GOVP1199502348

제 1 차년도
연차보고서

6f
150

국산 주전산기용 소프트웨어 개발 지원도구 개발
Development of Software Development Support Tool for Domestic Computers

그래픽 사용자접속 지원도구 개발
Development of Graphical User Interface Builder

연 구 기 관
한국과학기술연구원
시스템공학연구소

과 학 기 술 처

배 포 선

사본번호	부수	배 포 처
1/150	1	시스템공학연구소 영구보존용
2/150	1	시스템공학연구소 도서과 보관용
3/150-6/150	4	시스템공학연구소 연구관리과 보관용
7/150-8/150	2	시스템공학연구소 소프트웨어공학부
9/150-11/150	3	과학기술처
12/150-150/150	139	기타배포기관

제 출 문

과학기술처장관 귀하

본 보고서를 "국산주전산기용 소프트웨어 개발 지원도구" 과제의(세부과제 "그래픽 사용자접속 지원도구 개발"의) 연차보고서로 제출합니다.

1994년 10월 20일

주관연구기관명 : 한국과학기술연구원
시스템공학연구소

총괄연구책임자 : 이 단 형

연 구 책 임 자 : 신 규 상

선 임 연 구 원 : 노 영 주

연 구 원 : 이 선 미, 박 현 민
이 영 철, 손 경 순

연 구 조 원 : 이 형

참여기업연구원 : 이 영 무, 강 미 영

여 백

요 약 문

I. 제 목

그래픽 사용자접속 지원도구 개발(2년중 1차년도)

II. 연구개발의 목적 및 중요성

소프트웨어 생산기술의 발전사는 방법론 및 그를 지원하는 도구의 변천사를 중심으로 이해할 수 있다. 이해의 핵심은 복잡도를 다루어온 방법을 개발해온 과정에 있다. 컴퓨터 시대의 초창기인 50년대와 60년대에는 주로 알고리즘의 복잡도를 해결하기 위한 프로그래밍 언어 개념의 개발에 노력이 집중되었고, 이것이 어느정도 해결된 70년대에는 문제의 복잡도를 해결하기 위한 노력으로 개발 방법들이 속속들이 개발되었다. 80년대에 들어서는 이러한 각종의 방법들의 산재에 의한 환경의 복잡도를 해결하기 위한 개발 및 관리 방법론이 개발되었고, 80년대 중반 이후에는 이러한 방법론들을 지원하는 도구의 개발이 진행되었다. 개발환경의 정비에 따라 80년대 후반에는 전문 지식의 복잡도를 은닉한 소프트웨어 요소들을 이용하여 생산성 및 품질을 향상 시키고자하는 재사용 기술의 개발이 추구되기 시작하여, 90년대에 들어서면서 재사용을 기술적으로 지원하는 객체지향 기술 및 도구의 개발이 본격적으로 이루어 졌다. 이와 더불어 사용자의 이용 복잡성을 해결하는 방법으로 GUI 기술에 대한 발전이 급속도로 이루어졌다. 그결과로 UNIX 계열에서는 OSF/Motif가, PC에서는 MS-Windows가 거의 표준 GUI로서의 자리

를 굳혀가고 있으며, 이를 지원하는 자동화 도구들이 개발되어 CASE 도구의 한 요소 도구로서 자리잡혀가고 있으며, 이 영향으로 시각 프로그래밍 기술 개발이 고조되고 있으며 CASE 도구와의 통합을 위한 연구개발이 활발히 진행되고 있다. 이미 GUI 개발 자동화 도구는 현업에서 주요한 개발 지원도구로 자리하고 있으며, 국내에도 5종류 정도의 Motif GUI 개발 지원도구들이 판매되고 있다.

GUI 도구와 같은 프로그래밍 자동화 도구는 프로그램의 단순한 개발 지원에서 그 역할이 한정되지 않고 이 도구를 통한 지속적인 프로그램의 유지보수가 이루어지므로 해당 소프트웨어는 이 개발시에 사용한 도구에 종속이 지속적으로 유지된다. 따라서 이 도구의 개발은 소프트웨어의 개발 및 유지보수에 관한 핵심 기술이라고 볼 수 있다. 즉, 이 기술의 확보는 대외 경쟁력의 확보를 의미한다.

