련한 컴포넌트를 따로 제작하고, VR 인터페이스를 제작하였다. 이 결과를 2차년도에 개발한 TMO 디자인 도구 및 코드 생성기를 보완한 도구로 디자인하고 최종 코드를 생성하였다. 각각의 과정에 대한 방법 및 결과를 아래에서 차례대로 기술하였다.

요구 사항에 대한 General Description

- 시스템은 특정 항구에 대한 훈련 데이타(이하 EXDB)를 가지고 있으며 instructor는 trainee들이 훈련하게 될 상황을 설정하고, 설정된 상황에 따라 trainee들이 훈련한다.
- 실제 해도를 바탕으로 EXDB를 개발한다.
    - EXDB의 개발은 radar 및 navigation을 위한 DB와 visual system를 위한 DB 생성을 의미한다.
    - EXDB 개발 시 해안선을 중심으로 설계하되 해상 구조물은 무시한다.
    - EXDB는 수심, 해안선 및 해로에 대한 vector구조를 중심으로 한다.
- instructor는 instructor용 control panel(이하 ICP)을 통하여 각종 시뮬레이션 조건 설정 및 훈련 상황을 감독한다.
    - ICP는 훈련 상황을 설정할 수 있는 simulation controller, 선박들의 움직임을 보여주는 2D Viewer(radar display)를 가지고 있다.
    - Instructor는 simulation controller를 통하여 기상 조건, 각 선박들의 초기

위치 및 속도 등을 설정한다. 또한, 훈련 중에 기상 조건 등을 동적으로 변경할 수 있다.

- 2D Viewer(radar display)는 훈련 중인 각 선박들의 위치를 EXDB를 바탕으로 보여주는데, 넓은 해역을 자유롭게 panning 및 zooming할 수 있어야 한다.

- Trainee는 trainee용 control panel(이하 TCP, 그림 1)을 이용하여 실제 선박 운항과 근접한 환경에서 훈련한다.
  - TCP에는 radar display와 선박 운항과 직결된 steering stand, engine telegraph 그리고 실제 선박을 운항하는 것과 같은 시야를 제공하기 위한 3D viewer가 있다.
  - Trainee용 radar display는 EXDB와 simulation을 바탕으로 주위의 해안선 및 다른 선박들의 위치를 표시한다. 또한, 훈련 중인 선박을 중심으로 한 표현과 지도를 중심으로 한 표현이 가능하고 range도 자유롭게 변경 할 수 있어야 한다.
  - Steering stand를 이용하여 선박의 방향을 조절한다.
  - Engine telegraph를 이용하여 선박의 속력을 조절한다.
  - 3D viewer에서 묘사해야 할 것들은, 단순화된 지형, 부표, 등대, 바다 및 선박 등이다. 또한, 다양한 기상조건 등을 효과적으로 표현해야 한다.
  - 충돌 검사의 범위는 선박들간의 충돌과 선박과 해안과의 충돌이다.

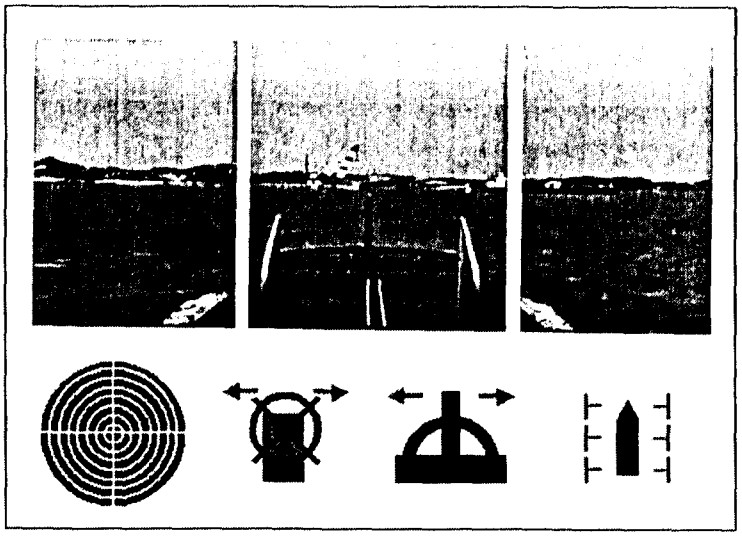


그림 1

- Simulation은 실제 선박의 운동역학을 기반으로 단순화된 모델을 사용한다.
    - 선박의 외부 조건으로는 바람과 파도, 수심, 해안선으로부터의 거리 등을 고려하고, 해류는 무시한다.

- 다음 그림 2는 시스템의 개괄적인 모델이다.

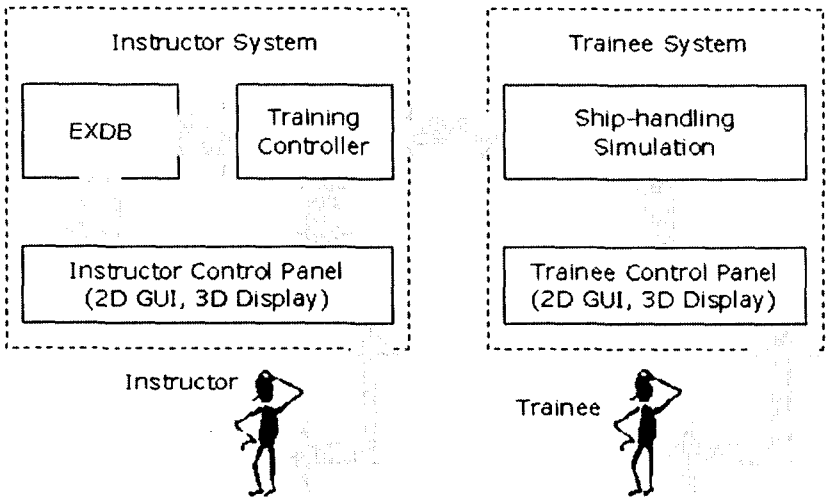
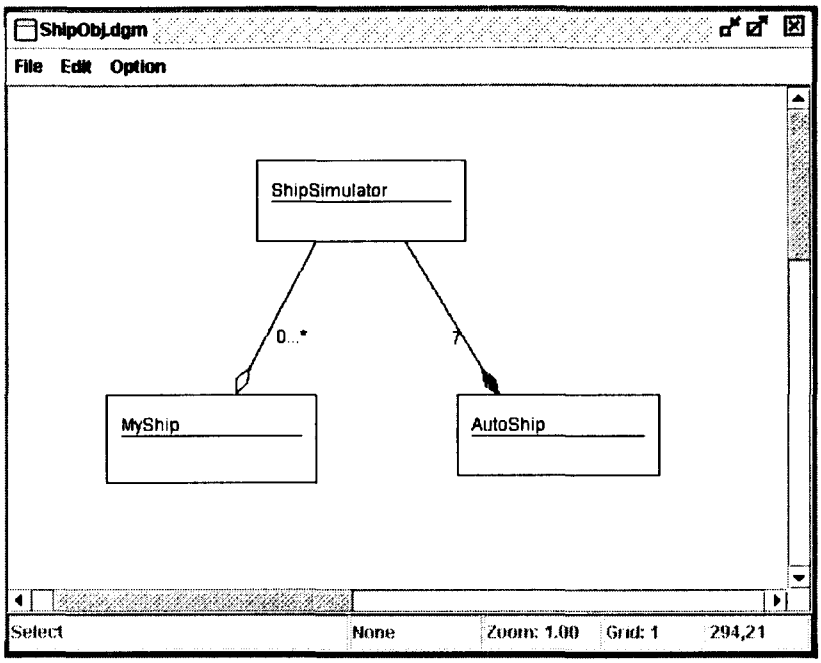


그림 2

ASADAL 명세

ASADAL 명세는 전체 시뮬레이션을 관장하는 ShipSimulator와 훈련 참여자의 시뮬레이션을 제어하는 MyShip, 그리고 지도상에서 random하게 움직이는 AutoShip 세 개의 객체로 나누어져 개발되었다.

먼저 모델에 대한 Class Diagram은 아래와 같다.



위의 그림에서 보이는 것처럼 Class는 ShipSimulator, MyShip, AutoShip으로 구성되어 있다. 그림에서 ShipSimulator와 MyShip관계의 속이 빈 다이아몬드는 Aggregation 관계이고, ShipSimulator와 AutoShip관계의 속이 찬 다이아몬드는 Composition 관계이다. Aggregation과 Composition의 관계는 객체의 Dynamics를 기준으로 다음과 같이 분리될 수 있다.

- Aggregation

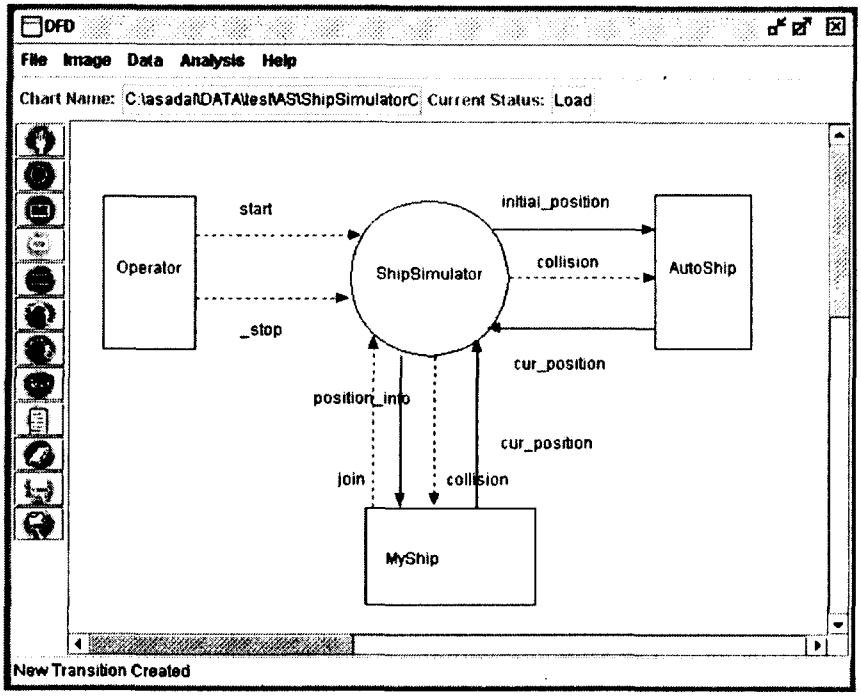
  관계가 Dynamic하게 맺어질 수도 있고 끊어질 수도 있는 것이다. 관계의 대상의 생명주기는 관계의 주체의 생명주기와 상관없다.

- Composition

  관계가 Static하게 결정되어 있는 것이다. 관계의 대상의 생명주기는 관계의 주체의 생명주기와 같다.

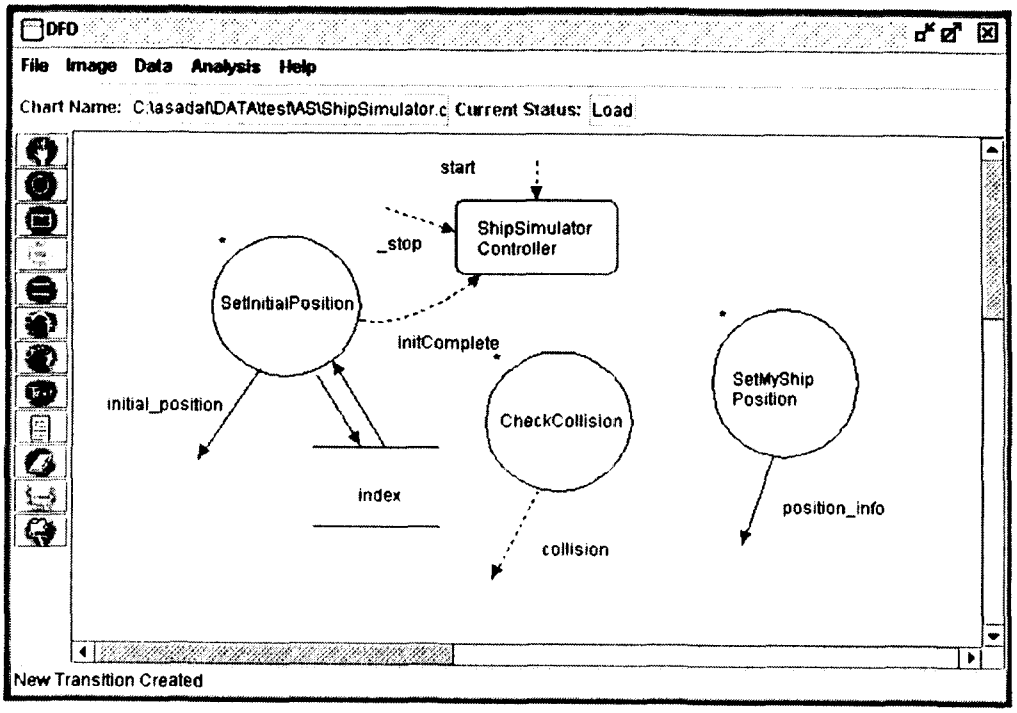
위 예제에서 AutoShip은 ShipSimulator가 생성이 될 때 7개가 생성되어 연결되며, MyShip은 정해지지 않은 개수가 Dynamic하게 연결된다.

Class Diagram을 작성한 후, 각각의 Class에 대하여 DFD와 Statechart로 정형적으로 명세하였다. AutoShip과 MyShip은 뒤에 VR 가시화 시스템에서 행위와 기능으로 설명하기로 하고, 여기서는 ShipSimulator의 ASADAL 명세를 살펴보기로 한다.

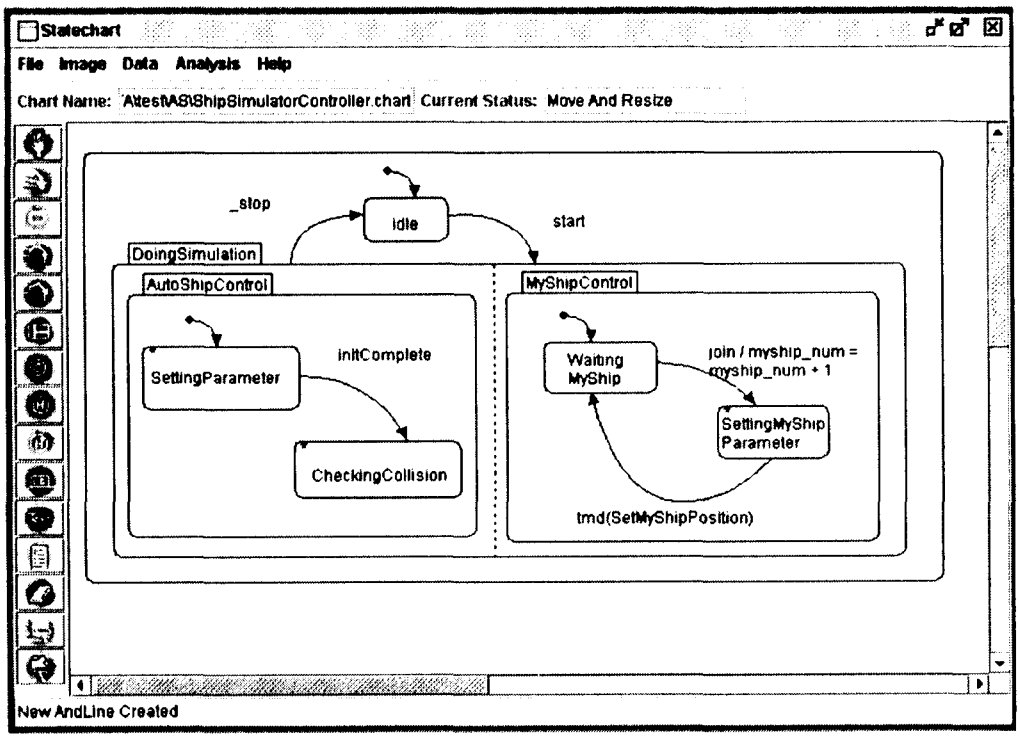


위의 그림은 ShipSimulator에 대한 Context Diagram이다. ShipSimulator는 Simulation Operator로부터 시작/종료의 이벤트를 받아처리하고, AutoShip Class에게 초기 위치와 충돌 정보를 제공하며 AutoShip Class로부터는 현재 위치를 받는다. 그리고, MyShip Class로부터 Simulation 참여를 요청하는 이벤트를 받고, 각종 위치 정보

와 충돌 정보를 준다. MyShip Class는 현재 자신의 위치를 보내준다.

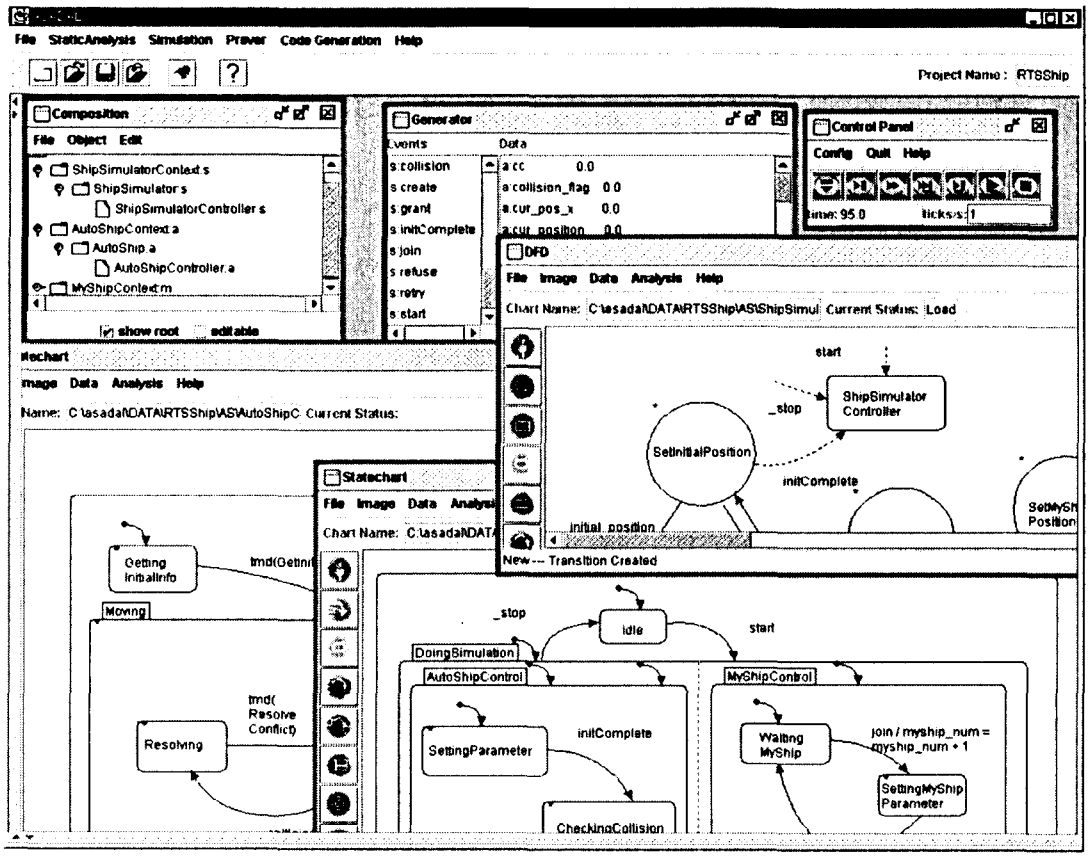


위의 그림은 ShipSimulator의 첫 레벨 DFD이다. ShipSimulator는 AutoShip의 초기 위치를 결정하는 프로세스, MyShip의 초기위치를 결정하는 프로세스, 충돌검사를 하는 프로세스로 구성되어 있다.



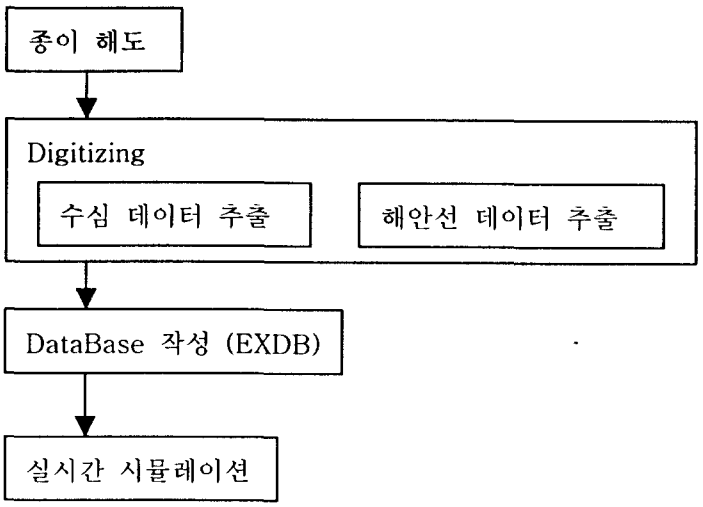
위의 그림은 ShipSimulator의 첫번째 레벨 DFD에 대한 Controller이다. ShipSimulator는 idle 상태에서 "start" 이벤트에 의해 구동되기 시작하며, Concurrent하게 AutoShip과 MyShip을 관리하는 상태로 나뉜다. AutoShip은 시작 시에 모든 배가 생성되므로, 초기값을 주고, 충돌 검사를 하는 상태로 바뀐다. MyShip의 경우, 언제든 join 이벤트가 들어올 수 있으므로, 초기값을 주고 다시 기다리는 상태로 돌아온다.

이러한 명세 결과들을 바탕으로 해서 명세의 결과를 검증하기 위해서 ASADAL Simulator를 사용하였다. ASADAL Simulator는 필요한 event를 보내거나, data 값을 설정할 수 있고, State나 Process의 활성화 여부를 곧바로 확인할 수 있다. 다음 그림은 Simulation을 하는 화면을 캡처한 것이다.