그래픽 사용자접속 지원도구를 개발하는 목적은 의미적으로는 GUI 개발 국내 기술력의 확보에 있으며, 기술적으로는 자동화 기술의 개발에 있으며, 이용 측면에서는 Motif 스타일의 GUI 개발에 있으며, 한편으로는 들무새 시스템의 GUI 개발 지원에도 그 목적이 있다.

이러한 목적을 달성하기 위하여 다음과 같은 목표의 달성을 추구하고 있다.

- 그래픽 사용자 접속부 개발 기간의 50% 단축
- 그래픽 사용자 접속부 요구사항 분석기간의 50% 단축
- 소프트웨어 개발 자동화 기술의 획득

III. 연구개발의 내용 및 범위

본 연구는 국산 주전산기용 소프트웨어 개발 지원도구 들무새 시스템 개발 과제의 세부과제로 진행되고 있다. 들무새 시스템은 정보저장소를 중심으로하는

통합된 지원도구를 목표로 하고 있으며, 대략 프로세스(process) 명세화 도구군, 자료(data) 명세화 도구군과, 이를 배경으로 한 코드생성 도구, 사용자 접속 도구, 그리고 하부구조인 정보저장소로 구성되어 있다.

따라서 사용자 접속 지원도구는 들무새 시스템의 하나의 부속 시스템으로서 Motif 스타일의 사용자 인터페이스 개발을 지원해주는 도구의 역할을 하며, 한편으로는 들무새 시스템의 GUI 개발에 이용하고자 개발하고 있다.

본 연구는 1차년도인 당해 년도에는 프로토타잎의 개발과 실용 버젼의 개발을 위한 분석 및 설계로 진행되었다.

1. 프로토타잎의 개발

들무새 시스템의 GUI 개발을 지원하기 위하여 우선적으로 개발함.

- 작업절차 관리기

도구에서 지원하는 개발작업을 통합하여 관리

- 시각 명세 작업대

사용자 접속부의 개발을 시각적으로 지원

- 명세언어

사용자 접속부를 텍스트로 표현하는 정형화된 언어

- 명세코드 번역기

명세언어로 표현된 코드를 그래픽 표현으로 바꿈

- 명세코드 생성기

그래픽 명세를 명세언어로 된 코드로 바꿈

- C/C++ 원시코드 생성기

명세코드를 컴파일 가능한 C/C++ 원시코드 군으로 바꿈

2. 실용도구의 분석 및 기초 설계

프로토타잎의 지식을 바탕으로 실용도구를 체계적으로 개발함.

- Rumbaugh의 OMT 방법으로 표현
- Object Diagram
- State Diagram
- Dataflow Diagram

IV. 연구개발 결과 및 활용에 대한 건의

본 연구는 총 2년 중 1차년도 과제를 수행하였으며, 그 결과물은 다음과 같다.

- GUI Mosaic 프로토타잎
- GUI Mosaic 실용 버젼 1.0 분석 및 설계

GUI Mosaic 프로토타잎은 실용 버젼의 주요기능의 대부분을 갖추고 있으나, 일반 사용자들이 사용하기에는 아직 세련되지 못한 상태이다. 그러나 전문가들이 조심하여 사용할 수 있는 수준으로 들무새 시스템의 GUI 개발에 이용될 예정이다. 또한 이 도구의 개발에 이용된 각종의 기술들은 자동화 도구를 개발하기 위해서는 반드시 필요한 기술로서 타 자동화 도구를 개발함에 있어서 중요한 참고 기술이 될 것이다.

한편, 실용 버젼의 개발에 이용하고 있는 Rumbaugh의 OMT 방법은 유력한 객체지향 개발 방법들 중의 하나로, 본 연구개발 문서들은 국내에서의 객체지향 개발 사례로 이용될 수 있다.

SUMMARY

I. Title

Development of a Graphical User Interface Builder

II. Importance and Objectives

The brief history of software engineering can be explained as challeges to software development methodologies and tools. Generally speaking, there were challenges to solve algorithm complexity by evolution in programming language concepts during 1950's and 60's, problem complexity by development of various methods during 1970's, environmental complexity by integration of those methods into uniform methodologies and their supporting tools during 1980's, domain complexity by application of new paradigms such as reuse, object-orientation, etc, and interfacing complexity between users and applications by building standards of user interfaces based on graphical windows such as OSF/Motif, MS-Windows, etc. during the early 1990's. And now, visual programming technology including programming automation based on reuse is becomming one of the hot technical issues.

GUI tools which make possible the easy construction of GUI are generally accepted as a productive method and provided users as an important

component of CASE. It is expected that GUI tools will drive the integration of CASE and visual programming tools. So, the importance of our developing a GUI tool is that it makes us to obtain the automation-related techniques as well as the tool itself.

The objectives of the study are:

- 50% decrease of GUI development efforts in time
- 50% decrease of GUI analysis efforts in time
- Defining the basic automation techniques

III. Scope

This study is a sub-project of the project, "Development of a CASE for the National Computing System, TICOM: Deulmusae". Deulmusae consists of the process specification tools, data specification tools, a source code generator, a GUI tool, and a information repository.

The name of the GUI tool which we are developing is GUI Mosaic.

The scope of the study are as follows:

1. Development of the GUI Mosaic prototype
   - Working Procedure Manager(WPM)
   - Visual Specification Workbench(VSW)
   - Formal Specification Language(FSL)
   - FSL Code Interpreter(FCI)
   - FSL Code Generator(FCGen)

- C Code Generator(CGen)

- C++ Code Generator(CppGen)

2. Analysis and system design of the practical GUI Mosaic

- Documentation using the Rumbaugh's OMT method

- Object Diagrams

- State Diagrams

- Dataflow Diagrams

IV. Results and Suggestions

The study is the two-year long project. The first-year results of the study are as follows:

- GUI Mosaic prototype

- Analysis and system design documents on the practical GUI Mosaic

The GUI Mosaic prototype has most core functions of its practical version. It, however, is not a mature version for general users.

The prototype will be used for developing the GUIs of Deulmusae. The obtained automation technology could be a good reference to developers interested in automated tools. And the analysis and design documents can be used for an case study on object-oriented software development in Korea.

여 백

CONTENTS

목 차

여 백

제 1 장 서 론

여 백

제 1 장 서론

제 1 절 연구개발의 필요성

1. 기술적 필요성

소프트웨어 생산기술의 발전사는 방법론 및 그를 지원하는 도구의 변천사로 볼 수 있다. 공학적인 접근이 없었던 시절에 구조적인 접근법은 생산기술의 새로운 지평을 열었으며 지금도 가장 많이 사용되는 방법으로 자리하고 있다. 그러나 정보와 작업이 복잡해짐에 따라 실세계를 보다 쉽게 컴퓨터의 세계와 연결시킬 수 있는 방법으로 객체지향 방법이 그 이용범위를 넓혀가고 있다. 이러한 사조들의 관점은 목적 시스템을 어떻게 표현해 갈 것인가에 중점을 두고있는 반면에 컴퓨터 자체의 이용 내지는 표현능력을 향상시키고자 하는 관점에서는 컴퓨터에서 가능한 모든 정보를 시각화하려는 사조도 있다.

이러한 사조는 생산기술 측면에서는 시각 프로그래밍으로 나타나게 되었다. 이 기술은 보다 체계를 잡아가고 있으며, 대략 GUI(Graphical User Interface) front-end, Lanaguage front-end, DB front-end의 유형으로 발전하고 있다. DBMS의 세계적인 표준을 바탕으로 DB front-end로서의 시각 프로그래밍 기술 및 도구는 상용화된 상태에 있으며, GUI front-end는 이제 일반화되고 있는 GUI에 힘입어 실용화의 단계로 접어들고 있다. 그러나 Language front-end의 경우는 언어 자체의 모호함 내지는 기존 언어의 시각화의 한계성으로 인하여 다른 두가지 경우와는 다르게 실용화 단계에 까지는 이르지 못하고 있다.

GUI front-end는 일반적으로 GUI 도구라고 불리고 있으며, 응용 소프트웨어 개발에

있어서 화면의 프로그래밍이 반 이상을 차지하고 있는 시점에서 이 기술 및 도구의 개발은 프로그래밍 생산성을 획기적으로 향상 시킬 수 있는 기술로 인식되고 있다. 이러한 특성을 감안하듯 이미 PC 및 WS에서 운용되는 도구들이 속속들이 개발되고 있으며 국내시장에도 선을 보이고 있다.

한편, 시각 프로그래밍 기술은 기존의 CASE 기술의 흐름을 바꿀 수 있는 충분한 요소를 내포하고 있는 것으로 인식되고 있으며, 유수한 기업들이 두 기술의 통합을 목표로 연구개발을 추진하고 있다.