EXDB 제작

EXDB제작은 훈련해역에 관한 정보를 갖고 있는 종이 해도를 선박 시뮬레이터를 위한 DB로의 제작을 말한다. 이 EXDB에는 radar display를 위한 data, 수심에 대한 data등을 포함한다. 이번 과제에서는 상용화된 EXDB 저작 tool을 사용하지 않고 직접 tool을 제작하여 포항 근해의 EXDB를 저작하였다. EXDB의 전체적인 저작 과정은 다음 그림과 같다.



***EXDB** 저작과정*

1. Digitizing

1) 수심 데이터 추출

- 사용 S/W는 자체 개발한 툴이다.

- 종이 해도상에 수심 값이 표시되어 있는 위치에 마우스를 클릭하여 가로, 세로 위치를 읽어 들인다.

- 위의 값을 위도, 경도 값으로 변환하고, 수심 값은 키보드를 통해 수치화한다.

- 위도, 경도, 수심 값을 file에 저장한다.

2) 해안선 데이터 추출

- 1)에서 사용한 S/W에서 해안선에 굴곡이 일어나는 부분을 마우스로 클릭한다.

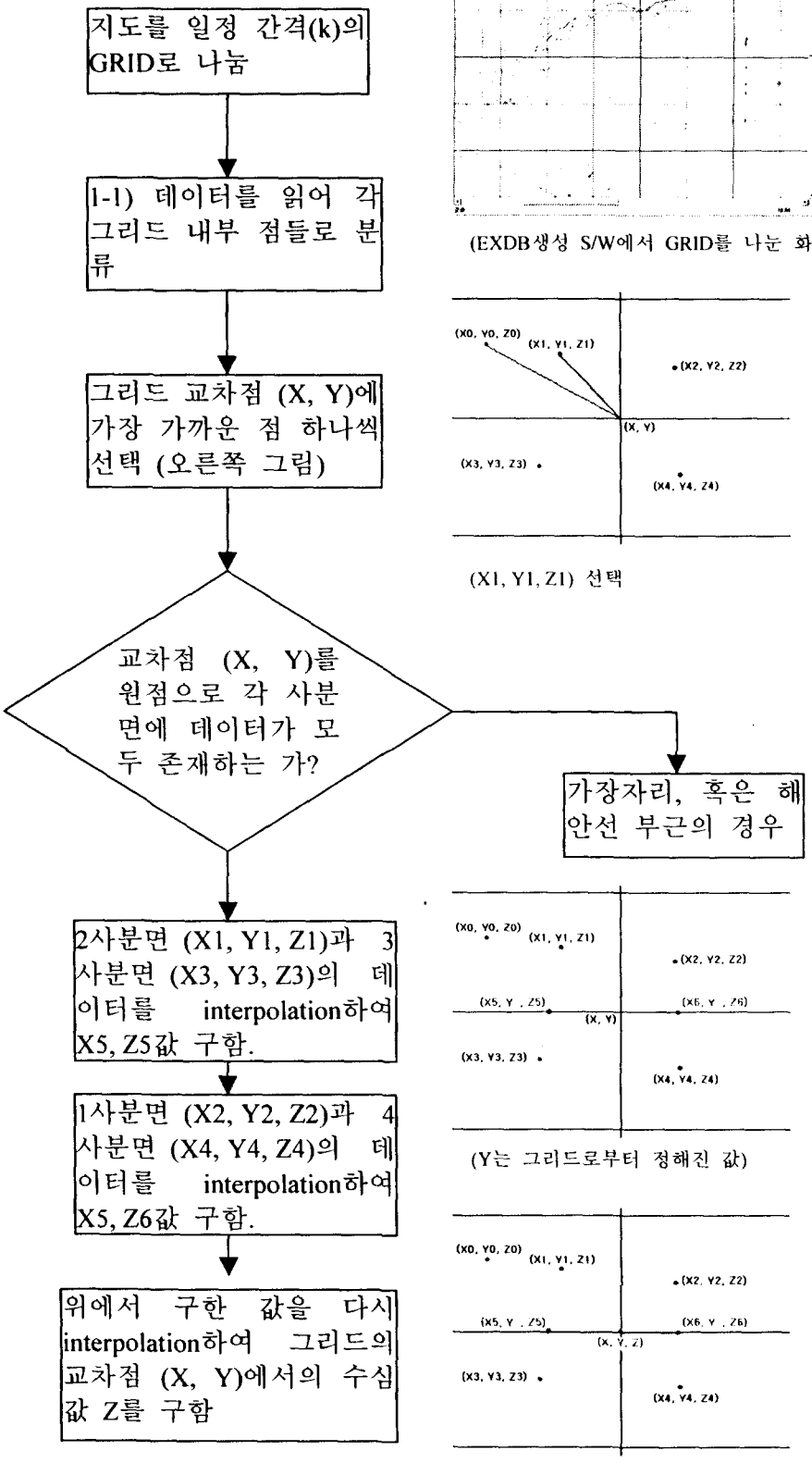
- 위에서 변환한 위도, 경도 값을 하나의 vertex로 보고 polygon을 만들어 file 로 저장한다.

- 하나의 지도에 여러 개의 독립적인 그룹으로 볼 수 있는 해안선이 있는 경우 그 개수만큼의 polygon을 만든다.

2. DataBase 작성

- 종이 해도에서 얻을 수 있는 데이터는 원해의 경우 간격이 넓고, 근해(해안선 주변)의 데이터는 조밀한 형태이다. 따라서 이 두 경우를 나누어 두개의 데이터베이스를 작성한다.

1) 원해


지도를 일정 간격(k)의 GRID로 나눔
1-1) 데이터를 읽어 각 그리드 내부 점들로 분류
그리드 교차점 (X, Y)에 가장 가까운 점 하나씩 선택 (오른쪽 그림)
교차점 (X, Y)를 원점으로 각 사분면에 데이터가 모두 존재하는 가?
가장자리, 혹은 해안선 부근의 경우
2사분면 (X1, Y1, Z1)과 3사분면 (X3, Y3, Z3)의 데이터를 interpolation하여 X5, Z5값 구함.
1사분면 (X2, Y2, Z2)과 4사분면 (X4, Y4, Z4)의 데이터를 interpolation하여 X5, Z6값 구함.
위에서 구한 값을 다시 interpolation하여 그리드의 교차점 (X, Y)에서의 수심값 Z를 구함
(EXDB생성 S/W에서 GRID를 나눈 화면)
(X0, Y0, Z0)
(X1, Y1, Z1)
(X2, Y2, Z2)
(X, Y)
(X3, Y3, Z3)
(X4, Y4, Z4)
(X1, Y1, Z1) 선택
(X5, Y , Z5)
(X6, Y , Z6)
(Y는 그리드로부터 정해진 값)
(X, Y, Z)

* 가장자리 혹은 해안선 부근

- 3개의 면에 데이터가 존재하는 경우

  세 데이터로 삼각형을 이루게 하였을 경우 빗변을 이루는 두 점을 보간(이한 inerpolateoin) 하고 다시 그 결과와 마주보는 정점을 interpolation한다.

- 2개의 면에 데이터가 존재하는 경우

  2개의 데이터를 interpolation하고 이웃 하는 격자(grid)의 교차점 깊이 정보를 사용하여 다시 interpolation 한다.

- 1개의 면에 데이터가 존재하는 경우

  데이터가 존재하는 방향으로 이웃 하는 격자의 교차점의 깊이 정보를 사용하여 interpolation 한다

- 모두 존재하지 않는 경우

  육지이므로 깊이(depth)를 -1로 저장한다.

2) 근해

- 원해에서의 방법과 동일하다. 그러나 원해에서는 하나의 격자 내에 하나 이상의 데이터가 존재하도록 크기를 정하지만 근해의 경우 한 격자 내부에 지나치게 많은 데이터가 들어가지 않는 범위에서 크기를 결정한다.

- 원해와 근해를 구분하는 방법은 해안선과의 거리를 사용한다.

*** Depth decision**

임의의 위치 (x, y)가 주어지면 가장 가까운 격자의 교차점 네 개의 값을

DataBase 작성과 동일한 방법으로 interpolation 한다.

*** Collision with Terrain**

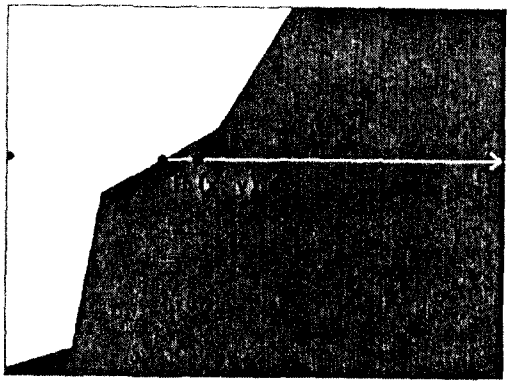
해안선은 하나 이상의 polygon으로 구성되어 있으므로 임의의 위치 P(x, y)에서 해안선에 충돌하는지를 검사할 때는 Even-Odd Rule을 이용한 polygon inside/outside check를 한다.

임의의 점 P와 수평방향의 벡터 r로 이루어진 선분

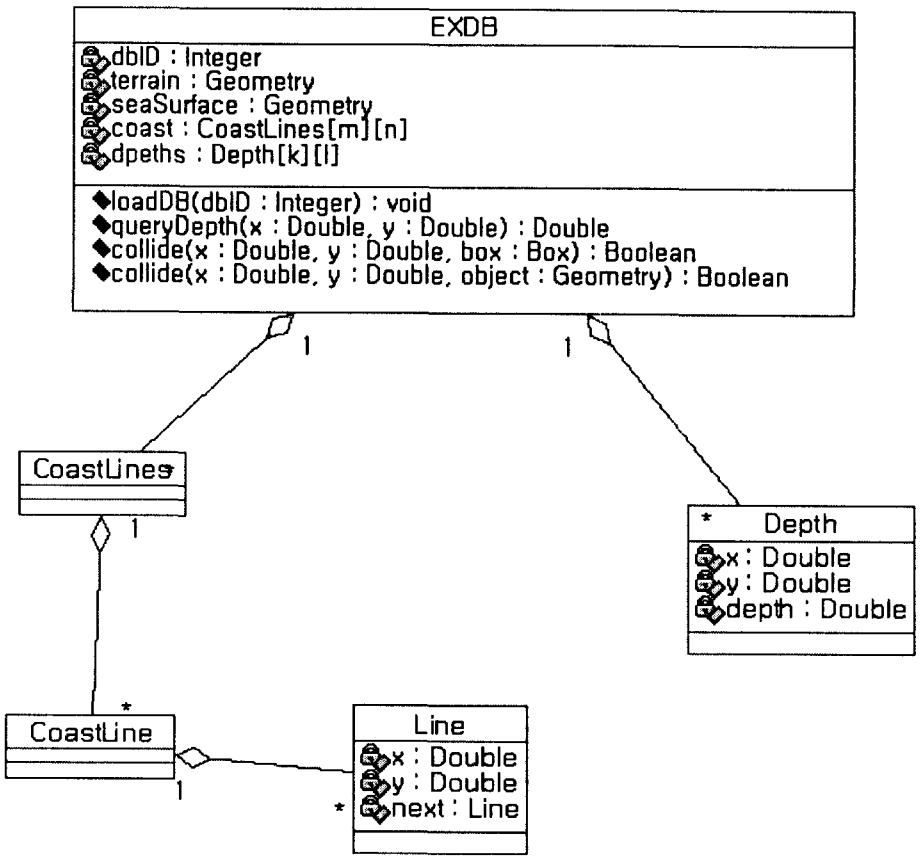
$$l(t) = p + tr$$

에서 선분 l이 해안선 다각형에 몇 번 교차되는지를 살펴보아 짝수 번 교차되면 바다에 있는 점이고, 홀수 번이면 육지의 점이다.

아래 그림에서의 경우 점 P(x, y)는 두 번(짝수 번) 교차되므로 바다에 있는 점이다.



3. Design

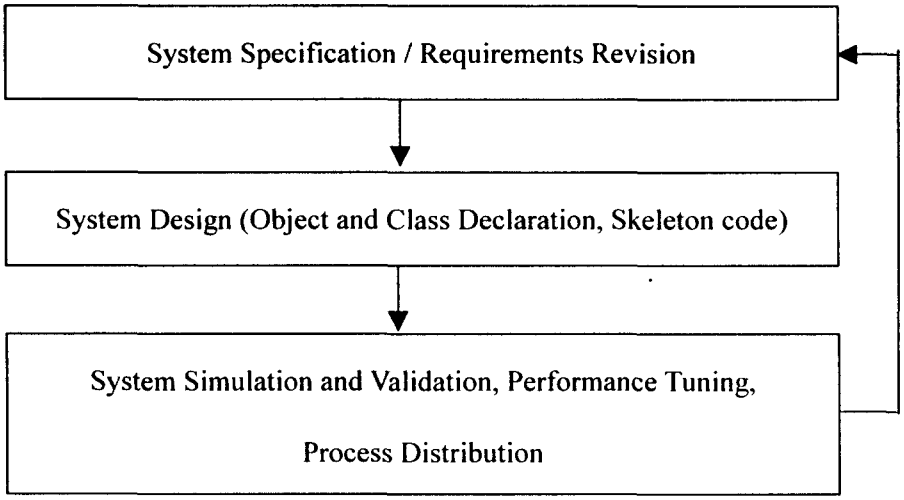


VR 가시화 시스템

VR 가시화시스템의 제작은 기존의 ASADAL/PROTO의 방법론을 기반으로 한다. ASADAL/PROTO에서는 가상 공간에 존재하는 객체들의 형태/기능/행위를 concurrent하게 modeling하되 top-down approach를 적용한다는 것이 핵심이다. 그리고, ASADAL/PROTO 방법론은 선박 시뮬레이터와 같은 VR system의 특성을 고려하여 다음 그림과 같은 spiral process에 따라 진행된다.

그림의 세 과정을 되풀이 하면서 초기에는 전체적인 시스템의 행위에 중점을 두고, 다음엔 VR system을 위한 performance, interaction model 및 VR object modeling, 마지막 단계에선 presence와 special effects를 고려하게 된다. 이것을 정리하면 다음과

같다.

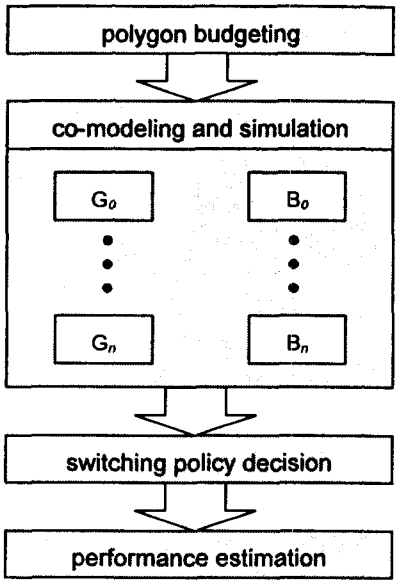


1 단계 : Object / Gross Behavior / System Architectrure

2 단계 : Performance / Task Decomposition / Interaction Model / Form, Function, Behavior models Refeinement

3 단계 : Presence / Special Efflects

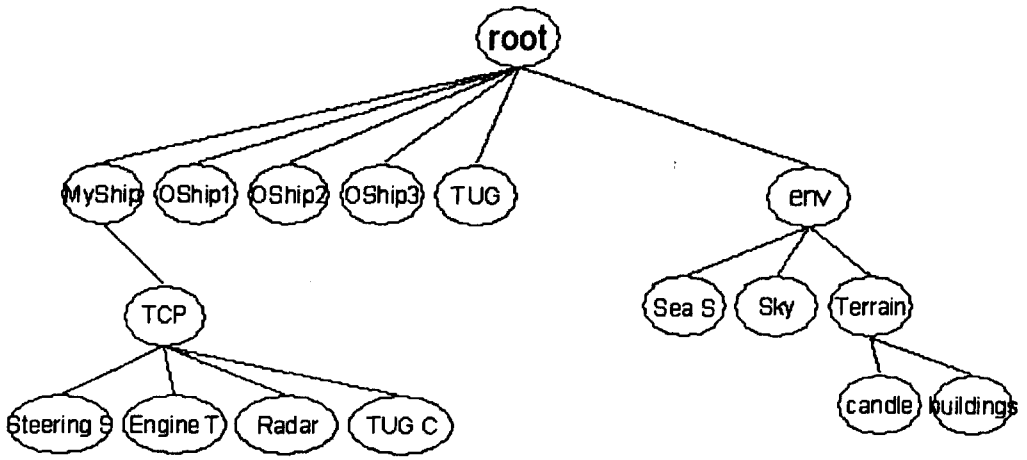
전체적인 VR System을 위한 modeling은 위와 같이 이루어지지만, 가상환경에 존재하는 VR 객체들은 형태/행위/기능을 동시에 고려하면서 LOD Engineering을 통하여 이루어진다. LOD Engineering에서는 고전적인 LOD modeling과는 달리, 간단한 model로부터 복잡한 model로 VR 객체들을 만들어 가는 과정에서 중간 단계의 model들을 LOD model로 채택하는데, LOD modeling에 top-down approach를 적용한 것이라고 할 수 있다. LOD Engineering의 전체적인 과정은 다음 그림과 같다.



LOD Engineering의 전체적인 과정

- Scene Graph

  VR object들을 modeling하기 전에 VR system에 필요한 object를 식별하기 위해 선 우선 다음과 같이 scene graph를 통하여 object들을 식별하고, 또 object들의 계층 관계를 알아낸다.

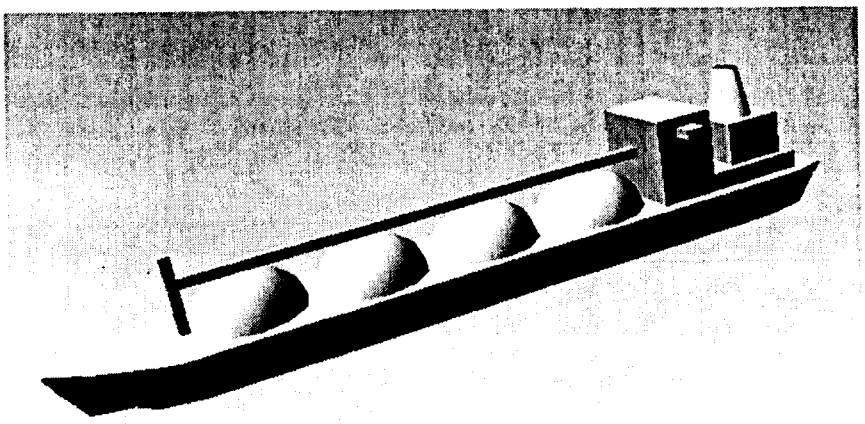


- Polygon Budgeting (target system's rendering power : 5000 ± 1000)

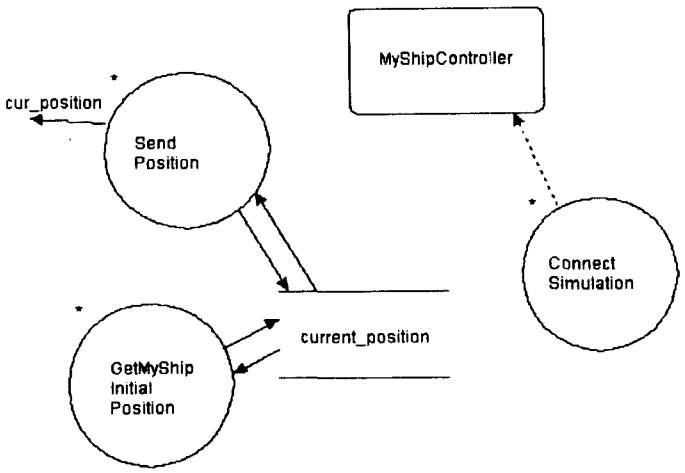
Target system의 graphics 성능을 고려하여 적당한 polygon수를 계획하기 위하여 polygon budgeting을 한다. 아래 표는 target system이 frame당 약 5000개의 polygon을 처리할 수 있다고 가정한 것이다.

Object		**Objects**	**Total Polygons**	**Polygons per Object**
MyShip		*1*	*500*	*500*
OtherShip1	LOD0	*10*	*1,000*	*100*
	LOD1	*5*	*1,500*	*300*
	LOD2	*2*	*1,200*	*600*
Sky		*1*	*100*	*100*
Terrain		*1*	*1,000*	*1,000*
SeaSurface		*1*	*100*	*100*
Total Polygons		*5,400*		

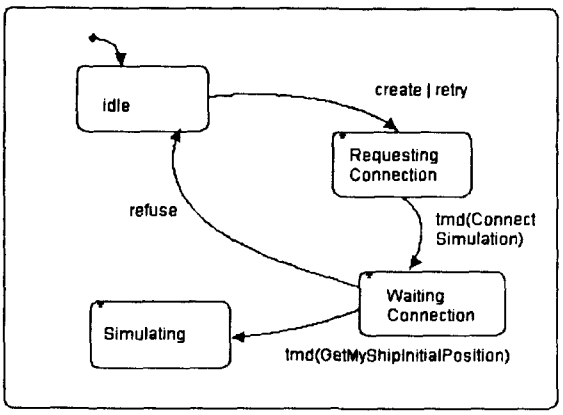
- MyShip (410 triangles)
  - 형태



○ 기능


MyShipController
cur_position
Send
Position
Connect
Simulation
current_position
GetMyShip
Initial
Position

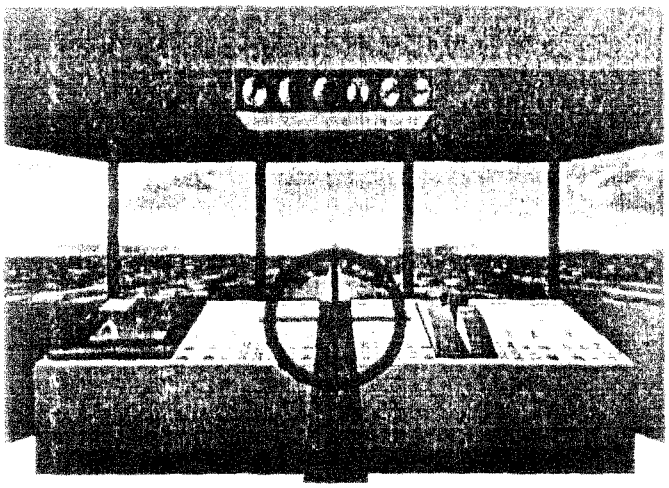
○ 행위



위의 그림은 MyShip의 행위를 표현한 Statechart이다. MyShip은 사용자가 시작한다는 event를 보내면, ShipSimulator에 연결을 시도하고, 연결이 성사되면, Simulation을 시작한다.

o TCP

사용자가 조작하게 될 control panel이다. Control panel에는 방향을 조절하기 위한 steering stand, 속도를 조절하기 위한 engine telegraph, 그리고, radar display를 포함된다.