2. 경제, 산업적 필요성

소프트웨어의 얼굴 역할을 하는 GUI는 거의 모든 소프트웨어에서 없어서는 안될 부분으로 자리잡고 있으며, 그 시장도 급속한 속도로 팽창하고 있다. 이에 따라, 외국의 선발 상품들이 국내시장에 상륙하고 있으며, 일본의 경우는 일본어화를 마친 상태로 적극적인 판매에 나서고 있다. 머지않은 장래에 우리도 일본의 경우처럼 될 것으로 예상하고 있으며, 도구의 생산성 향상 효과로 말미암아 GUI를 갖는 모든 소프트웨어의 개발에 이용될 것이 확실하다.

따라서 현재는 시장의 규모가 분명치 않으나, 향후 1년내에 구체적인 규모를 예측할 수 있을 것이다. 도구의 이용 편리성에 비추어 볼 때, 이 도구의 시장 규모는 현재의 CASE 도구 시장과 같은 수준에 이를 것으로 예측된다.

3. 사회, 문화적 측면

생활수준의 향상으로 멀게만 생각했던 컴퓨터는 가정에까지 보급되어 사람들의 생활양식을 바꾸어 놓고 있다. 학생들의 공부를 도와주는 컴퓨터 가정교사, 주부들의 생활양식

에 지대한 영향을 미치고 있는 가정의 자동화(Home Automation), 사무실과 연결된 컴퓨터를 사용한 업무 처리등은 사회, 문화적 양식을 바꾸기에 충분하다. 그러나 컴퓨터는 하드웨어 만으로는 존재할 수 없고 소프트웨어와 공존해야 하는데 생활양식의 변화를 가져오는 소프트웨어의 수요는 문화적 양식에 따라 다양해지고, 섬세해지리라 본다. 이러한 수요에 대처하기 위해 소프트웨어 개발 속도는 지금과는 획기적인 향상을 이룩하여야 한다. 또한 일반인들이 자신에 맞는 소프트웨어를 스스로 개발할 수 있는 환경이 요구된다. 본 연구개발의 산물은 이러한 기술을 진일보 시키는 계기가 될 것으로 기대하고 있다.

제 2 절 연구개발의 목표 및 내용

본 연구개발 2년의 최종목표는 국산 주전산기용 소프트웨어 개발 지원도구인 들무새 시스템의 단위 조구인 GUI Mosaic 시작품의 개발에 있으며, 내용적으로는 다음과 같다.

- 그래픽 사용자 접속부 대화형 명세와 코드의 자동생성을 위한 지원도구의 개발
  - . 그래픽 사용자 접속부 개발 기간의 50% 단축
  - . 그래픽 사용자 접속부 요구사항 분석기간의 50% 단축
  - . 들무새 시스템의 GUI 개발에 이용

당해 년도는 2년의 연구개발 기간 중에서 1차년도로서 GUI Mosaic 프로토타잎의 개발과 실용 버젼의 개발을 위한 분석 및 설계로 진행되었다.

ㅇ 프로토타잎의 개발

들무새 시스템의 GUI 개발을 지원하기 위하여 우선적으로 개발함.

- 작업절차 관리기

  도구에서 지원하는 개발작업을 통합하여 관리

- 시각 명세 작업대

사용자 접속부의 개발을 시각적으로 지원

- 명세언어

사용자 접속부를 텍스트로 표현하는 정형화된 언어

- 명세코드 번역기

명세언어로 표현된 코드를 그래픽 표현으로 바꿈

- 명세코드 생성기

그래픽 명세를 명세언어로 된 코드로 바꿈

- C/C++ 원시코드 생성기

명세코드를 컴파일 가능한 C/C++ 원시코드 군으로 바꿈

○ 실용도구의 분석 및 기초 설계

프로토타잎의 지식을 바탕으로 실용 도구를 체계적으로 개발함.

- Rumbaugh의 OMT 방법으로 표현

- Object Diagram

- State Diagram

- Dataflow Diagram

당해 년도의 결과물은 GUI Mosaic 프로토타잎과 GUI Mosaic 실용 버젼 1.0 분석 및 설계 문서이다. GUI Mosaic 프로토타잎은 실용 버젼의 주요기능의 대부분을 갖추고 있으나, 일반 사용자들이 사용하기에는 아직 세련되지 못한 상태이다. 그러나 전문가들이 조심하여 사용할 수 있는 수준으로 들무새 시스템의 GUI 개발에 이용될 예정이다. 또한 이 도구의 개발에 이용된 각종의 기술들은 자동화 도구를 개발하기 위해서는 반드시 필요한 기술로서 타 자동화 도구를 개발함에 있어서 중요한 참고 기술이 될 것이다.

한편, 실용 버젼의 개발에 이용하고 있는 Rumbaugh의 OMT 방법은 유력한 객체지향 개발 방법들 중의 하나로, 본 연구개발 문서들은 국내에서의 객체지향 개발 사례로 이용될 수 있다.

제 2 장 기술 현황 분석

여 백

제 2 장 기술현황 분석

제 1 절 개요

과거 컴퓨터가 귀한 시절에는 컴퓨터는 두려운 존재였다. 생소한 용어들로 구성된 많은 명령어들을 알고 있어야 했으며, 명령어를 사용함에 있어서도 조금의 오차도 없이 엄격해야 했음에도, 컴퓨터는 친절하지 않았다. 그러나 이제 컴퓨터는 더이상 불친절하지 않다. 더우기 화려한 치장을 하고 사용자에게 보다 양질의 서비스를 제공하기 위하여 노력하고 있다. 반대로 사용자들은 그많은 명령어들을 일일이 외우고 있지 않아도 된다. 조금 틀리더라도 컴퓨터는 친절하게 가르쳐 준다. 컴퓨터는 더이상 두려움의 대상이 아니다.

이같은 컴퓨터의 대변신을 적극 도와준 일등공신은 GUI, 즉 그래픽 사용자 인터페이스이다. GUI는 컴퓨터 오퍼레이팅 시스템과 사용자 사이의 상호 작용을 그래픽으로 표현해 줌으로써 교량역할을 수행하여 컴퓨터를 배우기 쉽고, 사용하기 쉽게 도와주고 컴퓨터의 사용을 확산시켜주고 사용자의 생산성을 증가시켜 주었다.

GUI는 대개 공통된 특성을 가지고 있다. 첫째 공통된 모양과 구조를 가지고 있다. 아이콘을 이용하여 컴퓨터의 여러가지 요소들을 표현하고 있으며, 메뉴방식으로 원하는 것을 차례로 찾아갈 수 있다. 또한 버튼, 슬라이더, 다이알로그 박스 등과 같은 그래픽적인 요소들을 주로 사용하고 있으며, 컴퓨터와 사용자를 대화식 환경으로 이끌어 간다. 둘째는 윈도우를 가지고 있다. 윈도우는 여러 응용프로그램의 실행을 가능하게 하는 다중 창을 지원하며 모양조절과 스크롤 기능을 제공하여 작업의 효율성을 높여준다. 세째는 공통된 조작(operation)을 가지고

있다. 즉 윈도우의 열기 닫기 크기 조절 등의 윈도우 조작과, 화일의 생성, 열기, 저장하기, 인쇄하기 등의 화일 조작, 편집을 위한 자르기, 복사하기, 옮기기 등의 편집 조작을 공통적으로 제공하고 있다.

위와 같은 특성들을 가지고 GUI는 일반사용자에게는 "Look and Feel"을 제공하고, 개발자에게는 제어패널로서의 기능을 제공하고 있다.

제 2 절 GUI 역사

그래픽 사용자 인터페이스가 일반사용자에게 친숙해 지기까지는 Apple 사의 매킨토시와 Microsoft 사의 Windows가 큰 역할을 하였다. 이는 1980년대 후반의 일이며 GUI의 효시라고 할 수 있는 시스템은 그로부터 10년쯤 전인 1970년대 후반에 발표되었다. 이 시스템은 Star 라는 이름을 가진 시스템으로 Xerox 사의 연구센터인 Palo Alto Research Center 에서 개발하였다.

Apple 사의 Steve Jobs 는 이 시스템의 아이디어를 적용하여 1984년에 매킨토시를 발표하였다. 매킨토시는 컴퓨터를 모르는 일반사용자들의 컴퓨터에 대한 두려움을 감소시켜 주면서 선풍적인 인기를 얻게 되었다. 매킨토시가 상품화에 성공하자 Microsoft 사의 Bill Gates도 GUI를 적용하기 위한 장기적인 계획을 세워 1989년에 Windows를 발표하였다. 또한 IBM도 Windows의 영향을 받아 Presentation Manager를 만들었는데 이 시스템은 매킨토시와는 다른 모양을 가지고 있다[1].