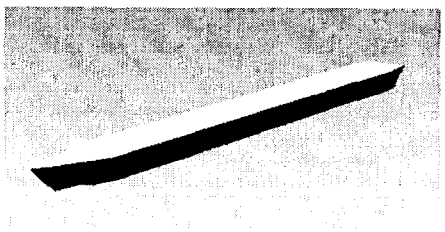


TCP(사용자 control panel)

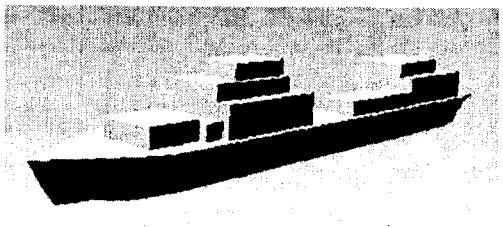
- OtherShip(AutoShip : 309 triangles)

  형태의 경우 3단계의 모델을 적용하고 행위/기능의 경우는 그 요구사항이 간단하므로 1간계의 모델만 적용한다.

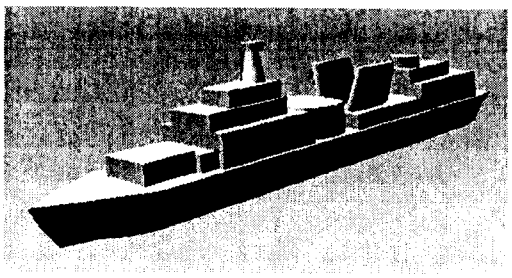
  - 형태
    - LOD1 (65 triangles)



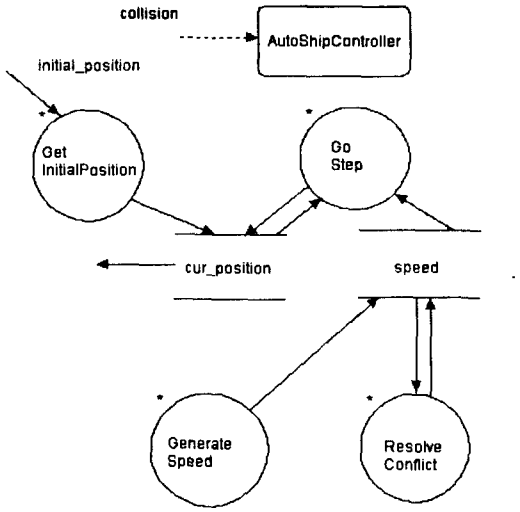
    - LOD2 (132 triangles)



    - LOD3 (309 triangles)

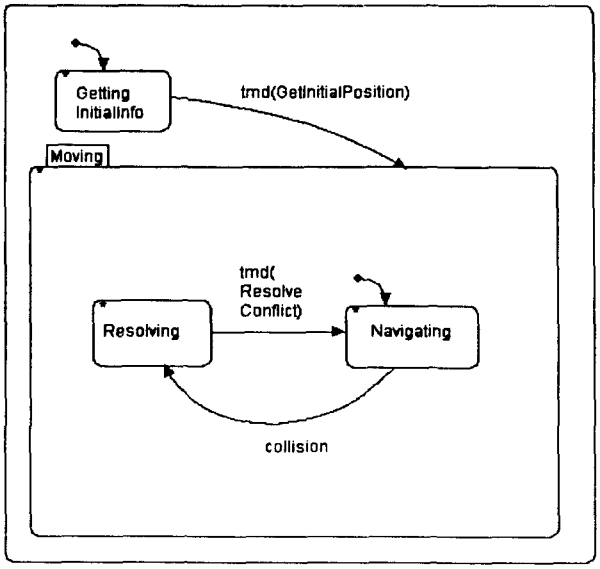


o 기능

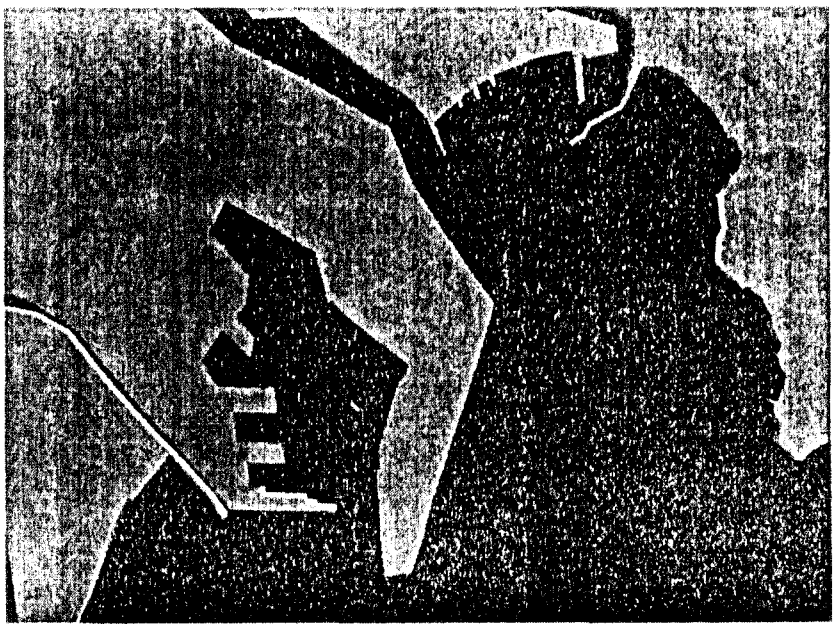


위의 그림은 AutoShip의 주요 프로세스들을 포함하는 DFD 이다. GetInitialPosition은 ShipSimulator로부터 초기값을 받아들여서 Setting하는 프로세스이고, GoStep은 현재의 속도, 각속도, Orientation 으로 다음위치를 계산하여 결정하는 프로세스이다. GenerateSpeed는 random하게 속도 파라미터를 결정하는 프로세스이고, 마지막으로 ResolveConflict는 다른 배와 충돌이 일어났다는 신호가 왔을 때, 후진을 하도록 처리하는 프로세스이다.

- 행위



- Environment
    - Sea and Terrain



- Sky

TMO 기반의 객체 지향적 설계 및 자동 코드 생성

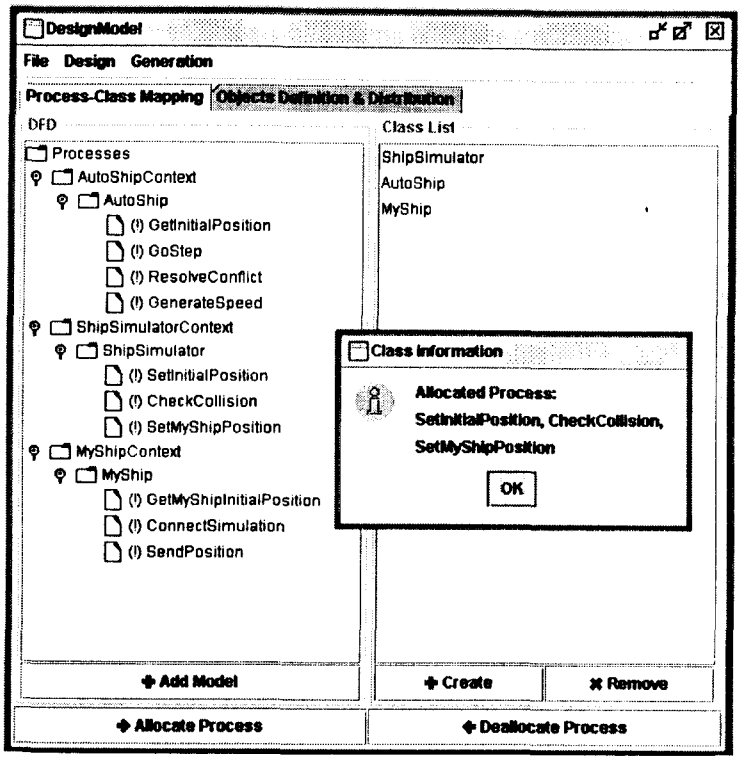
2차년도 과제에서는 Top-Down 방식으로 구성된 ASADAL 명세 결과로부터 어떻게 TMO object를 만들어내고, 코드 생성을 할 것인가에 대한 방법 및 이를 지원하는 도구를 연구개발 하였는데, 이번년도에서는 객체지향적인 명세를 지원하도록 확장하였으며, 분산 노드에 배치할 수 있도록 하였다. 여기서는 2차년도 과제와 비교하여 보완된 방법론의 내용에 대해 선박 시뮬레이터의 디자인 결과를 함께 소개하면서 설명하도록 하겠다.

(ㄱ)디자인 단계의 Class 생성

2차년도의 과제에서는 Top-Down 방식으로 하나의 Context Diagram만을 Load하여 모델을 구성하였으나, 이러한 방식은 여러 개의 객체를 Instantiation하는 등의 방법을 효과적으로 지원하지 못하고, 모델링상의 복잡성을 더하는 문제가 있었다. 이에 따라, 객체 지향적인 명세 방법을 지원하도록 하였다.

요구 분석 단계의 Class는 디자인 단계에서 똑 같은 Class로 할당될 수도 있지만, 여러 개의 Class로 분할되어 할당될 수도 있다. 예를 들어서, 채팅서버를 명세단계에서 하나의 Class로 만들었고, 만일 이 채팅서버 Class가 전체 채팅 방을 관리하는 일과, 메시지를 보내는 두 가지 일을 하도록 명세 되었다고 하자. 그런데 나중에 이 Class로부터 만들어진 객체가 부하가 너무 커서 하나의 분산 노드에서 처리되기가 힘들다면, 이 채팅서버 Class를 채팅 방을 관리하는 것과, 메시지를 보내는 두 개의 Class로 분리하여 서로 다른 분산 노드에 배치하는 것이 가능할 것이다. 즉, 명세 단계에서는 논리적인 의미로 Class를 만들고 명세하게 되지만, 디자인 단계에서는 데이터 전달량, 노드 부하 등의 성능을 고려하여 Class를 분할할 수가 있게 되는 것이다.

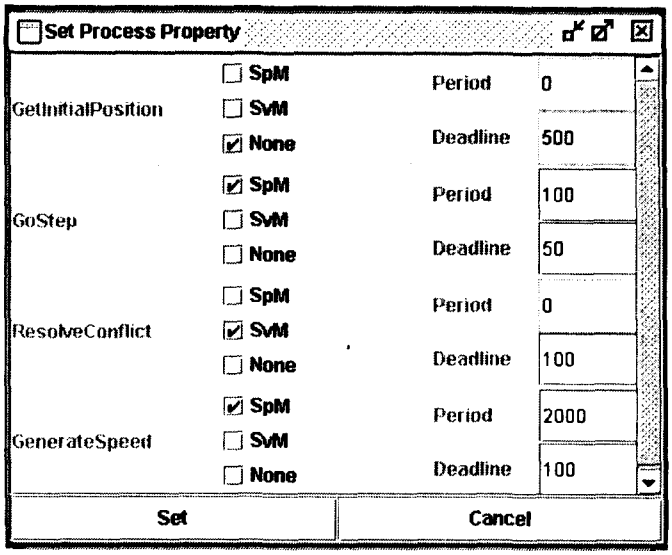
이렇게 Class를 분할 하는 방법은 2차년도 과제에서 Top-Down 방식으로 제작된 명세를 여러 개의 TMO 클래스로 할당하는 방식으로 수행된다. 이렇게 하면, 명세 단계의 Class는 논리적인 관점에서의 Class로 관리하고 디자인 단계의 Class는 디자인 단계의 고려 사항, 즉, 성능이나 보안, 분산 등을 고려하여 결정하고, 관리할 수 있게 된다.



위의 화면은 디자인 단계의 Class를 생성하는 화면의 모습이다. 화면 왼쪽에는 AutoShip, ShipSimulator, MyShip Class의 프로세스들이 보이고, 화면 오른편에는 디자인 단계에 생성하고자 하는 Class 의 리스트이다. ShipSimulator의 경우, 명세 단계의 Class와 디자인 단계의 Class를 1:1 Mapping하였다. 화면에 보이는 다이어그램은 ShipSimulator에 할당된 프로세스의 이름들을 보여준다.

이렇게 프로세스를 할당하여 Class를 만들고 나면, 각 Class가 가지는 프로세스가 Method가 되어 이의 속성을 설정하게 된다. 2차년도의 과제에서는 이 속성을

SpM과 SvM 둘 중 하나로만 할당하도록 하였으나, 실제로 Real-time requirement가 Critical 하지 않은 경우에는 일반적인 함수 호출로 처리할 수도 있기 때문에, 속성을 SpM, SvM 외에도 None 을 추가하였다.



위에 보이는 화면은 AutoShip에 할당된 프로세스들의 속성을 설정하는 화면이다. AutoShip에서 초기값을 받아들이는 함수는 시뮬레이션이 시작하기 전에 수행되는 메쏘드이므로 Real-time requirement가 Critical 하지 않아 None 으로 설정하였고, 위치를 계산하는 GoStep은 100msec의 주기를 가지는 SpM으로, 충돌시에 속도 파라미터를 재설정하는 ResolveConflict 는 100msec의 deadline을 가지는 SvM으로, 속도 파라미터를 random 하게 바꾸는 GenerateSpeed는 2000msec의 주기를 가지는 SpM으로 각각 설정하였다.

(ㄴ) 디자인 단계의 Class로부터 디자인 객체 생성

만들어진 디자인 단계의 Class로부터 다음과 같은 과정을 거쳐 디자인 객체를 생성한다.

A. 명세 단계의 Class가 분할이 된 경우, 분할된대로 타입을 다시 결정하여 객체를 생성한다.

B. Composition 관계인 경우, 그 크기가 Static하게 결정되어 Array로 만들어진다.

C. Aggregation 관계인 경우, 동적으로 생성될 수 있으므로 Pointer로 만들어진다.

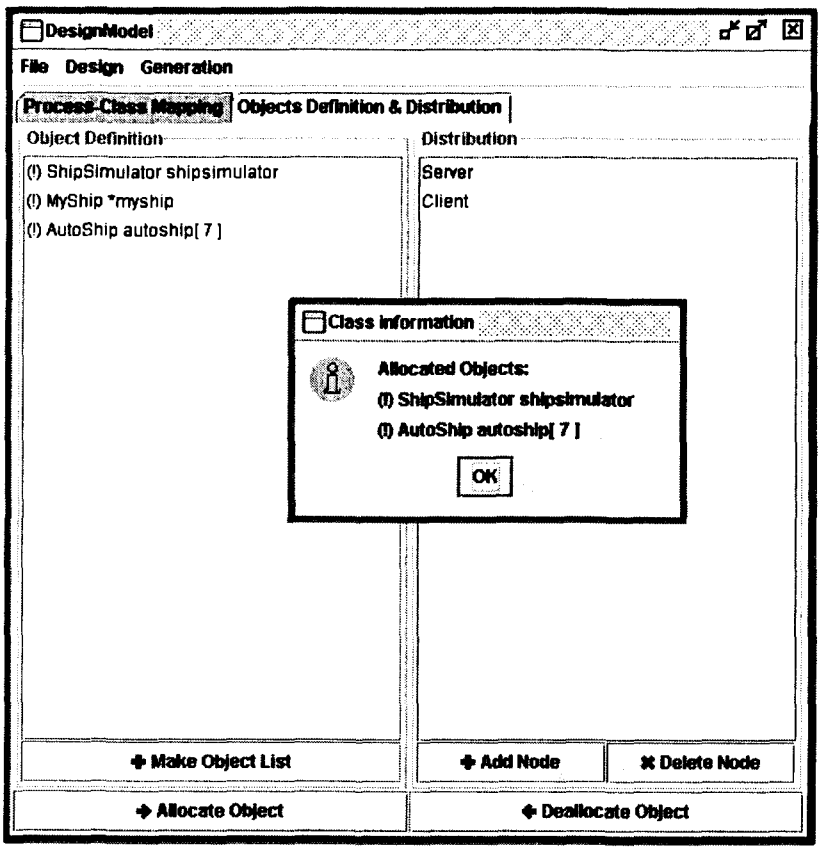
위와 같은 과정을 통해서 ShipSimulator에서는 다음과 같은 객체가 만들어진다.

- ShipSimulator shipsimulator
- MyShip *myship
- AutoShip autoship[7]

MyShip의 경우, Aggregation 관계이므로 pointer로 처리되었고, AutoShip의 경우 7의 Cardinality를 가지는 Composition 관계이므로 size가 7인 Array로 생성되었다.

(ㄷ) 분산 노드에 배치

디자인의 마지막 과정은 위 과정을 통해서 추출된 객체를 분산 노드로 배치하는 일이다. 다음 화면은 객체를 분산 노드로 할당하는 모습을 보여준다.



ShipSimulator 개발에는 2개의 분산 노드로 나누어진다. Server 노드의 경우에는 shipsimulator와 autoship이 할당되었고, Client 노드는 myship이 할당된다.

한 가지 주지할 사항은, myship의 경우 Dynamic하게 생성이 되므로, 반드시 Client라는 하나의 노드로만 사용되어야 하는 것은 아니라는 것이다. 코드는 참조하여 Instantiation할 수 있는 형태로 생성되므로, Client형태의 노드는 중복되어 만들어질 수 있다.

(ㄹ) 코드 생성

코드 생성의 기본적인 형태는 2차년도 과제와 유사하다. 그러나, 분산 노드와 관련한 사항들이 추가로 고려되었다. 아래에서는 2차년도 과제와 비교하여 달라진

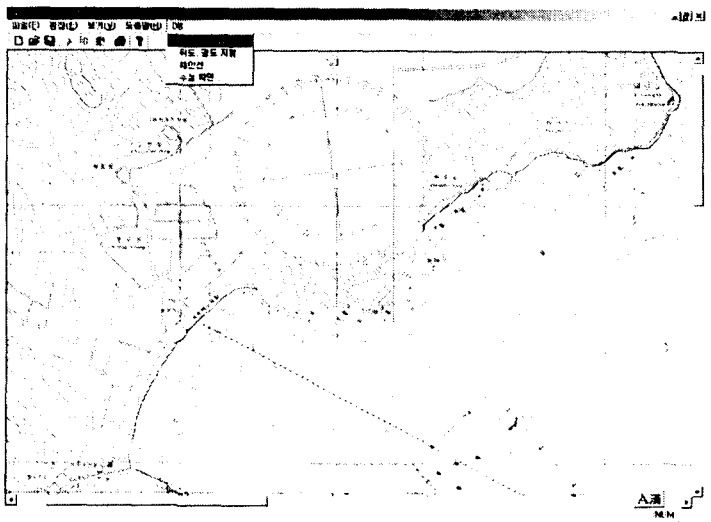
점을 중심으로 코드 생성 방법을 기술하였다.

- Class 단위로 Statechart와 Process를 코드로 생성한다.
- 노드 단위로 TMO를 초기화시키는 참조 함수를 생성한다.
- 만일 두 Class가 서로 다른 노드에 할당된 경우에는 두 Class간의 데이터 통신을 TMO를 이용한 채널 방식으로 처리한다.
- 만일 두 Class가 같은 노드에 할당된 경우에는 해당 class에 대한 refrence를 가지고 직접 데이터를 Access하도록 한다.

실행 결과

(ㄱ)EXDB 저작도구

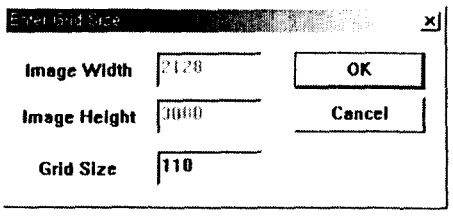
Windows환경에서 동작하며, 지도는 BMP나 JPEG 형식으로 된 것을 입력 받는다.



프로그램에서 지도를 loading한 뒤 DB관련 메뉴가 보여지는 화면

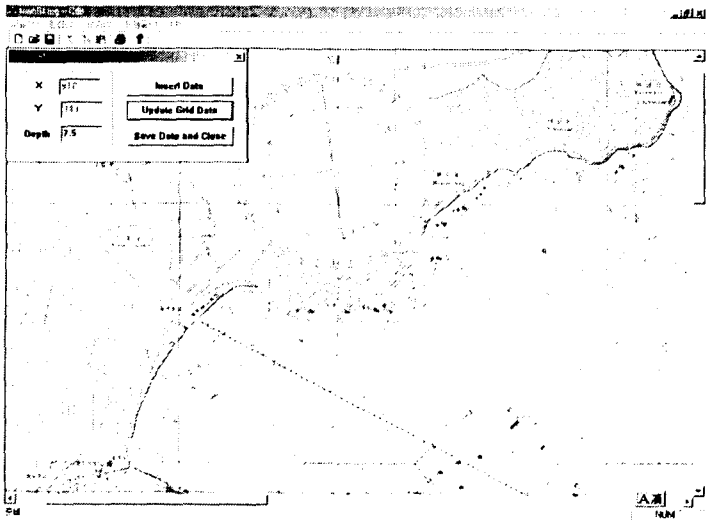
- *수심 자료 입력*

바다의 수심자료를 입력하기 위한 메뉴를 선택하면 로딩된 지도의 크기가 나타나고 이에 따라 어느 정도 크기의 Grid로 나눌 것인지 묻는 다이얼로그 박스가 뜬다.