한편 MIT에서는 DEC 의 지원을 받은 Athena 연구과제를 통하여 그래픽스와 통신을 하나로 묶은 X 를 1984년 발표하였다. X 라는 이름은 Stanford 의 W를 전신으로 하였기 때문에 다음 알파벳을 따서 붙여지게 된 이름으로 X 의 출현으로

UNIX를 OS로 하는 시스템들은 거의 X를 기반으로 하고 있다. 1988년에는 여러 컴퓨터 회사들이 X 컨소시엄을 구성하여 X11 시스템을 발표하게 되었는데 1994년 5월 발표된 X11R6버전이 가장 최근의 시스템이다.

1990년대에는 많은 GUI도구들이 발표되었는데 UNIX 세계에서는 AT & T 와 Sun Microsystems에 의해 발표된 Open Look과 IBM, DEC, HP 가 주도하는 OSF(Open Software Foundation) 의 Motif로 양분되었다. 이들 두 시스템은 모두 X Window 시스템을 기본으로 하고 있다. 최근에는 UNIX 와 GUI 의 표준화를 위해서 OSF의 Motif를 표준으로 정하려는 움직임을 보이고 있다[2].

1990년대 GUI의 두드러진 특징은 응용 프로그램에서 사용자 인터페이스를 분리하여 개발과 관리를 지원할 수 있는 시스템이 많이 출현하고 있다는 것이다. 더우기 이들 시스템들은 사용자 인터페이스 개발의 지원과 함께 코드까지 생성해주고 있어 소프트웨어 개발자들을 위한 CASE Tool 로 자리잡고 있다.

제 3 절 GUI 스타일

지금 컴퓨터 시장에는 많은 종류의 GUI 들이 발표되어 있고 또한 치열한 경쟁을 벌이고 있다. 이들을 분류해 보면 세가지로 나눌 수 있다.

첫째는 Apple 컴퓨터를 위한 GUI 인 매킨토시 시스템으로 이 시스템은 GUI가 ROM 안에 완벽하게 통합되어 있어 컴퓨터 부팅시에 GUI가 출현하여 모든 작업을 GUI를 통해서 처리할 수 있다. 매킨토시는 하드웨어 시스템과 통합되어 있기 때문에 설치하기가 쉽고, 많은 응용 프로그램들이 구비되어 있으며, 응용 프로그램들은 안정적이며 서로 호환성을 가지고 있고, 통신기능이 내장되어 있다는 장점이 있다[3]. 반면에 협동적으로 이루어지는 다중작업은 매우 불량하다.

둘째는 Intel 기반의 GUI들로서 여기에는 IBM의 OS/2 PM과 Microsoft 사의 Windows, H/P의 NewWave와 같은 시스템 들이 있다. OS/2 의 Presentation Manager는 응용 프로그램들은 안정적이며 서로 호환성을 가지고 있고, 다중작업 기능이 내장되어 있고, 온라인 도움말(help)을 제공하며, 매크로와 작업 자동화 기능이 포함되어 있다. 반면에 이 시스템은 마케팅이 원활하게 이루어지지 않고 있으며, 응용 프로그램의 수가 적다.