수심자료 입력 선택시 보여지는 grid size입력 창

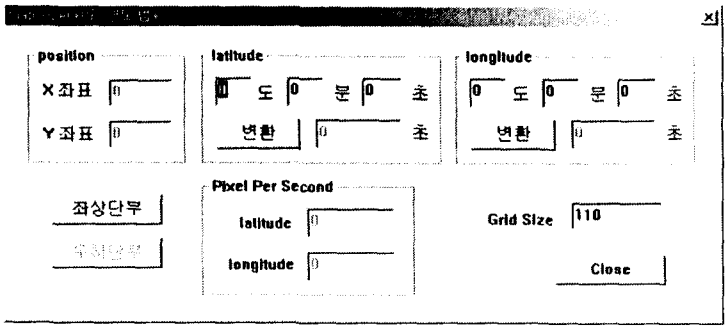
그리고 원하는 지점을 마우스로 클릭한 뒤 해당 포인트의 수심을 키보드 입력을 통해 지정 후 insert 버튼을 클릭하여 저장한다. 이 데이터를 그리드 교차점의 값으로 변환하기 위해서는 Update grid data라는 버튼을 클릭한다.



수심 입력화면

- *위도 경도 지정*

지도의 그림과 실제 위도, 경도 위치를 mapping시키기 위하여 지도에서 쉽게 위도, 경도의 값을 정확하게 알 수 있는 두개의 지점을 클릭하여 값을 입력한다. 이 때 두 지점의 위도, 경도값은 달라야 한다.

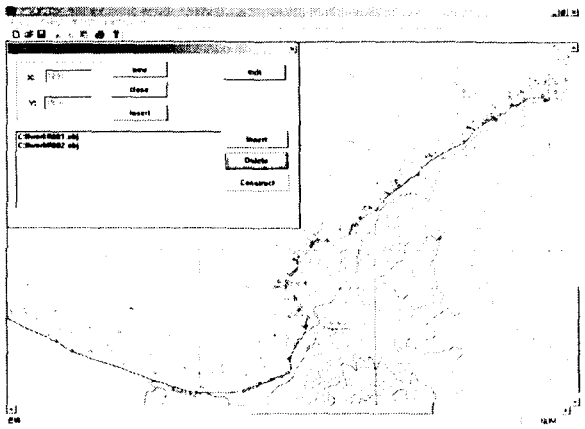


screen corrdinate와 지도의 위도, 경도를 mapping하기 위한 창

- *해안선*

new버튼을 클릭하여 새로운 해안선을 만든다.

굴곡이 있는 해안선의 지점들을 순서대로 클릭하고 insert버튼을 누른 뒤 close를 하면 하나의 해안선 line에 대한 자료를 저장할 수 있다.

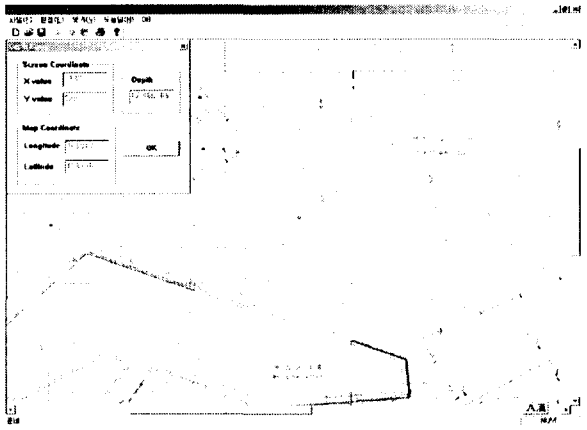


해안선 자료 두개를 선택하여 에디트창에 보여진 화면

필요한 해안선들의 자료를 모두 저장한 뒤 텍스트창 오른쪽에 있는 insert버튼을 눌러 원하는 해안선 자료가 있는 파일을 첨부 시킨 뒤 construct버튼 클릭으로 해안선의 최종 자료를 만든다.

- *수심 확인*

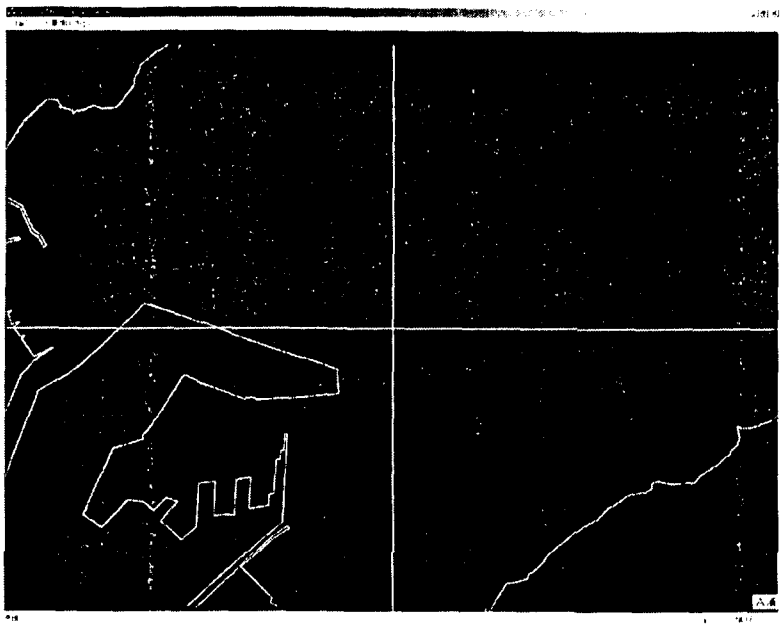
만들어진 수심데이터를 이용하여 원하는 지점에서 마우스를 클릭하면 깊이값을 보여준다.



수심확인

(ㄴ)Instructor System

Instructor System은 training file을 load한다. 이 training file에는 훈련을 하게 될 EXDB(해역), 자동으로 움직이는 배들의 초기위치, 방향 및 개수, 피 훈련자들이 초기 위치 및 방향 등에 관한 정보를 갖고 있다.

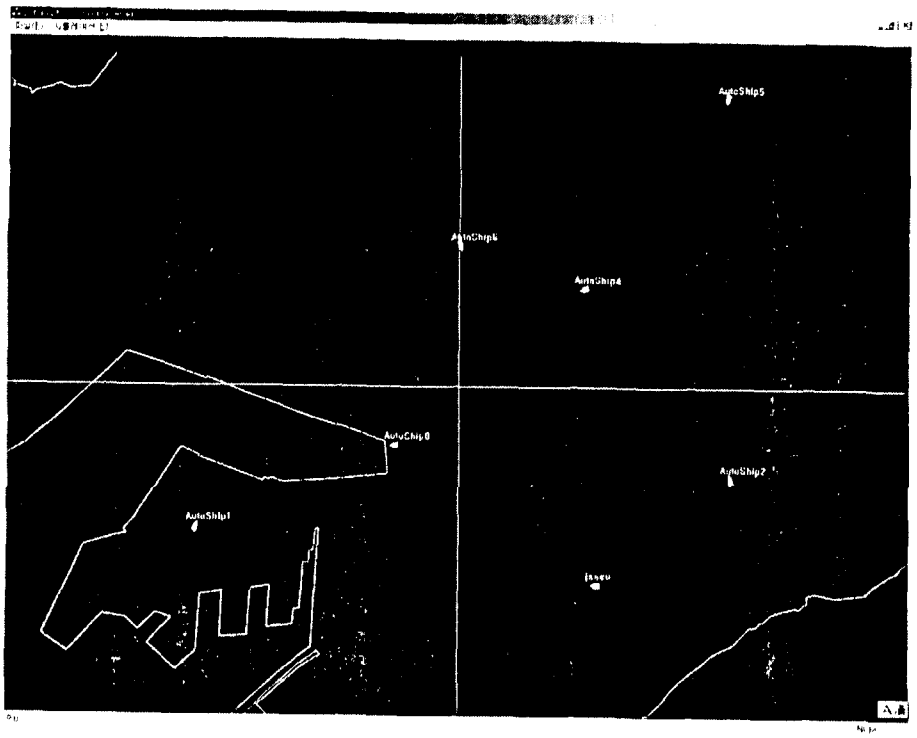


포항 해역 EXDB를 load한 화면

Trainee가 Instructor system에 접속하면 훈련 준비 상황이 되는데, 이 상태가 되면 instructor system에는 훈련에 참가하는 사람의 배와 자동으로 해역을 항해하는 여러 배들이 표현된다.

자동으로 항해하는 배들은 배들끼리의 충돌 혹은 해안선과의 충돌 시 충돌 회피 알고리즘에 따라 다른 해로를 선택하게 되고, 배들끼리의 충돌 감지는 배 사이의 거리로 감지하며 배와 해안선과의 충돌 감지는 해안선 및 수심으로 판단한다.

Instructor는 이 system을 사용하여 훈련 상황을 감독하여 필요한 경우 특정 부분을 확대/축소하여 볼 수 있으며, 인근 해역으로의 panning도 가능하다.



훈련중인 화면

(ㄷ)Trainee System

Trainee system은 초기에 Instructor System과의 연결을 시도하여 연결이 성공하면 현재 훈련 준비중인 해역에 관한 EXDB를 load한다. 그리고, Trainee system로부터 radar를 위한 data를 받고, 자신의 위치와 방향을 Trainee system에 전해준다.

Trainee는 2차원 radar display및 3차원으로 가시화된 환경을 바탕으로 항해를 하는데, steering stand로 방향을 조절하고 engine telegraph로 속도를 조절한다.

Trainee system의 control panel에는 radar display 뿐만 아니라, 현재의 속도, 각속도 및 수심 등의 정보가 표시되어 항해에 도움을 준다. 만약 다른 배들이나 해안선과의 충돌이 발생 할 때에는 선박이 멈추게 되어 훈련에 실패하였음을 알려준다.

Trainee system의 훈련 화면

자동으로 항해하는 선박들

나. 병렬성을 고려한 디자인 모델

병렬성을 고려한 디자인 모델, 즉 태스크 기반의 디자인 실시간 시스템에서 가장 많이 사용되고 있는 디자인 모델이다. [Gomaa93]태스크는 시스템의 병렬성을 사용자가 직접 제어하여 실시간 시스템의 가장 중요한 요구 사항 중 하나인 예측 가능성을 확보하는데 기본이 된다. 본 연구에서 목표로 하는 바는 실행 가능한 ASADAL 요구 명세로부터 디자인 단계의 고려 사항을 사용자가 결정할 수 있도록 지원하는 도구를 만들어 병렬성을 고려한 실행 가능한 디자인 모델을 구축하는 것이다. 이렇게 구축된 실행 가능한 디자인 모델로부터 실시간 OS 위에서 작동하는 코드를 자동 생성할 수 있다.

(ㄱ) 디자인 과정

실행 가능한 디자인 모델을 만드는 전체 과정을 살펴 보면 다음과 같다.

<table>
<tr><td rowspan="2">태스크
모델
생성 과정</td><td>태스크 생성</td><td>I/O 관계 설정</td><td>태스크 간 연결</td><td>수행 순서
결정</td></tr>
<tr><td>프로세스 할당
Statecharts 할당</td><td>I/O와 관련된
컴포넌트 연결</td><td>태스크간 연결이
필요한 데이터
추출 및
연결 관계 결정</td><td>태스크에 할당된
프로세스,
Statecharts,
IO 컴포넌트의
수행순서 결정</td></tr>
<tr><td rowspan="2">실행 정책
결정 과정</td><td>실행 정책 결정</td><td>실행 정책에 따른
파라미터 설정</td><td rowspan="2"></td><td rowspan="2"></td></tr>
<tr><td>태스크 스케쥴링
방법 결정

자원 공유에 따른
접근 방법 결정</td><td>태스크 주기
태스크 우선 순위
태스크 데드라인</td></tr>
</table>

다음의 각 항에서 각 과정을 자세히 살펴보도록 한다.

✓ 태스크 생성 단계

태스크 생성 단계에서 수행해야 할 작업은 다음의 세 가지 작업으로 나눌 수 있다.

1. 말단 프로세스를 태스크로 분할
2. Statechart의 각 분할영역에 대한 후보 태스크 결정
3. Statechart의 각 분할영역에 태스크 할당

위의 세 과정에서 1번과 3번 과정은 사용자가 직접 해야 하는 과정이고, 2번 과정은 자동적으로 수행되는 과정이다.

먼저 해야 하는 과정은 DFD의 말단 프로세스를 병렬성을 가지는 단위는 태스크로 분할하는 과정이다. DFD의 모든 말단 프로세스는 반드시 하나의 태스크에 소속되어야 한다. 이들을 분할할 때는 DARTS 방법론[Gomma84]에 자세히 소개되어 있듯이, 기능적인 집중성(Cohesion), 시간적인 집중성, I/O 의존도, 주기성 등을 고려하도록 한다.

기능 모델인 DFD를 태스크로 분할하고 난 후 해야 할 일은 행위 모델인 Statechart를 태스크에 할당하는 일이다. Statechart를 태스크에 할당하기 위해서는, ASADAL 명세 결과에서 Statechart의 논리적인 계층성을 이용하여 모델의 모든 DFD의 컨트릴러(Controller)를 하나의 Statechart로 연결해야 한다. 이렇게, Statechart의 논리적 계층성을 이용하여 모든 컨트릴러를 하나의 Statechart로 결합하고 나면, 이 Statechart를 태스크별로 분할하는 작업을 수행해야 한다. 이를 위해서 Statechart의 분할점과 분할영역에 대한 정의가 필요하다.

정의1 : 분할점

어떤 상태 S의 타입이 AND이거나, 부모 상태가 없는 루트 상태이면 그 S를 분할점이라고 한다.

정의2 : 분할영역

분할 영역은 상태의 집합으로, 분할점 S로부터 그 자식 상태들 중 분할점이 아닌 상태들을 순환적으로 계속 포함시킨 것이다.

Statechart의 태스크 분할에서 가장 기본적인 원칙은, 태스크에 할당되는 단위가 주어진 Statechart의 분할영역이라는 것이다. 이러한 원칙을 정한 이유는 첫째로, 태스크는 서로 병렬적으로 수행되게 되므로, 병렬성의 단위인 AND상태에서 분할되는 것이 바람직하기 때문이고 두 번째로, ASADAL의 Statechart 문맥에 의해서 분할영역 사이에서는 상태 전이가 일어나지 않기 때문에, 태스크간의 의존도가 낮아질 수 있기 때문이다.

Statechart의 분할은 궁극적으로는 사용자가 하게 되지만, 어떤 태스크에 할당하는 것이 바람직한 지 후보 태스크 추출을 통해 보여줄 수 있다. 후보 태스크를 추출하기 위한 기준은 다음과 같다.

- 어떤 상태 S가 어떤 말단 프로세스 P의 활성화를 담당한다고 할 때, S가 포함된 분할 영역은 P가 소속된 태스크에 소속되는 것이 좋다.
- 어떤 분할 영역 A1의 분할점이 어떤 분할 영역 A2의 분할점의 부모 상

태라면, A1은 A2와 같은 태스크에 소속되는 것이 좋다.

이 두 가지 원칙은 결국 태스크간의 의존성을 낮추기 위한 것이다. 첫 번째 원칙은 프로세스를 활성화시킬 것인가의 정보가 다른 태스크로부터 오는 것을 견제하는 것이고, 두 번째 원칙은 분할 영역의 활성화 여부가 다른 태스크에서 결정되는 견제하는 것이다. 이러한 원칙을 기반으로 하여, 분할 영역에 대한 후보 태스크를 추출하는 알고리듬을 만들 수 있다. 이렇게 후보 태스크 추출이 끝나면, 추출된 결과를 근거로 사용자가 직접 각 분할 영역을 태스크에 할당해야 한다. 모든 분할 영역에 대하여 태스크 할당이 끝나면, 각 태스크에 할당된 분할 영역들을 태스크별로 하나의 Statechart로 연결하는 작업을 하게 된다.

✓ **I/O** 설정 단계

디자인 단계에서 수행해야 할 두 번째 일은 입출력과 관련된 작업을 하는 것이다. 입출력을 보강하기 위해서 DFD를 직접 고치는 방법이 있을 수 있다. 즉, 어떤 프로세스가 A라는 입력을 처리한다면, A라는 입력을 받아들이는 프로세스를 추가하는 것이다. 그러나, 이러한 방식은 요구 분석 모델의 DFD와 디자인 단계에서의 DFD를 따로 관리해야 하는 번거로움이 있다. 따라서, 본 방법론에서는 DFD에 수정을 가하지 않고, 입출력과 관련한 컴포넌트를 설정하여 이들을 태스크에 할당하도록 하였다. 만일, 시스템이 입출력과 관련하여 변화를 일으킨다면, 할당한 입출력 컴포넌트의 내용을 수정하면 되므로, 높은 유지 보수성을 가질 수 있다.

입력 컴포넌트는 다음의 두 가지 속성으로 나누어볼 수 있다.

1. 주기적으로 폴링(Polling)을 하여 값을 얻는 컴포넌트
2. 인터럽트(Interrupt)를 받아들이는 컴포넌트. 즉, 인터럽트 핸들러 (Interrupt

Handler)

주기적으로 폴링을 하는 컴포넌트는 이미 만들어져 있는 태스크에 할당을 하거나, 새로운 태스크를 만들어 그 곳에 할당을 하여야 하고, 인터럽트를 받아들이는 컴포넌트는, 인터럽트의 등록 정보를 지정해야 하고, 필요한 경우, 그 컴포넌트가 수행시킬 태스크를 지정한다출력을 담당하는 컴포넌트는 인터럽트에 의한 것은 없고 모두 주기적으로 폴링을 하는 형태이다. [1]

주지할 사실은, 주기성을 가지는 입출력 컴포넌트가 반드시 하나의 태스크에만 할당되어야 하는 것은 아니라는 것이다. 주기성을 가지는 태스크에서 폴링을 하면서, 예외 상황이 발생한 경우 비주기적인 태스크에서 값을 받아들이는 경우도 있기 때문이다.

태스크에 할당된 모든 입출력 컴포넌트는 수행 조건을 가질 수 있다. 이 수행 조건은 상태의 활성화로 표현한다. 만일, 어떤 입력 컴포넌트는 "ON"이라는 상태가 활성화되어 있을 때만 값을 받아들인다면, 수행 조건을 "ON"이라고 하면 된다. 만일 아무런 수행 조건을 가지지 않는다면 해당 태스크가 활성화될 때 항상 수행을 하게 된다.

✓ 태스크 간 연결 단계

DFD와 Statechart를 태스크로 분할하고, 입출력 컴포넌트를 할당하고 나면, 태스크의 구성요소가 모두 완성이 된다. 이제 다음으로 해야 할 일은 이 태스크 간의

[1] 보통 실시간 OS 에서는 인터럽트 핸들러가 무한정으로 대기(block)하는 일이 없도록 하기 위하여 세마포어(Semaphore)와 같은 대기 함수를 사용하지 못하도록 되어 있다. 이러한 제약 사항을 극복하기 위하여 보통 인터럽트 핸들러는 우선순위가 높은 비주기적 태스크(Aperiodic task)를 기동 시키는 일을 담당한다. 이러한 구조인 경우, 인터럽트 핸들러와 태스크를 연결시키게 된다.

연결 관계를 확립하는 일이다.

태스크간의 연결 타입이 결정되어야 하는 데이터는 다음과 같다.

- 요구 분석 단계의 DFD에서 데이터 흐름이 연결되어 있던 프로세스가 서로 다른 태스크에 할당된 경우 이 데이터 흐름을 통해 전달되는 데이터
- 입출력 컴포넌트로부터 받아들인 데이터

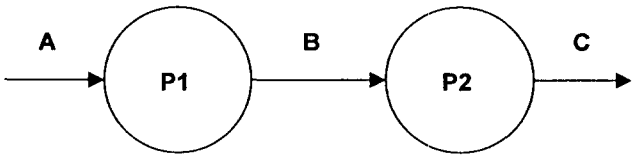
태스크간 연결 타입은 다음의 두 가지로 정해 진다.

- 크기: 연결 채널의 크기. 1보다 같거나 큰 자연수 값을 가진다.
- 알고리듬: 연결 채널에서 데이터를 빼내는 알고리듬. 예를 들어 FIFO(First In First Out), LIFO(Last In First Out) 등이 있을 수 있다.

크기와 알고리듬을 통하면, 다양한 형태의 연결 타입을 표현할 수 있다. 예를 들어서, 크기를 1로 알고리듬을 FIFO로 정한다면, 가장 먼저 도착한 데이터를 채널에 넣어두고, 이 후 들어오는 데이터는 무시하게 된다. 즉, 소비자 입장에서는 데이터를 소비하고 난 후, 가장 처음에 생산된 데이터를 다음에 소비하게 된다. 만일 알고리듬이 FIFO가 아니라 LIFO라면, 생산자는 채널에 계속 데이터를 덮어 쓰게 되고, 소비자는 가장 나중에 생산된 데이터를 소비하게 된다.

✓ 태스크 수행 순서 결정 단계

태스크 모델을 만드는 마지막 단계는 태스크 안에서의 수행 순서를 결정하는 일이다.



수행 순서 결정이 필요한 경우

위의 그림에서 수행 순서 결정이 필요한 경우의 예를 들었다. 만일 프로세스 P1, P2가 위와 같이 연결되어 있고, 이 두 프로세스가 같은 상태에서 활성화되도록 되어 있다면 명세 모델 상에서는 아무런 하자가 없다. P1이 반드시 P2보다 먼저 수행되어야 하는 표현은 없지만, 프로세스 P2는 P1이 B를 생산할 때까지 기다린 후, C를 생산할 것이기 때문이다. 그러나 순차적인 태스크 내부에서는 그러하지 못하다. 반드시 P1을 P2보다 먼저 수행해야만 같은 주기 안에서 C를 생산해낼 수가 있다. 즉, 한 태스크 안에서는 병렬성이 존재하지 않으므로, 병렬성을 가정했던 프로세스의 활성화에 순서성을 부여해야 한다는 것이다.

이 과정에서 프로세스만이 아니라, 할당된 입출력 컴포넌트와 Statechart도 함께 순서 결정을 하게 된다. 일반적인 구조는 입력 컴포넌트, Statechart, 프로세스, 출력 컴포넌트 순이겠지만, 필요한 경우 이들의 순서를 바꿀 수가 있다.

✓ 실행 정책 결정 및 인수 설정 단계

태스크 모델이 모두 만들어지면, 이제 이 태스크들을 어떻게 수행시킬 것인가 하는 실행 정책을 결정하고, 설정한 실행 정책에 필요한 인수(Parameter)를 설정해야 한다. 이러한 실행 정책은 인수 결정은 [Mark93] 을 참고했다.

이 단계에서 결정해야 할 실행 정책과 인수는 다음과 같은 것이 있다.

- 실행 환경

다중 프로세서(Multiprocessor) / 단일 프로세서(Uniprocessor)

- 스케줄링 방법

  고정된 우선 순위 스케줄링(Fixed-priority scheduling) 방법 / 고정되지 않은 우선 순위 스케줄링(Dynamic scheduling) 방법

- 데이터 공유 정책

  FIFO / 우선 순위 상속(Priority inheritance) / 우선 순위 내장(Priority ceiling)

- 태스크 속성

  시간적 속성: 주기적 / 비주기적

  인수: 주기, 우선 순위, 활성화 조건

  위험도: Hard / Soft / Firm

이렇게 태스크 실행 정책을 세우게 되면, 전체의 디자인 모델이 실행 가능해지게 된다.

✓ 태스크 실행 문맥

여기서는 태스크의 실행 문맥에 대해서 알아보도록 한다. Statechart는 기본적으로, 끝이 없이 수행되는 모델이다. 반면 태스크는 정해진 시간 안에서 수행이 완료되어야 한다. [Pnueli91] 따라서, 태스크에 할당된 Statechart가 한 주기 안에서 어디까지 상태 전이를 해야 하는가에 대한 문맥이 필요하다. 아래에서 이 주기적인 태스크에 할당된 Statechart의 실행 문맥을 설명하였다.

1. 태스크의 Statechart로 전달된 모든 이벤트를 이벤트 집합 I에 넣는다.
2. I에 있는 모든 이벤트가 동시에 발생한 것으로 간주하고, Super step을 실행

한다.(이 때 태스크 외부로부터 들어오는 이벤트는 반영하지 않는다.)

여기서 주지할 사실은, 첫째, 태스크의 Statechart가 상태 전이를 하기 전에 외부로부터 전달된 모든 이벤트들을 한꺼번에 발생한 것으로 처리한다는 것이고, 둘째, 태스크의 Statechart가 Super step을 하는 동안 발생하는 외부로부터의 이벤트는 반영하지 않고, 그 다음 주기에 처리한다는 것이다. 이벤트를 한꺼번에 발생한 것으로 처리하는 이유는, 분산되어 배치되어 있는 이벤트들의 순서를 맞추려면, 이벤트들이 생성된 의존 관계를 모두 파악해야 하는데, 이 경우 부하가 너무 크기 때문이다. 그리고, Super step 동안 외부 이벤트를 반영하지 않는 이유는, Statechart가 외부 이벤트에 의해서 연속적인 상태 전이를 하는 것을 막고 모델을 단순화하기 위함이다.

이러한 태스크 실행 문맥은 명세 모델과 다른 실행 결과를 낳을 수도 있지만 디자이너가 이러한 변화를 예측할 수 있으므로 효과적인 디자인이 가능하다.

✓ 자동 코드 생성을 위한 라이브러리

위의 과정을 통해서 만들어진 실행 가능한 디자인 모델에, 사용할 플랫폼의 정보를 더하면, 실제로 실시간 오에스(OS) 위에서 태스크 기반으로 작동 가능한 코드를 만들 수 있다. 그러기 위해서는 플랫폼의 정보를 적절히 Hiding할 수 있는 라이브러리가 필요하다. .여기서는 이 라이브러리에 관해 설명하도록 한다.

라이브러리는 Statechart 구동기, 태스크 스케줄러, 타이머, 초기화 모듈로 나누어볼 수 있다. 각각에 대해서 살펴보도록 하자.

- Statechart 구동기

Statechart 구동기에는 Statechart, 상태, 상태 전이, 연결자에 대한 구조체 정의와, Statechart가 Super step을 실시하는데 필요한 함수들이 포함되어 있다. 특징은, 함수 포인터를 최대한 활용하고, 수행 중간에 동적 메모리 할당[4]이 일어나지 않도록 하여 수행 시간을 절약하였다는 것이다. Statechart가 실행되는 순서는 다음과 같다.

1. 태스크 외부로부터 발생한 이벤트를 가져온다. (Statechart는 구조체에서 getEvt라는 함수 포인터를 가지게 되는데, 각 태스크의 Statechart는 외부로부터의 이벤트를 받아들이는 함수를 만들고 이를 getEvt의 포인터에 연결 하도록 코드가 생성된다.)
2. 현재 활성화 중인 상태를 출발 상태로 가지는 상태 전이를 검색하여, 주어진 이벤트에 의해서 전이가 일어날 수 있는 상태 전이를 찾아낸다.
3. 찾아낸 상태 전이를 출발 상태의 레벨이 낮은 순서로 정열한다. (루트에 가까울수록 레벨이 낮다) 즉, 상태 전이를 우선 순위가 높은 것 순위로 정열한다.
4. 정열된 상태 전이를 앞에서부터 차례대로 꺼내어, 상태 전이가 일어날 때, 빠져나가는 상태와, 들어오게 되는 상태를 찾아낸다. 만일 상태 전이의 출발 상태가 빠져나가는 상태에 포함되어 있다면, 이미 자신보다 우선 순위가 높은 상태 전이가 일어났다는 것이므로 무시한다.
5. 빠져나오게 된 상태에 대해서 상태 명세서의 OnExit 부분을 수행하고, 현재 활성화된 상태 집합에서 제한다.
6. 선택된 상태 전이에 대해서 행동을 수행한다.
7. 들어가게 된 상태에 대해서 상태 명세서의 OnEntry 부분을 수행하고, 현

[4] 실제로 테스트 결과, VxWorks 에서 malloc 함수는 0.1msec, free 함수는 0.15msc 정도의 시간이 소요되는 것으로 파악되었다.

재 활성화된 상태 집합에 추가한다.

8. 5-7 과정동안 발생한 이벤트가 없으면 종료한다.
9. 5-7 과정동안 발생한 이벤트에 대하여 태스크 밖으로 알려져야 하는 것을 sendEvt 함수 포인터를 이용하여 전송한다. (1번에서 설명한 getEvt와 같은 방식이다.)
10. 5-7 과정동안 발생한 이벤트로 2번 과정부터 다시 시작한다.

- 태스크 스케줄러

VxWorks는 태스크의 우선 순위는 처리를 하지만, 주기적/비주기적 태스크를 직접 구분하고, 태스크의 주기 간격을 맞춰주는 등의 작업은 하지 않는다. 따라서, VxWorks가 제공하는 세마포어를 사용하여 이러한 작업을 하는 스케줄러 라이브러리를 만들었다. 사용자는 아래와 같은 간단한 함수를 호출함으로써 태스크를 등록할 수 있다.

```
void Initialize_Task(int type, char *name, FUNCPTR entry, FUNCPTR run_entry,
                     int priority, int period)
```

여기서 type은 주기적/비주기적 태스크를 구분하는 구분자이며, name은 태스크의 이름, entry는 실제 태스크의 형태를 갖추고 있는 함수의 이름, run_entry는 태스크로 동작할 함수의 이름, priority는 태스크의 우선 순위, period는 태스크의 기동 주기를 각각 나타낸다.

- 타이머

타이머는 Statechart안에서 사용되는 타임 아웃 이벤트를 처리하기 위해 만들어졌다. 사용자가 타임아웃 이벤트에서 단위 시간으로 잡은 시간(tm(e, 1)이라고 했을 때 1이 의미하는 실제 시간)마다 한번씩 수행하면서 타임 아웃과 관련한 요

청을 처리한다. 생성된 코드는 이 타이머에 시간과 불리언 값의 포인터를 넘겨 등록을 한게 되고, 타이머는 요청된 시간을 체크하여, 그 시간이 경과하면, 생성된 코드가 넘겨준 포인터에 TRUE값을 넣도록 되어 있다.

- 초기화 모듈

  VxWorks 환경 하에서 수행되는 코드들이 초기화 시에 사용해야 할 공통적인 부분을 라이브러리화 하였다.

✓ 지원 도구

여기서는 이제까지 설명한 실행 가능한 디자인 모델을 만들고, 실제 VxWorks 실시간 OS환경 하에서 수행 가능한 C코드 생성하는 도구의 사용법을 설명하도록 한다. 위에서 디자인 모델 만드는 방법의 순서에 따라 차례대로 설명해나갈 것이다.

1. 프로젝트 로드(Load)

디자인 모델을 만들고자 하는 요구 분석 모델의 프로젝트를 ASADAL의 메인 메뉴에서 로드한다.

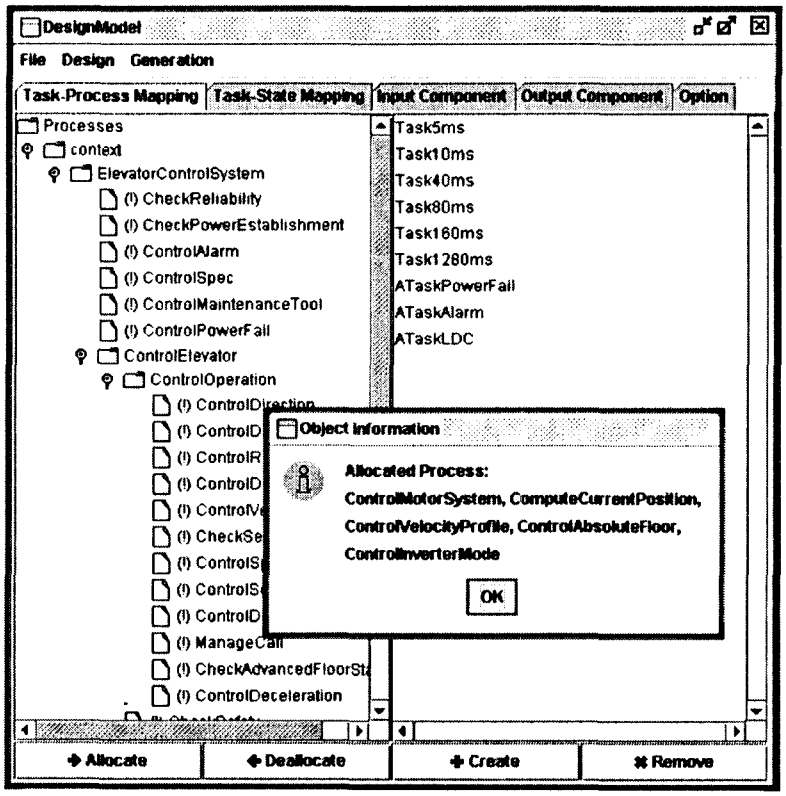
2. 디자인 도구 생성

ASADAL 메인 메뉴에서 Code Generation 항목의 "Generate C Code"를 선택한다. 디자인 도구는 File, Design, Generation 3개의 주요 메뉴 항목과, Task-Process Mapping, Task-State Mapping, Input Component, Output Component, Option의 네 가지 탭으로 구성되어 있다.

3. DFD 로드

생성된 도구의 Design 항목에서 "Load DFD"를 선택한다. 그러면, 1에서 로드한 프로젝트의 DFD 들을 선택할 수 있는 다이알로그가 생성된다. 여기서 Context Diagram에 해당하는 DFD를 선택하고 OK를 한다. 그러면, Task-Process Mapping 탭에서 좌측 편 Process에 Context Diagram을 시작으로 모든 프로세스가 Tree구조로 보이게 된다.

4. Task 생성 및 Leaf Process 할당

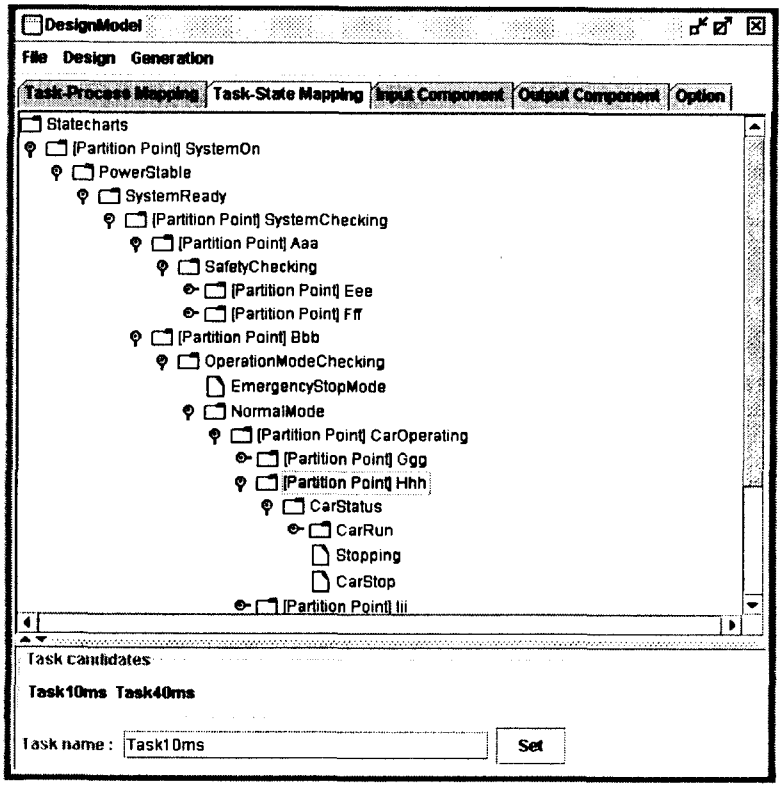


Leaf Process 할당

앞서 설명하였듯이 모든 말단 프로세스는 반드시 태스크에 할당되어야 한다.

새로운 태스크를 만들기 위해서는 그림의 우측 하단에 보이는 "Create" 버튼을 누르면 된다. 프로세스를 할당하기 위해서는 좌측의 프로세스와, 우측의 태스크가 모두 선택된 상태에서 좌측 하단의 "Allocate" 버튼을 누르면 된다. 만일 중간 프로세스(Intermediate Process)를 선택한 상태에서 "Allocate"버튼을 누르면, 그 프로세스의 모든 자식 프로세스가 해당 태스크에 할당되게 된다. 할당된 결과는 Design메뉴에서 "Show Task Information"을 선택하면 위의 그림과 같이 확인할 수 있다. 할당을 취소하기 위해서는 프로세스를 선택한 상태에서 "Deallocate" 버튼를 누르면 되고, 태스크를 없애기 위해서는 태스크를 선택한 상태에서 "Remove"버튼을 누르면 된다. 이미 선택이 된 말단 프로세스는 프로세스 트리에서 (!) 표시를 앞에 달게 된다.

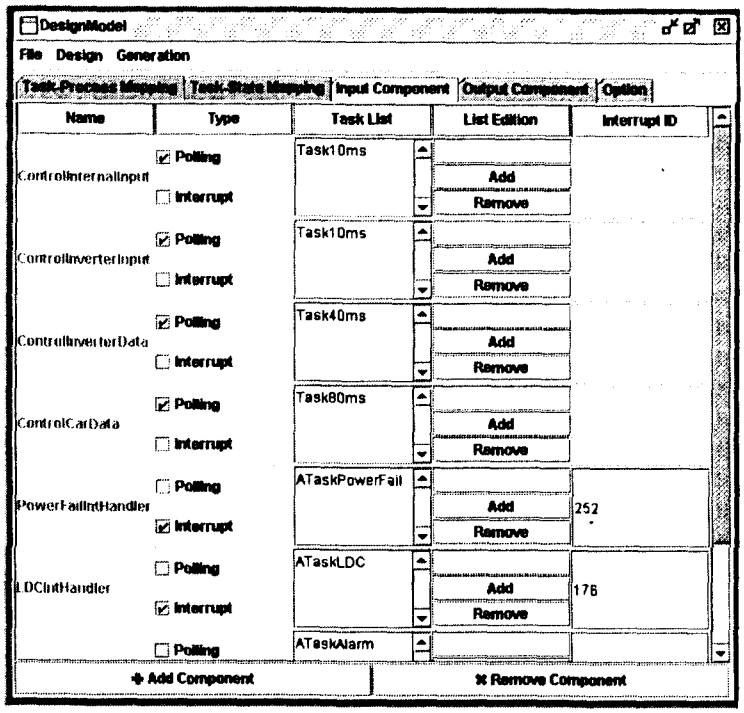
5. Statechart 할당



Statechart 할당

다음 단계는 태스크에 Statechart를 할당하는 것이다. 모든 말단 프로세스가 4번의 과정으로 태스크에 할당이 되면, Design메뉴 항목에서 "Distribute Statechart"를 선택한다. 그러면, 위 그림과 같이 DFD와 연결된 모든 Controller를 연결하여 하나의 루트 상태를 가지는 Statechart로 만들고, 각 분할점에 대한 후보 태스크를 추출하여 보여주게 된다. 이 결과는 Task-State Mapping 탭에서 볼 수 있다. 분할점은 이름 앞에 "[Partition Point]"라는 어두를 붙이게 되고, 분할점인 트리 노드를 선택하면 하단의 "Task candidates"에서 후보 태스크를 확인할 수 있다. 사용자가 해야 할 일은 모든 분할점에 대해서 태스크를 할당하는 것이다. 만일, 4번 과정에서 만들지 않은 태스크를 할당하고 싶으면, Task-Process Mapping 탭에서 새로운 태스크를 먼저 만들고 나서(프로세스는 없고, Statechart만 할당되는 태스크가 있을 수도 있다.) Task name 필드에 태스크 이름을 넣으면 된다. 반드시 "Set" 버튼을 눌러야만 값이 입력이 되고, 이미 입력된 값은 해당 분할점을 선택했을 때, Task name 필드에서 보여지게 된다.

6. 입출력 컴포넌트 설정

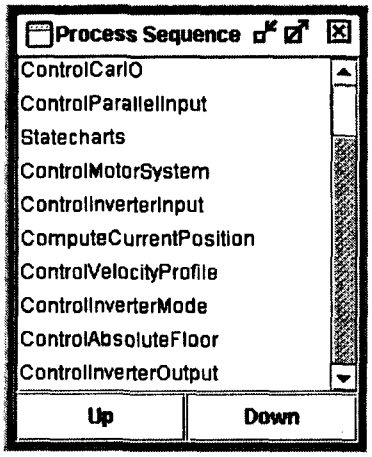


입출력 컴포넌트 설정

다음으로 해야 할 일은, 입출력 컴포넌트를 설정하는 일이다. 입출력 컴포넌트는 "Input Component", "Output Component" 탭에서 각각 수행할 수 있다.[5] 새로운 컴포넌트를 추가하기 위해서는 위 그림에서 좌측 하단에 있는 "Add Component" 버튼을 누르고 컴포넌트의 이름을 입력하면 되고, 삭제하기 위해서는 "Remove Component" 버튼을 누르고 역시 컴포넌트의 이름을 입력하면 된다. 컴포넌트가 주기적인 태스크에 할당되어, 값을 폴링할 것인지 아니면 인터럽트 핸들러인지를 "Type"에서 선택하고, 만일 폴링일 경우는 어떤 태스크에서 폴링을 할 것인지 Task Edition에서 입력한 후 "Add" 버튼을 눌러 Task List에 추가한다. 입출력 컴포넌트는 이러한 방식으로 다수의 태스크에 할당될 수 있다. 만일, 인터럽트 핸들러 일 때는 Task Edition에서 이 핸들러가 기동시킬 태스크를 입력하고, Interrupt ID에 어던 인터럽트의 핸들러인지 고유한 ID(시스템에서 정해져 있는 번호)를 등록한다.

[5] 만일 어떤 컴포넌트가 입출력을 같이 담당한다면, 입력 컴포넌트에 넣어두면 된다.

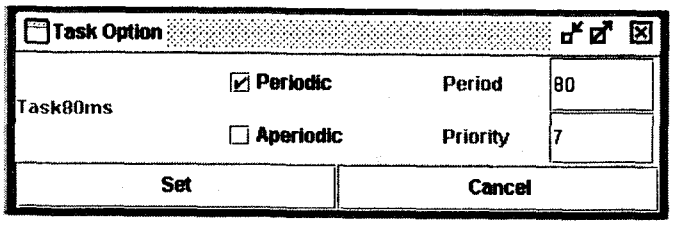
7. 태스크 안에서의 프로세스 순서 조정



프로세스 순서 조정

태스크에 프로세스, Statechart, 입출력 컴포넌트가 모두 할당되면, 이들의 순서를 조정 해야 한다. 이것은 Task-Process Mapping 탭에서 태스크를 선택한 후, Design 메뉴에서 "Set Process Sequence"를 선택하면 된다. 그러면 위 그림과 같은 창이 생성되는데, 순서를 바꾸고자 하는 프로세스를 선택하고, Up, Down 버튼을 사용하여 순서를 바꾸면 된다. Statecharts라고 써진 항목은, 태스크에 할당된 Statechart를 언제 수행시킬 것인가 하는 것이다. (기본적인 순서는 입력 컴포넌트, Statechart, DFD 프로세스, 출력 컴포넌트이다.)