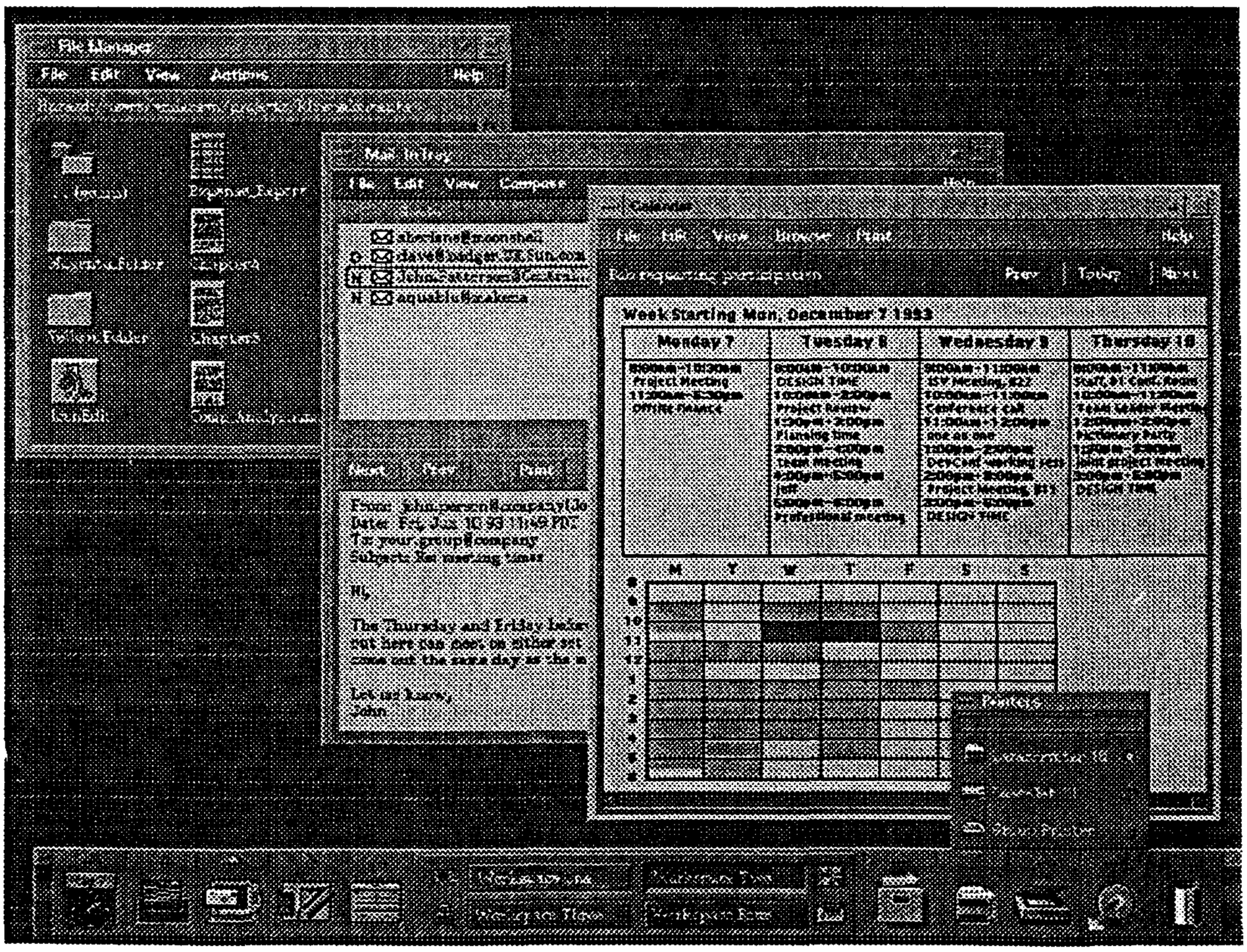
Windows 3.x 버전은 많은 응용 프로그램들을 가지고 있고, 버전이 낮은 DOS뿐만 아니라 새로운 Windows 프로그램도 가동시킬 수 있고, 확장 모드에서는 여러 개의 DOS응용 프로그램을 수행시킬 수도 있다. 또한 매크로와 작업 자동화 기능이 포함되어 있고, 가격이 비교적 싸다는 장점을 가지고 있다. 반면에 모든 특성을 최대한 이용하기 위해서는 많은 부수적인 하드웨어와 큰 기억용량을 필요로 한다.

세째는 UNIX를 기반으로 하는 GUI들로서 여기에는 OSF/Motif와 AT&T/Sun의 Open Look, Next의 NextStep등이 대표적이다. Motif와 Open Look은 다중작업 기능과 통신기능이 내장되어 있으나, 매크로와 작업 자동화 기능이 포함되어 있지 않고 화일 관리기가 없으며, 개발자를 위한 지침이 반드시 적용되어야 하는 것이 아니라서 응용 프로그램이 일정한 모양을 유지하지 않을 수 있는 가능성이 충분하다.

NextStep은 시스템이 하드웨어에 설치되어 배달되고, 응용 프로그램들은 안정적이며 서로 호환성을 가지고 있고, 프로세스 상호간의 통신을 잘 지원한다는 장점이 있는 반면, Next 컴퓨터와 IBM RS/6000에서만 이용가능하며, 응용 프로그램들이 많지 않고 작업 자동화 기능이 포함되어 있지 않다[2].