8. 태스크 실행 파라미터 결정



태스크 실행 파라미터 결정

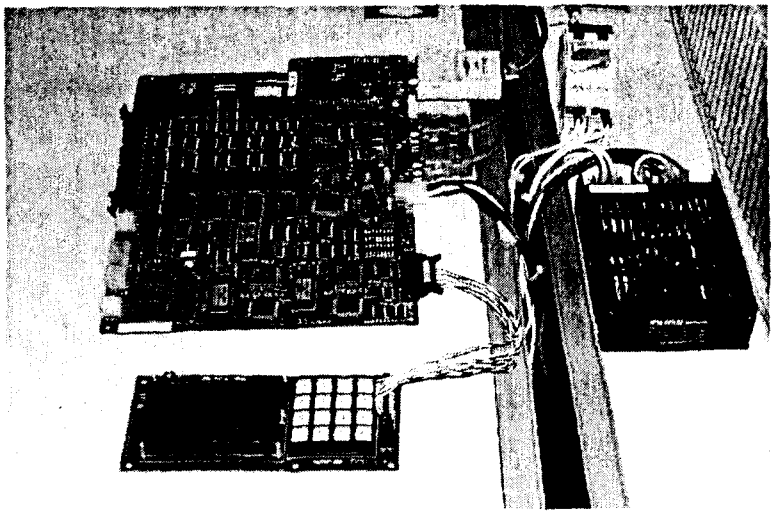
마지막 단계는 태스크의 실행 파라미터를 결정하는 것이다. (본 도구는 VxWorks에서 지원하는 "Fixed Priority Scheduling"방식을 전제하고 있다.) 파라미터는 위 그림에서 보이듯이 주기적인지, 비주기적인지를 선택하고, 주기적일 경우는 주기와(msec 단위) 우선 순위를 입력하며, 비주기적일 경우는 우선 순위만 입력한다.

9. 코드 생성

마지막 코드 생성은 Generation 메뉴에서 "Generate Code"를 선택하면 된다. 그러면 해당 프로젝트의 EX\GEN 디렉토리에 앞에서 설명한 코드들이 생성되게 된다.

(ㄱ) 실행 예시

본 방법과 도구를 통해서 실시간 OS인 VxWorks 환경에서 엘리베이터 시스템을 수행하여 보았다. 본 도구에서 입력 컴포넌트에 엘리베이터의 외부 하드웨어의 동작을 흉내내는 가상 소프트웨어 컴포넌트를 넣었고, 출력 컴포넌트에 결과를 보여줄 수 있는 간단한 GUI 프로그램의 인터페이스를 출력 컴포넌트에 넣었다. 아래에서 본 도구를 이용한 테스팅 환경을 화면으로 소개하였다.



자동생성된 코드가 Downloading되는 보드의 모습.

(이 보드에는 VxWorks 가 탑재되어 있다.)

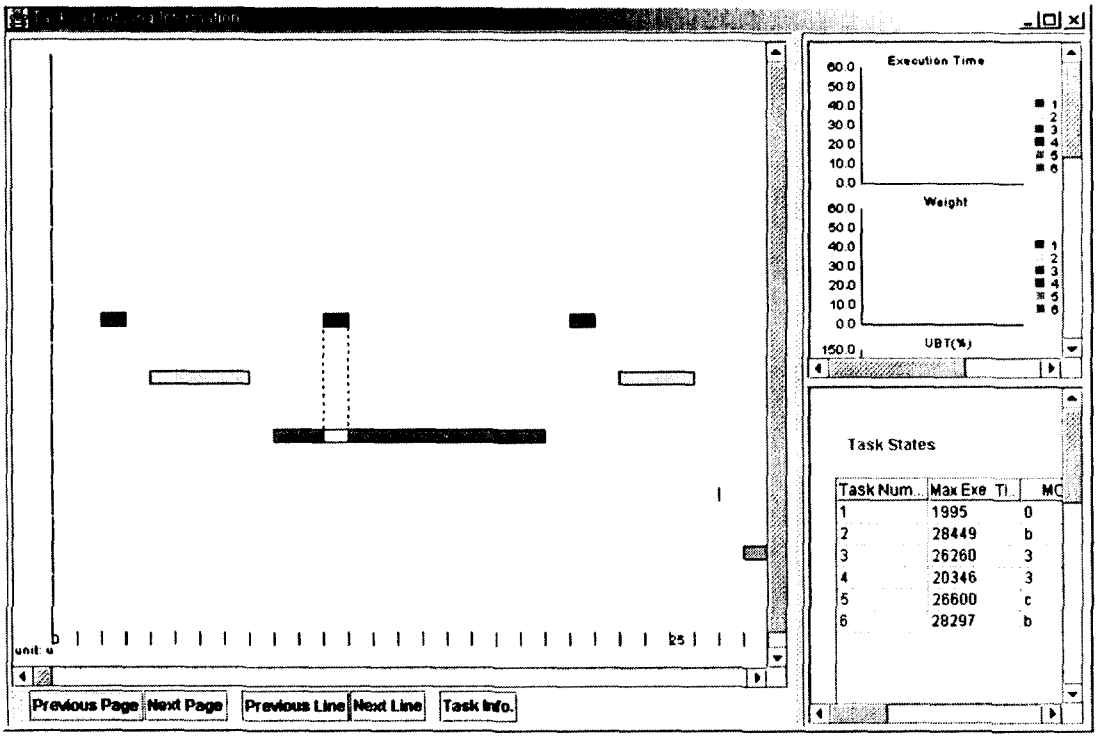
테스팅 환경

(우측의 Workstation으로부터 코드를 Downloading하고,

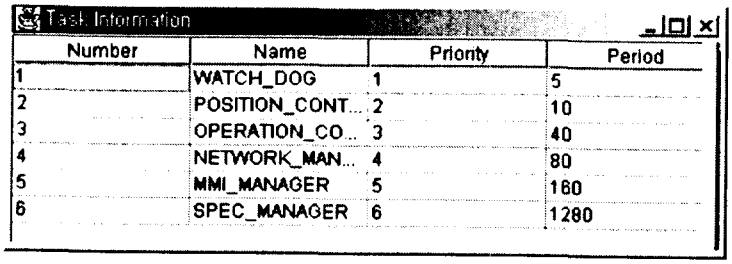
좌측의 PC는 보드와의 Serial 통신을 통해 결과를 확인한다.)

(ㄴ) 디자인 결과 분석 도구

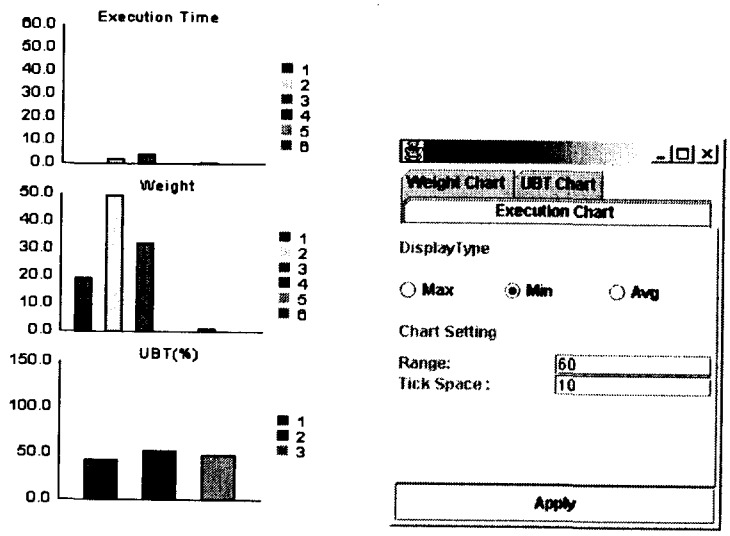
태스크 기반의 실시간 시스템 디자인에서 가장 중요한 것은 태스크의 우선 순위와 주기를 조절하여 Schedulability를 확보하는 것이다. [Stankovic90] 이러한 Schedulability를 확보하기 위해서는 태스크의 수행 시간을 측정하여 Utilization을 계산해야 한다. 이러한 기능을 수행해 주는 도구는 주로 OS를 만든 회사에서 개발하여 지원을 하지만, 이러한 도구는 가격이 매우 비싸므로, 본 과제에서는 간단한 시간 측정 방식을 사용하여 디자인 결과를 분석할 수 있는 도구를 개발하였다. 이 도구는 자동 생성된 코드가 디버그 옵션에 의해서 생성하는 Log 파일을 분석하여 Graphical 하게 보여준다.



위의 그림이 디자인 도구의 화면이다. 화면 왼편에 보이는 것은 태스크의 활동상황이며, 오른편 위쪽의 그래프는 태스크의 Execution Time, Weight, UBT(Utilization Bound Test)에 대한 결과이며, 오른편 아래쪽의 도표는 최대 수행시간이 걸렸을 때 사용자가 보여주기를 바라는 정보를 보여주는 도표이다.



위의 화면은 수행된 태스크에 대한 정보이다. 본 도구는 위에 보이는 6개의 태스크를 기준으로 한 엘리베이터 시스템에 대하여 수행 측정되었다.



위에 보이는 화면은 그래프의 모습과 그래프의 옵션을 조절하는 창을 보여주고 있다. 각 그래프는 Hyper-period(각 태스크의 주기의 최소 공배수)마다 값을 재계산하여 보여준다. 화면 왼편의 태스크 활동상황에서 Hyper-period마다 빨간 줄이 보이고 그 빨간 줄을 더블 클릭하면 그래프가 업데이트 된다.

각 그래프에 대해서 설명하면 다음과 같다.

- Execution Time

  수행 시간은 각 태스크 별로 Hyper-period 안의 Min, Max, Average 값을 기준으로 볼 수 있다.

▪ Weight

Weight는 각 태스크가 전체 수행 시간 상에서 어느 정도의 비중을 차지하는 가를 보여준다.

▪ UBT

UBT는 Utilization Bound Test 로서, 다음의 공식으로 얻어지는 기준값으로부터 Utilization이 어느 정도를 차지하는가를 보여주는 것이다. 만일 UBT가 100이상이면 Schedule을 할 수가 없다는 것이다.

$$\frac{C_1}{T_1} + \bullet\bullet\bullet + \frac{C_n}{T_n} \le U_{(n)} = N(2^{1/n} - 1)$$

$N = Task$ 수

$C_i = Task\ i$의 실행시간

$T_i = Task\ i$의 실행주기 [LiLa]

Task States

Task Num...	Max Exe. Ti...	MODE	RUNNING	DECELER...	PROFILE S...	DOOR CL...
1	1995	0	0	0	0	0
2	28449	b	0	0	0	0
3	26260	3	0	0	0	0
4	20346	3	0	0	0	0
5	26600	c	0	0	0	0
6	28297	b	0	0	0	0

위에 보이는 화면은 Task의 최대 수행 시간에 대한 정보를 보여주는 화면이다. 사용자는 최대 수행 시간이 걸렸을 때 보고 싶은 데이터 값을 수행 전에 입력할 수 있고, 여기서 그 값을 확인할 수 있다.

다. ASADAL 도구 보완

본 연구 과제를 통해서 개발한 방법론은 ASADAL 이라는 이름의 통합 도구로

서 개발 및 관리되어 왔다. 과제 일정의 마지막 해인 올해는 그 동안 개발된 도구의 문제점으로 지적되어온 DFD, Statechart 의 Drawing에 있어서 사용자 인터페이스의 불편함을 개선하였고, Simulation시의 각종 버그를 해결하였다. 여기서는 본 도구의 개선 사항 중 가장 특이할만한 사항인 프로토타이핑 도구에 대해서 자세히 알아보도록 하겠다.

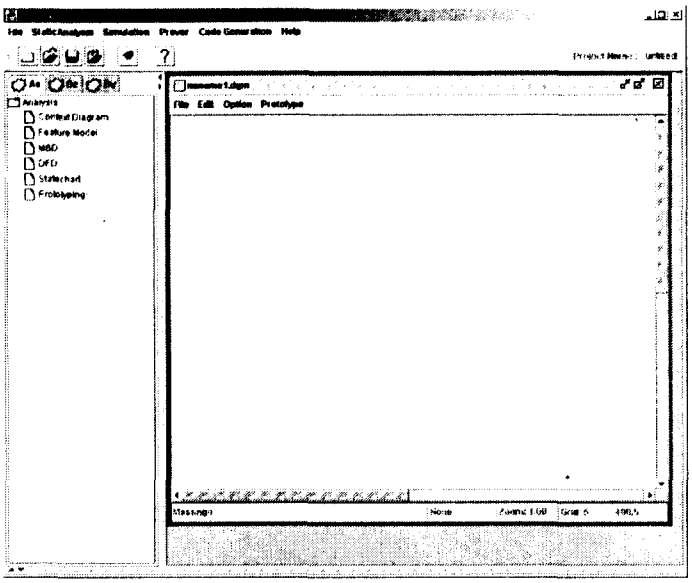
프로토타이핑 도구

(ㄱ) 개요

시스템의 수행 결과를 초기 단계에서 확인하기 위하여 빨리 시스템을 구현하여 그 결과를 먼저 확인해 보는 것을 프로토타이핑이라고 한다. ASADAL에서 제공하고 있는 요구 명세는 실행 가능한 성질을 띄고 있기 때문에, 명세 결과물을 통하여 이러한 프로토타이핑이 가능하다. 프로토타이핑은 기본적으로 시스템을 외부 사용자 관점에 놓고, 올바르게 작동하는 가를 살펴보는 것이므로, 효과적인 프로타이핑을 위해서는, 외부 개체에 대한 쉬운 모델링 방법이 제공되어야 한다. 특히, 내장형 시스템(Embedded system)의 경우, 외부 개체가 H/W 인 경우가 많고, 이들이 능동적으로 이벤트를 주고 받기 때문에, 외부 개체 자체의 행위를 표현할 수 있는 방법도 제공되어야 한다.

본 도구에서 구현한 Visual Prototyping은 외부 개체의 모양을 쉽게 모델링하고, 이들의 행위를 작성한 코드의 삽입을 지원하고, ASADAL 명세를 통해 자동 생성된 코드와 연동하여 수행하는 환경을 제공하고 있다

Drawing



Prototyping Drawing Window 의 모습

위에서 보이는 화면은 Desktop 형태의 Main Window 이고, 각각의 다이어 그램은 Desktop의 Internal window로 생성되어 작업할 수 있다.

본 도구에서는 Polygon과 같은 Drawing 객체들, Button과 같은 사용자 인터페이스 객체들, Bar chart와 같은 그래프를 제공하고 있다. 현재 제공되고 있는 컴포넌트는 아래와 같다.

- ✓ Drawing 컴포넌트

  Polygon, Arc, Curve, Line, Text

- ✓ User interface 컴포넌트

  Push Button, RadioBox, RadioButton, ListBox, TextField, Slider

- ✓ Graph

  Guage, Bar Chart, Pie Chart, Line Chart

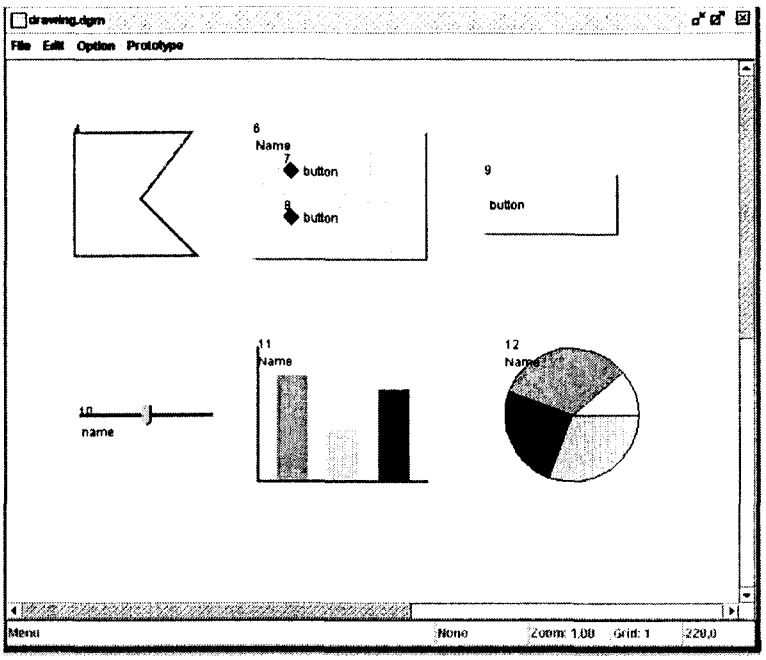
각 컴포넌트는 마우스 왼쪽 버튼을 누르면 그리기 시작하고 다시 왼쪽 버튼을 누르면 그림이 그려진다. (Polygon과 같이 여러 번의 클릭이 필요한 경우, double click

을 하면 그림이 그려진다.) 오른쪽 버튼을 누르면 그리기가 취소된다. 이 원칙은 Scale이나 Move에도 마찬가지이다.

각 컴포넌트에 대해서 다음과 같은 기능이 수행 가능하다.

- ✓ Scale : 컴포넌트를 선택하였을 때 보이는 파란 점을 이용해서 크기를 변경할 수 있다.
- ✓ Move : 컴포넌트의 선상에서 클릭을 하면 Move할 수 있다.
- ✓ Depth 조정: 컴포넌트의 보이는 순서를 조절할 수 있다. 즉, 각 컴포넌트에 대해 위로 올리기 뒤로 보내기 등이 가능하다.
- ✓ Parameter 설정 : 각 컴포넌트의 parameter를 설정할 수 있다. 예를 들어 선의 색깔, 도형을 채우는 색깔 등을 조절할 수 있다. 이러한 Parameter는 각 컴포넌트마다 조금씩 다른데, 이것은 3장에서 자세히 설명하도록 한다.

위와 같은 기능을 사용하여 Drawing 컴포넌트들을 원하는 형태로 그릴 수가 있게 된다. 그 모습의 예를 아래에서 나타내었다.



Drawing 한 결과의 예제

각 컴포넌트는 색깔을 바꾸거나 위치를 바꾸는 등의 API를 가지고 있어서 사용자가 Simulation 시의 컴포넌트의 변화를 설정할 수 있다. 이에 대한 자세한 소개는 다음의 코드 생성기와의 연결에 있다.

프로토타이핑의 수행 방법 설정

ASADAL 코드 생성기는 요구 명세의 내용을 코드로 자동 생성해 주는데, 이 코드는 사용자가 작성할 코드와의 효과적인 결합을 위해 API를 제공하고 있다. 이 API에는 이벤트나 데이터를 자동 생성된 코드로 보내고, 사용자가 관심 있는 이벤트나 데이터를 자동 생성된 코드로부터 받는 방법이 포함되어 있다. 따라서, 사용자는 원하는 프로토타이핑을 완성하기 위해서는 이러한 API를 사용하여 외부 개체를 위한 프로그램을 작성해야 한다.

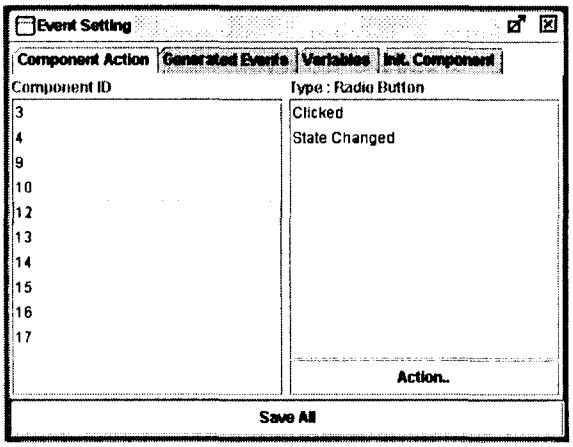
효과적인 프로토타이핑을 위해서 Visual prototyping 도구에서는 위에서 설명한

ASADAL로부터 자동 생성된 코드, 즉, 내부 시스템의 프로토타이핑 결과와 Prototyping도구로부터 그려진 결과물, 즉, 외부 시스템의 프로토타이핑 결과를 연결하는 중간 코드를 자동 생성하고, 이들을 동적으로 수행 가능하게 해 준다.

두개의 코드를 연결시켜주는 중간 연결 코드는 변수 선언부, 컴포넌트와 관련된 인스턴스를 얻어 오는 부분 그리고 이벤트 발생 및 수신부분 등으로 이루어져 있는데 이런 부분은 항상 반복적으로 나타나게 되는 패턴이므로 자동으로 코드를 생성하게 되면 프로그램 작성 시간도 줄어들게 될 뿐만 아니라 에러가 발생할 확률도 줄일 수가 있다. 여기서는 중간 연결 코드 작성을 위한 도구의 설명과, Prototype 컴포넌트가 제공하는 API에 대해 설명한다.

코드 생성기는 드로잉 윈도우의 Prototype-Event Setting메뉴를 선택하면 된다. 코드 생성기의 상단 부분은 컴포넌트와 관련된 액션을 설정할 수 있는 여러 개의 탭으로 분리된 화면이고 아래에 위치한 버튼들은 코드를 생성하고 , 코드생성과 관련된 옵션을 설정하는 기능을 제공한다. 다음에서 각각의 탭에 대해 설명하도록 한다.

◆ ***Component Action***



Component Action

왼쪽에 보이는 리스트는 이벤트를 발생할 수 있는 컴포넌트들의 ID이고 오른쪽 리스트는 선택된 컴포넌트들에서 발생할 수 있는 이벤트이다. 위의 경우 선택된 ID에 해당하는 컴포넌트의 종류는 라디오 버튼이고 거기에서 발생할 수 있는 이벤트는 Clicked 와 State Changed이다.

컴포넌트의 액션을 설정하기 위해서는

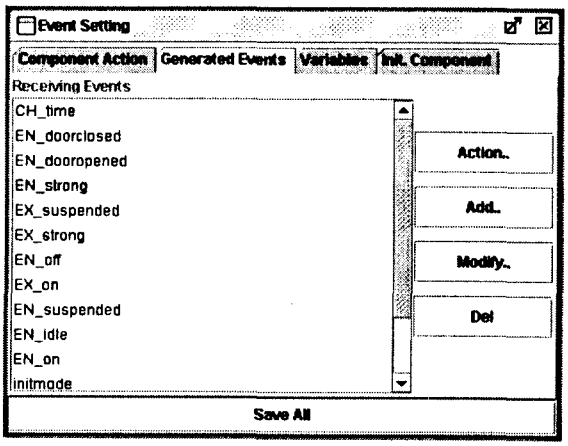
① 액션을 설정하고자하는 컴포넌트의 ID를 찾아 왼쪽에 보이는 리스트에서 선택한다.

② 액션이 어떤 이벤트가 발생했을 때 취해져야 할지를 오른쪽에서 보이는

리스트에서 선택한다.

③ Action 버튼을 클릭하여 수행 되어야 할 코드를 적어준다.

◆ ***Generated Events***



Generated Events

위의 화면에서 왼쪽에 보이는 리스트는 ASADAL에서 생성된 코드에서 전달되는 이벤트들이고, 오른쪽 버튼들은 이 이벤트들에 대한 액션을 정의하거나 이벤트를 추가, 삭제, 수정할 수 있도록 한다.

◆ ***Variables***

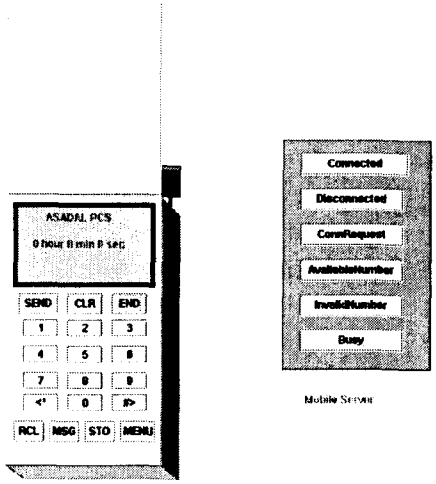
프로토타이핑에서 사용할 전역 변수를 선언하고 이 전역 변수들을 초기화하거나 삽입, 삭제 , 수정 하는 기능을 제공한다.

◆ ***Init. Component***

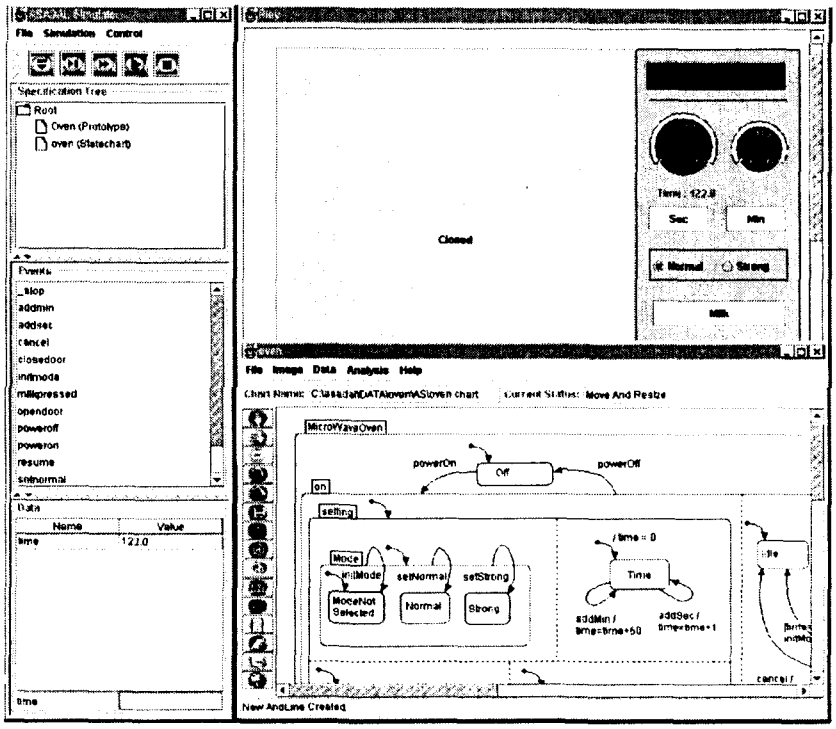
컴포넌트의 초기값을 설정할 때 사용할 수 있다. 예를 들면 슬라이더가 프로그램이 처음 시작 될 때 10으로 설정되어 있기를 원한다면 여기에 "var1.setValue (10);" (슬라이더의 ID가 1인 경우) 이라고 쓰면 된다.

프로토타이핑 결과의 시뮬레이션

이렇게 프로토타이핑을 만들게 되면 실제 DFD, Statechart로 만들어진 명세와 연동되어 수행될 수가 있다.



위 그림은 ASADAL 프로토타이핑 도구로 만든 핸드폰의 수행화면이다.



프로토타이핑은 그 자체만으로 수행될 수도 있고, Simulator와 연동하여서도 수

행될 수 있다. 위 그림은 ASADAL Simulator와 연동하여 수행되는 전자 오븐의 프로토타이핑 결과를 보여주는 것이다. 프로토타이핑 도구로부터 이벤트를 줄 수도 있고, Simulator의 이벤트 생성 창으로부터도 이벤트를 줄 수가 있다. 이러한 시뮬레이션은 실제 외부 사용자 입장에서 이벤트를 주면서 내부의 상태 흐름을 파악할 수 있어 명세를 검증하는데 큰 도움이 된다.

라. 전력 시스템 모델링 도구

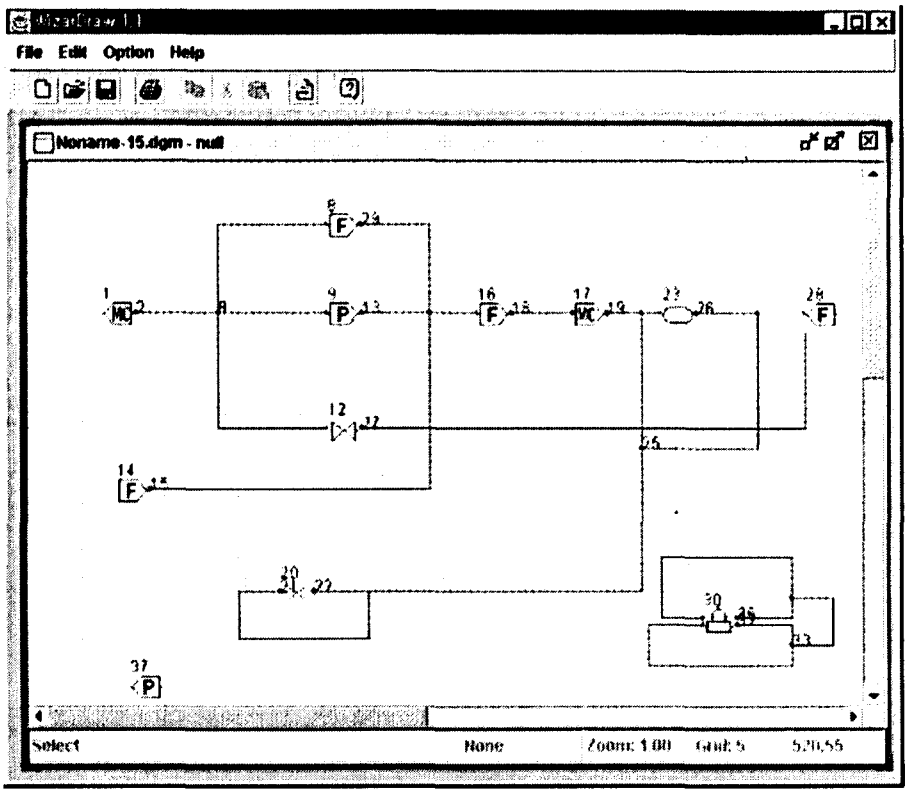
이번 년도에는 2차 년도에 개발한 전력 시스템 모델링 도구에서 에디터로부터 연결상태를 파악하는 정보를 추출하는 기능과, 그림 파일로 저장된 Component의 연결점을 설정하여 메뉴에 자동으로 추가하는 기능을 개발하였다. 여기서는 이 두가지 기능의 사용 설명을 살펴보도록 한다.

Model Editor 로부터 연결 관계 파일 생성 기능

커맨드 라인에서 "r.bat" 파일을 수행시키면 WizarDraw 1.1 이 수행된다. WizarDraw1.1에서 하는 그림을 그린 후 툴바에 있는 Model Editor 수행 버튼을 누르면 Model Editor가 수행되어 파일 두개를 생성한다.

하나의 파일은 그림 내의 게이트 및 파이프들의 위치 정보이고, 나머지 하나의 파일은 그 들간의 연결 관계를 나타내고 있다.

두 번째 파일은 연결 관계를 나타내고 있기 때문에 어떤 라인에도 연결되지 않은 게이트는 파일 내에 나타나지 않게 된다. 그리고 연결 상태를 판단하는 기준은 선의 끝점이 다른 라인의 일부에 연결이 되어있는가 이다.



예를 들어 위의 파일을 Model Editor의 입력으로 사용한 경우의 출력 파일은 아래와 같다. (모든 출력 파일의 형식은 동일하다.)

Noname-15.dgm_out1			Noname-15.dgm_out2
1	LMCP	55.0,100.0	2_3_6_5
2	Pipe	80.0,110.0	1 (1)
		140.0,110.0	8 (0)
3	Pipe	140.0,110.0	9 (0)
		140.0,195.0	12 (0)
		220.0,195.0	
5	Pipe	140.0,110.0	13_15_29
		220.0,110.0	8 (1)
6	Pipe	140.0,110.0	9 (1)
		140.0,45.0	14 (1)
		220.0,45.0	16 (0)
8	RBCF	220.0,35.0	
9	RBCP	220.0,100.0	18
12	VALV	220.0,185.0	16 (1)
13	Pipe	245.0,110.0	17 (0)
		330.0,110.0	
14	RBCF	65.0,230.0	19_22_25_26_21
15	Pipe	90.0,240.0	17 (1)
		295.0,240.0	20 (1)
		295.0,110.0	20 (0)
16	RBCF	330.0,100.0	23 (0)
17	RMCP	400.0,100.0	23 (1)
18	Pipe	355.0,110.0	
		400.0,110.0	27
19	Pipe	425.0,110.0	12 (1)
		465.0,110.0	28 (0)

```
20          LVALV    185.0,305.0
21          Pipe     185.0,315.0     32_33_36
                     155.0,315.0           30 (3)
                     155.0,350.0           30 (2)
                     250.0,350.0           30 (1)
                     250.0,315.0           30 (0)
                     210.0,315.0
22          Pipe     210.0,315.0
                     450.0,315.0
                     450.0,110.0
23          DRUM     465.0,100.0
25          Pipe     450.0,210.0
                     535.0,210.0
                     535.0,110.0
26          Pipe     490.0,110.0
                     535.0,110.0
27          Pipe     245.0,195.0
                     570.0,195.0
                     570.0,110.0
28          LBCF     570.0,100.0
29          Pipe      245.0,45.0
                      295.0,45.0
                     295.0,110.0
30           FWH     495.0,325.0
32          Pipe     520.0,340.0
                     560.0,340.0
                     560.0,370.0
                     455.0,370.0
                     455.0,340.0
                     495.0,340.0
33          Pipe     560.0,355.0
                     590.0,355.0
                     590.0,320.0
                     560.0,320.0
36          Pipe     520.0,335.0
                     560.0,335.0
                     560.0,290.0
                     465.0,290.0
                     465.0,335.0
                     495.0,335.0
37          LBCP      75.0,375.0
```

각 파일의 필드는 다음과 같다.

- Noname-15.dgm_out1

  * 게이트들 및 파이프는 컴포넌트라 한다.

  첫번째 필드 : 컴포넌트의 ID

  두번째 필드 : 컴포넌트의 종류

  세번째 필드 : 컴포넌트의 x, y 위치

- Noname-15.dgm_out2

  첫번째 필드 : 서로 연결된 파이프를 나타낸다. 즉, 서로 연결된 파이프들

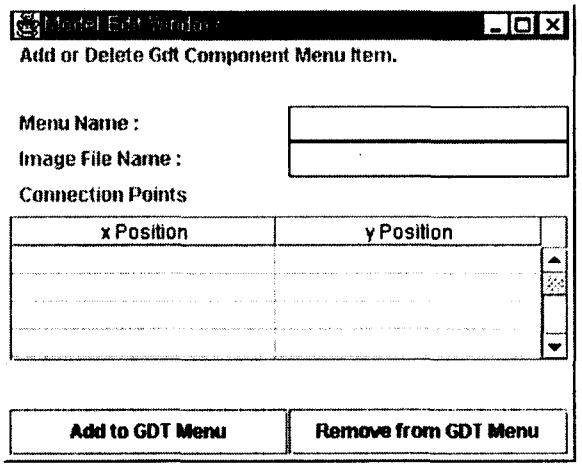
의 ID가 "_"로 연결되어있다.

두번째 필드 : 파이프에 연결된 게이트들을 나타낸다. 괄호 안의 숫자는 게이트가 파이프와 연결될 수 있는 부분 중 파이프와 연결된 곳의 번호를 나타낸다. 예를 들어 어떤 게이트가 파이프와 연결될 수 있는 부분이 두 곳이라면 한쪽은 0이고 다른 한쪽은 1이 된다.

Menu Editor

Menu Editor는 GDT에서 좀더 편리하게 선택 가능한 컴포넌트(마우스의 오른쪽 버튼을 클릭했을 때 나타나는 팝업 메뉴에서 Create를 선택하면 나타나게 되는 구성 요소들)를 쉽게 추가하고 삭제할 수 있도록 하기 위해 만들어졌다.

Model Editor를 실행하기 위해서는 GDT를 실행하는 것과 동일한 디렉토리에서 **"Model.bat"** 파일을 실행시키면 된다. 실제 실행 화면은 아래와 같다.



Menu Editor의 구성요소를 설명하면 다음과 같다.

■ **Menu Name**

GDT 메뉴에 어떤 이름으로 추가할지를 정해준다. 예를 들어 "RBCF"라

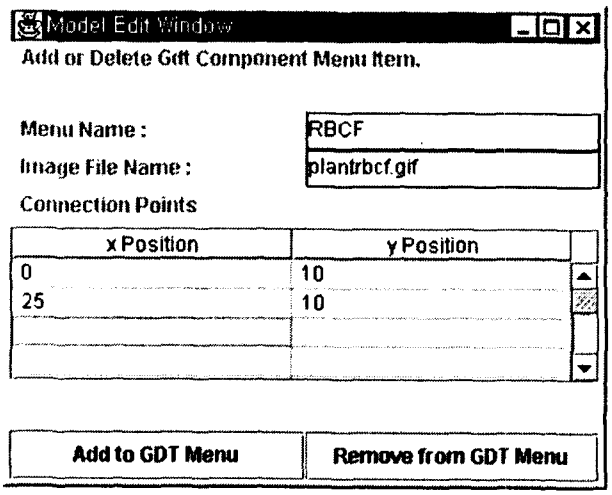
는 것을 추가하고 싶으면 여기에 RBCF라고 적어준다.

■ **Image File Name**

사용할 그림 파일을 적어준다. 즉 "RBCF"라는 것을 사용자가 선택한 후 마우스를 특정 위치에 클릭했을 때 화면에 나타나게 될 그림 파일을 적어준다. 만일 plantrbcf.gif를 사용한다면 여기에 plantrbcf.gif라고 적어준다. 주의해야 할 것은 모든 그림 파일은 현재 실행 디렉토리 밑에 있는 "\class\ddf\img" 디렉토리 밑에 있어야 한다는 것이다.

■ **Connection Points**

그림상의 어떤 위치가 다른 것(예를 들어 Pipe) 과 연결될 수 있는 점인지를 지정한다. 위치는 그림의 왼쪽 끝점이 (0,0) 이고 이것을 기준으로 설정한다. 만일 (0, 10), (25,10)위치에 연결점을 설정하고 싶다면 표 안에 다음과 같이 표기한다.



■ **Add to GDT Menu**

실제로 GDT 메뉴에 추가시킨다. 추가시킨 후 프로그램을 재설정하는 동안 약간의 시간이 걸린다. 마우스 포인터가 모래시계로 변해있는 동안 기다린 후 정상 상태로 돌아왔을 때 GDT를 실행시키면 원하는 메뉴

가 추가되어 있다.

■ Remove from GDT Menu

GDT 프로그램에서 Menu Name으로 지정된 메뉴를 삭제한다.

4. UCI 연구 내용 및 결과

UCI의 3차년도 연구 목표는 다음과 같다.

* RTS modeling 기법을 실시간 객체지향형으로 더욱 공고히 확립

- Object shadowing and Real-time group communication 지원

* 대규모 TMO형 real-time simulator를 더욱 효율적으로 신속히 execute 할 수 있는 분산 병렬 처리 engine 구축

- Formalization of APIs and I/O scheduler enhancement

UCI의 3차년도 연구 내용 및 결과는 다음과 같다.

연구 내용	연구 결과
(1) 실시간 객체지향 model 인 TMO를 바탕으로한 RTS modeling 기법을 더욱 공고히 확립하는 일환으로 distributed objects 간의 communication requirements를 더 줄이는 방법을 계속 연구 하였다. 지난해에 창안된 primary-shadow simulation 개념을 확장하여 공고히 확립하였다.	(1) 지난해에 창안된 primary-shadow simulation 개념을 확장하여 공고히 확립하였다. 이를 통하여 인접 TMO간의 data sharing을 효율적으로 하여 분산 처리의 효과를 극대화 하는 방법은 학문적 및 실용적 의의가 매우 크다 하겠다.

(2) 이전에 Group communication을 지원하고자 개발한 programmable data field channel (DFC)를 real-time group communication을 지원하는 version 으로 확장하였다. 명칭 또한 real-time multicast and memory-replication channel (RMMC) 로 바꾸었다. 필요한 execution engine 확장도 연구 하였다.	(2) 이전에 개발한 programmable data field channel (DFC)를 real-time group communication을 지원하는 version 으로 확장하였다. 효율적인 fault-tolerant real-time multicast를 지원 하는 방법 또한 개발 되었다. 명칭 또한 real-time multicast and memory-replication channel (RMMC) 로 바꾸었다. 필요한 execution engine 확장방법도 개발되었다.
(3) Windows NT 바탕으로 개발된 대규모 TMO형 real-time simulator를 더욱 효율적으로 신속히 execute 할 수 있는 분산 병렬 처리 engine 을 더욱 optimize 하였으며 중요한 개량은 I/O scheduler enhancement 이다. CORBA 환경에서의 RTS 지원도 개량 되었다.	(3) TMO-structured control system 개발시 deadline을 assign 하는 원칙을 formalize 하였다.
(4) 개발된 execution engine 의 application programming interface (API) 를 refine 및 formalize 하였다.	(4) Windows NT 바탕으로 개발된 대규모 TMO형 real-time simulator를 더욱 효율적으로 신속히 execute 할 수 있는 분산 병렬 처리 engine 을 더욱 optimize 하였으며 중요한 개량은 I/O scheduler enhancement 이다. 기본 개념은 I/O를 전적으로 수행하는 thread를 여러개 사용하되 timing analysis를 쉬웁게 하도롤 scheduling 하는 데 있다. CORBA 환경에서의 RTS 지원도 개량 되었다.

	(5) 개발된 execution engine 의 application programming interface (API)를 refine 및 formalize 하였다. 따라서 TMO programming을 더욱 수월케 하였으며 execution engine 의 portability 또한 향상 되었다. (6) TMO형 real-time simulator 구축 기법의 다른 측면에서의 demo 및 TMO programming 의 보다 이해하기 쉬운 illustration 으로서 DHRS (Distributged High-resolutikon Radar Simulator) 를 Windows NT machine network 상에서 동작 하도록 구현 하였다.

제 4 장 연구개발목표 달성도 및 대외기여도

본 장에서는 1차, 2차, 3차년도의 연구 목표 및 평가착안점에 입각하여 연구개발목표의 달성도를 기술하고, 본 연구 과제를 통하여 관련분야의 기술발전에 어떤 기여를 하였는가를 살펴보도록 하겠다.

제 1 절 1차년도 연구개발목표 달성도 및 대외기여도

1. 평가의 착안점 및 평가

<u>평가의 착안점</u>

(1). 분산 객체 모델러 및 시뮬레이터 시제품

a. 모델의 정형성, 표현력, 시뮬레이터의 Time Complexity가 거대하고 복잡한 실시간 시스템을 다룰 수 있는가?

b. 물리현상과 같은 실제 상황들이 표현 가능한가?

(2). 시뮬레이터 운영자 및 사용자를 위한 VR 인터페이스 개발 방법론 및 시제품

a. 운영자나 사용자가 훈련할 수 있을 정도의 VR인터페이스가 제작 가능한가?

(3). 실제 환경의 가시화 도구 시제품

a. 물리 현상 등 시뮬레이션 상황을 그대로 반영하여 보여주는가?

b. 애니메이션이 적당한 속도로 실행되는가?

(4). 명세의 실시간 시뮬레이션을 위한 방법 개발

a. 방법의 실행 가능성이 있고 타당한가(특히 실시간 시뮬레이션의 가능성)?

(5). 목표 시스템 시제품

a. 상세한 단계까지 명세되어 시뮬레이션되는가?

<u>평가의 착안점에 따른 목표 달성도에 대한 자체평가</u>

대부분의 목표를 달성했으며 진도상 약간의 구현이 남아있을 뿐이다. 2단계 첫 연도로서 연구의 가능성 검증을 위한 시제품이 성공적으로 만들어졌다.

2. 연구개발 달성도

1. 분산 객체 모델러 및 시뮬레이터 시제품

a. 모델의 정형성, 표현력, 시뮬레이터의 Time Complexity가 거대하고 복잡한 실시간 시스템을 다룰 수 있는가(100%)

b. 물리현상과 같은 실제 상황들이 표현 가능한가(100%)

2. 시뮬레이터 운영자 및 사용자를 위한 VR 인터페이스 개발 방법론 및 시제품

a. 운영자나 사용자가 훈련할 수 있을 정도의 VR인터페이스가 제작 가능한가(100%)

3. 실제 환경의 가시화 도구 시제품

a. 물리 현상 등 시뮬레이션 상황을 그대로 반영하여 보여주는가(80%)

b. 애니메이션이 적당한 속도로 실행되는가(80%)

4. 명세의 실시간 시뮬레이션을 위한 방법 개발

a. 방법의 실행 가능성이 있고 타당한가(특히 실시간 시뮬레이션의 가능성)(100%)

5. 목표 시스템 시제품

a. 상세한 단계까지 명세되어 시뮬레이션되는가(100%)

제 2 절 2차년도 연구개발목표 달성도 및 대외기여도

1. 평가의 착안점 및 평가

다음은 당해년도 계획서상에 명시되었던 각각의 평가의 착안점에 대한 자체 평가의 결과이다.

평가의 착안점	자 체 평 가
생성된 객체의 상세 디자인이 TMO 문법에 맞게 만들어졌는가?	ASADAL 명세로부터 자동으로 생성된 객체의 상세 디자인은 TMO 문법에 완전히 맞게 만들어졌으며 단순히 문법뿐 아니라 TMO의 특성을 최대한 살릴 수 있도록 만들어져 있으므로 TMO의 장점인 분산 시뮬레이션을 최대한 지원하고 있다.
생성된 객체들이 실시간 시뮬레이션이 되는가?	생성된 객체들은 자동 생성되었음에도 불구하고 그 성능에서 수동적으로 만든 객체들과의 차이를 보이지 않았으며 실시간 시뮬레이션에 문제 없었다.

실제의 거대 시스템 명세에서 TMO로의 자동 생성이 되는가?	실제 시스템에 적용, 시연하였으며 거대 시스템의 경우에도 기술적, 실제적으로 아무런 문제 없는 알고리즘을 이용하여 TMO 코드를 생성하였다.
VR 인터페이스가 저작 도구로 만들어지는가?	3차원 가상 환경이 저작 도구를 통해 만들어지도록 하였다.
VR 인터페이스가 사용자에게 친숙한가?	VR 인터페이스는 객체 지향적인 개념을 사용하여 VR 전문가가 아니더라도 쉽게 사용할 수 있게 하였으며 일반적인 사용자 대화형의 인터페이스를 사용, 사용자 친숙성을 높였다.
폼 정보가 쉽고 자연스럽게 표현되는가?	실제 하드웨어나 기계 시스템을 조립하듯 폼 정보를 입력하고 실제 현상이나 지켜야 할 제약 사항들을 자세한 수식 없이 여러 상징적인 표현을 사용하여 추상화함으로써 폼 정보를 쉽게 명세할 수 있게 하였다.
시뮬레이션 상황을 실제와 같게 보여주는가?	시뮬레이션 상황을 3차원으로 실제와 같이 보여주고 그 공간을 자연스럽게 Navigation할 수 있게 하였다.
시뮬레이션이 상황 분석에 충분한 속도로 되는가?	자동 생성된 TMO 코드는 실시간 시뮬레이터의 목적에 부합되도록 실제 시간과 완전히 같게 동작하도록 만들어져 있고 속도가 부족할

	경우 쉽게 여러 컴퓨터로 분산할 수 있으므로 속도 문제는 없다.
VR 인터페이스가 시뮬레이터와는 완전히 연동되는가?	자동 생성된 시뮬레이터 코드가 폼 정보로 표현된 VR 인터페이스와 완전히 연동되게 하였다.
배치형 시뮬레이터가 형태, 행위, 기능 명세 모두를 알맞게 시뮬레이션 하는가?	배치 시뮬레이터는 행위, 기능뿐 아니라 형태까지도 완전히 통합하여 다같이 동시에 연동되며 시뮬레이션 될 수 있도록 만들어졌다.
시뮬레이터의 모델링 도구의 유효성 및 완성도	시뮬레이터 모델링 도구는 이미 실제 사용자인 전력연구원의 연구원들로부터 현재 고액으로 구입, 사용하고 있는 CAD 도구 이상의 능력을 가지고 있음을 검증 받았다.

2. 연구개발 달성도

목 표	내 용 및 달성도
ASADAL 명세로부터 설계를 거쳐 최종적으로 TMO객체를 자동으로 생성해 내는 생성기 개	- ASADAL Statechart 명세로부터 Java Code까지의 Code 자동 생성기 (100%) - ASADAL DFD, Process 명세로부터 Java Code

발 (Automatic Code Generator)	까지의 Code 자동 생성기 (100%) - TMO 라이브러리와 연결 방법 (100%) - ASADAL 명세로부터 C++ 코드 자동 생성기 (100%) - ASADAL 명세로부터 TMO 코드 자동 생성기 개발 (100%)
가시화 도구 및 VR(Virtual Reality) 인터페이스 저작도구 개발	- 가시화 저작 도구 (100%) - Inference를 통한 공간 시뮬레이션 엔진 (100%) - 제약 사항 검사기 (100%)
실제 시스템 시뮬레이터의 시제품 개발	- 간단한 목표 시스템에 대한 시제품 (100%)
전력 시스템 모델링 도구 개발	- 전력 시스템 모델링 도구 개발 (100%) - 전력 시뮬레이션 입력 형태의 자동 파일 생성 (100%)

제 3 절 3차년도 연구개발목표 달성도 및 대외기여도

3차년도 연구 내용은 3장의 3절의 연구 목표에서 기술한 바와 같이 세부 내용에 있어서 약간의 수정 및 추가 사항이 있었다. 아래에서는 계획서에 서술된 내용을 기준으로 서술을 하였으며, 추가된 항목은 표시를 하였다.

1. 평가의 착안점 및 평가

여기서는 3차년도 계획서에 있는 평가의 착안점과 그에 대한 자체 평가를 하도록 한다.

(1) 대형 시스템에서의 다양한 요구사항의 분석 및 객체지향적 설계

a. *객체지향적 설계방법으로 대형 시스템에서의 다양한 실제 상황 및 환경을 모두 모델링 할 수 있는가?*

선박 시뮬레이터의 행위와 기능을 연구개발된 명세 방법으로 충분히 표현하고 모델링할 수 있었다.

b. 객체지향적인 분산시스템을 위한 설계가 충분히 지원되는가?

요구 분석 단계에는 분산 환경을 고려하지 않고 논리적인 객체지향적인 명세가 가능했으며, 디자인 단계에서 이들을 분산 시스템에 배치하는 방법이 충분히 지원되었다.

c. ASADAL Statechart와 DFD를 통해 명세된 모델들간의 상호 일관성이 유지되는가?

ASADAL Simulator를 통해 모델의 상호 일관성을 검증할 수 있었다.

d. Real-Time Simulator의 엄격한 Time Complexity를 효과적으로 표현할 수 있는가?

요구분석 단계에서는 TimeOut 과 같은 이벤트로 실시간 성질을 표현할 수 있었고, 디자인 단계에서는 TMO의 SpM과 SvM으로 실시간 성질을 반영할 수 있었다.

e. Performance와 재사용성 등을 증대시키기 위한 간단한 분산 설계와 같은 해결책이 지원되는가?

설계 시에 객체를 분산 노드에 배치할 수 있는 방법이 지원된다.

f. 설계결과의 시뮬레이터가 목표 시스템이 실행될 OS, Network 등과 관련된 파라미터들을 모두 반영할 수 있는가?

본 평가 항목은 TMO 기반의 코드 생성 결과에 대한 분석 도구에 대한 항목으로, 3장의 3절에서 기술한 사유로 수행되지 않았다.

g. 디자인 결과를 비교할 수 있는 툴이 데이터를 잘 관리하고 사용이 편리한가?

본 평가 항목은 추가된 항목인 병렬성을 고려한 디자인 모델 개발에서 평가하도록 한다.

(2) 자동으로 생성된 Real-Time Simulator Code

a. 자동생성 되어진 프로그램이 명세의 내용을 정확하게 모두 포함하는가?

Statechart, DFD 와 디자인 결정 사항이 함께 고려되어 정확하게 수행되었다.

b. 프로그램이 주어진 Time-Complexity 대로 실행되어지는가?

TMO의 SpM과 SvM을 이용하여 실시간 성질을 표현하게 되므로, 주

어진 요구사항대로 수행이 되었다.

c. 시뮬레이션 단계에서 측정된 값과 실제 수행시 얻어지는 값의 오차가 인정할수 있는 수준인가?

이 오차는 TMO의 수행 엔진의 오차와 관련된 것으로, TMO 수행 엔진의 오차는 거의 느끼지 못할 정도로 없었다.

(3) 가시화 시스템

a. Interface를 통한 공간 시뮬레이션 엔진과 가시화 저작 도구를 이용하여 Real-Time Simulation을 정확하게 가상공간에 표현할 수 있는가?

저작 도구에 VR system을 위한 요소들이 좀 더 보완 되어야 하지만, 저작 도구가 표현하는 방법론을 통하여 가상공간에 simulation을 정확하게 나타낼 수 있었다.

b. 가상 공간을 쉽게 모델링 할 수 있는가?

객체들의 모델링은 비교적 쉽게 할 수 있었으나, 가상 공간에서 중요한 요소인 presence, special effect등의 지원이 보완되어야 겠다.

c. 가시화 시스템이 실시간 시뮬레이터의 성능 저하를 가져오지 않는가?

초기의 간단한 모델로부터 시뮬레이션을 통하여 복잡한 모델로 점진적으로 진행을 하였기 때문에, 실시간 시뮬레이터의 성능 저하에 영향을 미치지 않을 정도의 가시화 시스템의 제작이 가능하였다.

d. 명세나 설계 전문가가 아닌 Target Domain 전문가가 쉽게 시뮬레이션 결과를 이해할 수 있는가?

VR기술을 이용하여 현실적인 가시화를 하였으므로, 쉽게 이해할 수 있

다.

(3) 병렬성을 고려한 디자인 모델(추가 항목)

a. 디자인 모델이 ASADAL 명세에서 표현된 행위와 기능 및 디자인 고려 사항을 잘 반영하고 있는가?

태스크가 ASADAL 명세의 DFD와 Statechart의 할당으로 만들어지므로, 행위와 기능이 모두 반영이 되었고, 성능과 실시간 성질을 태스크의 실행 정책 및, 연결 관계, 입출력 컴포넌트 등으로 반영하였다.

b. 디자인 결과를 분석하는 툴이 데이터를 잘 관리하고 사용이 편리한가?

디자인 결과는 LOG 파일 형태로 저장이 되고, Visual 하게 보여지므로, 관리가 용이하며 사용이 편리하다.

(4) ASADAL 도구 개선(추가 항목)

a. 개선된 Drawing 도구의 사용이 편리한가?

사용자 인터페이스를 Windows 스타일로 변경하였으며, Text Copy, Paste, Print Preview 등 유용한 기능을 추가하여 사용이 편리해졌다.

b. 프로토타이핑 도구가 제작이 쉽고 시뮬레이터와 잘 연동이 되는가?

버튼, 그래프, 도형 등을 사용자가 그리는 대로 실제 모습으로 전환해 주므로 제작이 간편하고, 시뮬레이터와 연동이 용이하도록 API와 도구를 지원한다.

(5) 전력 시스템 모델링 도구(추가 항목)

a. 컴포넌트의 연결점 설정과 메뉴 추가가 용이한가?

좌표를 통해 연결점을 설정할 수 있고, 메뉴 에디터를 통해 메뉴를 등록하면 바로 반영이 된다.

2. 연구개발 달성도

연구개발 달성도는 3장의 3절에서 작성한 연구 목표대로 그 달성도를 평가하였다.

목 표	내 용 및 달성도
선박 레이다 시스템의 사용자 요구 사항 명세	- 선박 레이다 시스템에 대한 다양한 요구 사항을 파악하고 ASADAL의 명세 도구인 DFD와 Statechart를 이용하여 명세한다.(100%) - 명세된 결과를 Graphical Simulator를 이용하여 분석한다.(100%)
검증된 명세를 기반으로 TMO 기반의 객체 지향적 설계	- 명세된 결과를 바탕으로 Performance, Complexity, Real-time property 등을 고려하여 TMO 기반의 객체 지향적인 설계를 한다.(100%) - 설계 객체들을 분산 환경에 배치한다.(100%)
객체 지향적 설계를 바탕으로 자동 코드 생성	- 객체 지향적인 설계를 토대로 TMO 기반의 코드를 자동 생성한다.(100%)

선박 레이다 시스템의 가시화	- VR(Virtual Reality) Interface를 구축하고, 자동 생성된 코드와 연동하여 가시화한다. (100%)
병렬성을 고려한 디자인 모델 개발	- 시스템의 병렬성을 고려한 디자인 모델(즉, 태스크 기반의 디자인 모델)을 명세로부터 구축하는 방법을 개발한다. (100%) - 태스크 기반의 디자인 결과를 바탕으로 실시간 OS 위에서 실행되는 코드를 자동생성한다. (100%) - 자동 생성된 결과를 수행함으로써 디자인 결과를 분석할 수 있는 도구를 개발한다. (100%)
ASADAL 도구 보완	- 1,2 차년도를 통해 만들어진 ASADAL 도구의 사용자 인터페이스를 개선한다. (100%) - 효과적인 시뮬레이션을 위하여 외부 사용자 인터페이스를 빠르게 제작하고 명세와 연결하여 시뮬레이션 할 수 있는 프로토타이핑 도구를 개발한다. (100%)
전력 시스템 모델링 도구 보완	- 2차년도에 제작한 전력 시스템 모델링 도구를 보완한다. (70%) : 보완한 도구에 대하여 사용상의 불편 및

	개선 사항이 다시 지적되었으며, 이 부분에 대해서 계속 피드백하기로 협의하였다.

제 4 절 UCI의 연구개발목표 달성도

	연구목표	달성도 (%)
1차년도 ('96)	* RTS modeling 기법을 실시간 객체지향형으로 확립 - Time-based synchronization and Group communication 지원 * 대규모 TMO형 real-time simulator를 더욱 효율적으로 신속히 execute 할 수 있는 분산 병렬 처리 engine 구축 - Windows NT based	100%
2차년도 ('97)	* RTS modeling 기법을 실시간 객체지향형으로 더욱 공고히 확립 - Minimization of communication requirements among distributed objects * 대규모 TMO형 real-time simulator를 더욱 효율적으로 신속히 execute 할 수 있는 분산 병렬 처리 engine 구축 - Thread scheduler enhancement and CORBA-compliant TMO support	100%
3차년도 ('98)	* RTS modeling 기법을 실시간 객체지향형으로 더욱 공고히 확립 - Object shadowing and Real-time group communication 지원	100%

	* 대규모 TMO형 real-time simulator를 더욱 효율적으로 신속히 execute 할 수 있는 분산 병렬 처리 engine 구축 - Formalization of APIs and I/O scheduler enhancement	

제 5 절 연구 결과의 대외 기여도

본 연구 개발 결과로 산출된 실시간 시뮬레이터 기술은 요구 분석, 설계, 코딩에 이르는 전 소프트웨어 개발 과정을 포함하는 체계적인 방법론이다. 개발된 실시간 시스템의 요구 분석 방법은 시스템의 행위, 기능 뿐 아니라 형태적인 면에 대한 관점으로 모델링이 가능하고, 이 모델 결과의 정확성을 검증하기 위한 분석 방법 및 도구가 개발되었다. 이러한 방법은, 시스템 개발 초기에 시스템이 가질 수 있는 문제점을 발견할 수 있기 때문에, 모든 코딩이 끝난 다음에 에러를 수정하는 방법과 비교하여 개발 비용을 단축할 수 있고, 시스템의 신뢰성을 높일 수가 있다. 그리고, 요구 분석 결과를 TMO 기반의 객체 지향 모델, 병렬성을 고려한 태스크 모델 등 실시간 디자인 모델에 연결하여 자동으로 코드를 생성하는 기술은, 전체 개발 시간을 단축하고, 요구 분석 결과와 최종 코드 사이에 생기는 불일치 문제를 해결할 수가 있게 된다. 본 연구 개발 결과 중 하나인 외부 환경을 3D 가시화 하는 기술은, 비행기 훈련 시스템, 선박 훈련 시스템 등 가상 현실감을 필요로 하는 시뮬레이터 프로그램에 효과적으로 접목되어 상품화될 수 있을 것으로 기대된다.

제 5 장 연구 개발 결과의 활용 계획

본 연구는 3차년도에 1, 2차 년도의 연구를 바탕으로 실제 거대 시스템인 선박 시뮬레이터에 적용하여 실효성을 검증하였다. 이번에 적용된 시뮬레이터의 경우, 선박 조정자들을 위한 훈련 프로그램으로서, 실시간 명세 방법을 통해 요구 사항을 정확히 분석하고, TMO를 통해 실시간 성질을 만족시키는 디자인과 구현을 하였으며, VR 인터페이스를 통해 현실감있는 외부 환경을 조성하여 성공적인 활용이 가능하였다. 이와 같은 모의 훈련 프로그램 외에도 원자력 발전소 시스템이나 교통 제어 시스템 등과 같은 위험도가 높고 복잡한 시스템에 적용하여, 실제 시스템을 구축하기 이전에 미리 문제점을 파악하여 신뢰성을 높이고, 개발 비용을 줄일 수 있게 된다.

현재 본 연구를 통해 개발된 방법은 ASADAL 이라는 이름의 통합 도구로 만들어졌으며, 연구 결과의 활용을 위해 관심 있는 업체들과의 Consortium 구성을 계획 중에 있다. 이러한 Consortium의 목표는 본 연구 과제를 통해 산출된 실시간 시스템의 체계적인 개발 방법을 소개하여 실시간 시스템 개발에 있어서 아직 낙후되어 있는 국내의 현실을 개선하는 것이다. 이를 위하여 2000년도에 Comdex Korea2000에 참가하여 연구 결과를 소개하고, Consortium의 취지를 설명하였다.

제 6 장 참고 문헌

[Boehm] B. W. Boehm, "A Spiral Model of Software Development and Enhancement," IEEE Computer, pp. 61-72, May 1988.

[Goldsack] S. J. Goldsack and A. C. W. Finkelstein, "Requirements Engineering for Real-Time Systems," Software Engineering Journal, pp. 101-115, May 1991.

[Gomaa93] H.Gomaa, "Software Design Methods for Concurrent and Real-Time Systems(DARTS)," Communications of ACM, pp. 938-949, Sep. 1984

[Kang95] Kyo C. Kang and Kwang. I. Ko, "PARTS : A Temporal Logic-Based Real-Time Software Specification Method Supporting Multiple Viewpoints", International Journal of Software Engineering and Knowledge Engineering, 1995, Vol. 5, No. 3, pp 407-422.

[Kang98a] Kyo C. Kang, Kwan W. Lee, Ji Y. Lee, Gerard J. Kim, and Hye J. Kim, "Refinement and Validation of Software Requirements using Incremental Simulation," "IEICE(The Institute of Electronics, Information and Communication Engineering) Trans. on Information and Systems," Vol.E81-D, No.2, pp. 171-182, February 1998.

[Kang98b] "Prototype = Function + Behavior + Form," Software Engineering Notes, Vol.23, No. 4, pp.44-49, July 1998.

[Kang98c] Kyo C. Kang, Kwan W. Lee, Ji Y. Lee, and Gerard J. Kim, "ASADAL/SIM: An Incremental Multi-level Simulation and Analysis Tool for Real-Time Software Specifications," Software: Practice and Experience, Vol. 28, Issue 4, pp. 445-462,

April 1998.

[Kang98d] Kyo C. Kang and Kwang Il Ko, "ASADAL/PROVER: A toolset for Verifying Temporal Properties of Real-Time System Specifications in Statechart," The Institute of Electronics, Information and Communication Engineering(IEICE) Transactions on Information and Systems, Vol. E82-D, No. 2, pp. 398-411, 1999.

[Kang00] Kyo C. Kang, Gerard J. Kim, Ji Y. Lee, Hye J. Kim, "Co-development of Real-Time Systems and Their Environments", accepted to be published at the 7th Asia-Pacific Software Engineering Conference (APSEC), Singapore, December 5-8, 2000.

[Lee99] Ji Y. Lee, Kyo C. Kang, Gerard J. Kim, and Hye J. Kim, "Adding Form to Real-Time System Specification and Simulation," International Journal of Software Engineering and Knowledge Engineering (IJSEKE), World Scientific Publishing Co., Vol.9, No.5, pp. 643-661, Oct. 1999

[Lee00] Ji Y. Lee, Hye J. Kim, and Kyo C. Kang, "A Real World Object Modeling Method for Creating Simulation Environment of Real-time Systems" , accepted to be published at the ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), Minneapolis, Minnesota USA, October 15-19, 2000.

[Luqi] Luqi and V. Berzins, "Rapidly Prototyping Real-Time Systems," IEEE Software, pp. 25-36, September 1988.

[Lila] Liu, C.L. and Layland, "Scheduling algorithms for multiprogramming in a hard real-time environement," J. Assoc. Comput. Mach., vol.20, pp. 46-61, 1973

[Mark93] M.H.Klein, T.Ralya, B.Pollak and R.Obenza, A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems, Software Engineering Institute and Carnegie-Mellon University, Kluwer Academic Publishers, 1993

[Pnueli91] A.Pnueli and M.Shalev, "What is in a Step: On the Semantics of Statecharts," Theoretical Aspects of Computer Science, LNCS 298, Springer-Verlag, pp. 244-264, 1991

[Stankovic90] J.A.Stankovic and K.Ramamritham, "What is Predictability for Real-Time Systems," Journal of Real-Time Systems, Vol.2, pp. 247-254, Dec. 1990