최근에는 COSE(Common Open Software Environment) 그룹이 GUI 통합에 중요한 그룹으로 부각되고 있는데, 이 그룹에는 IBM, Hewlett-Packard, Digital, Sun 등

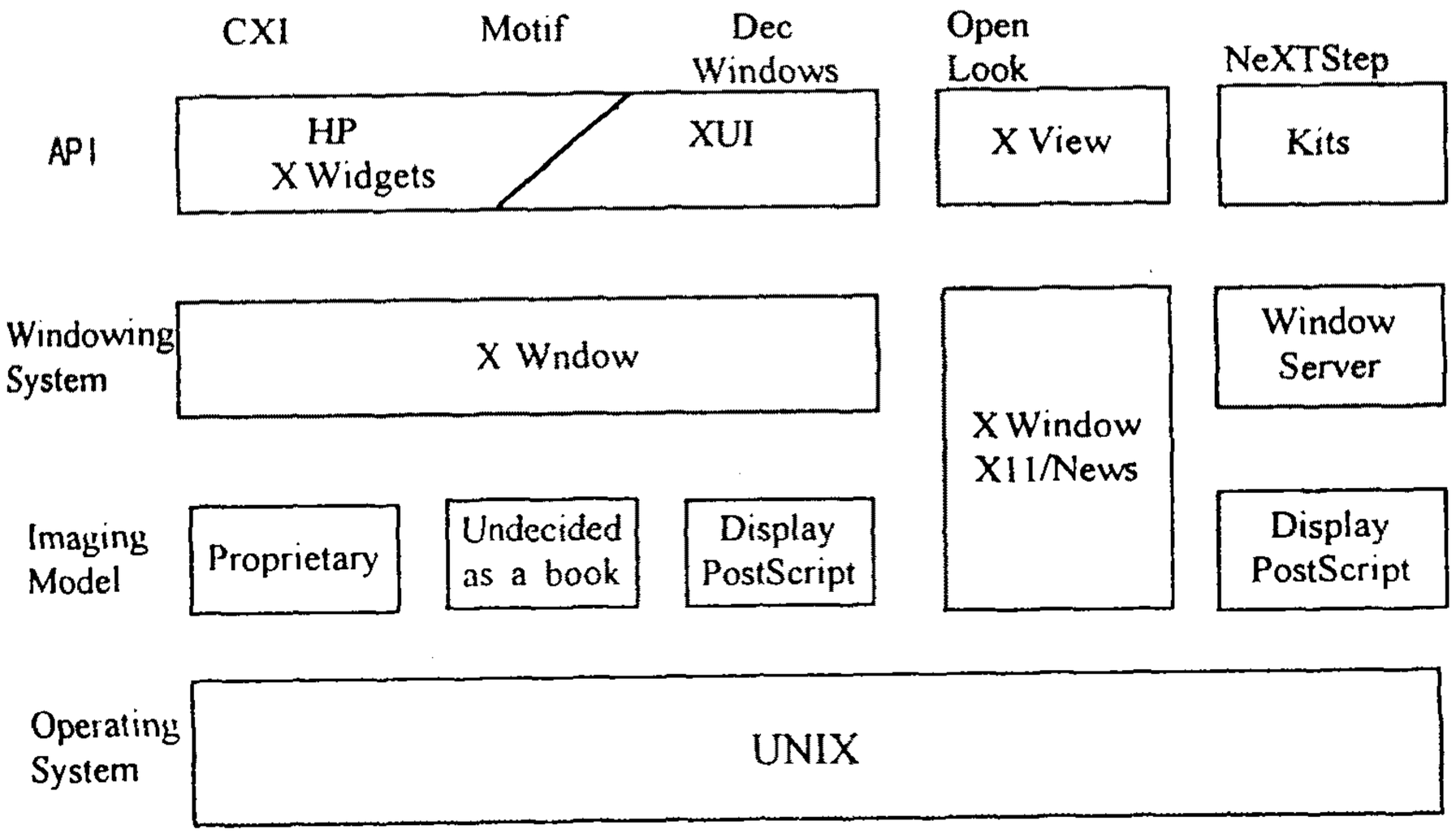
의 기업들이 참여하여 UNIX 플랫폼을 위한 통합된 GUI를 개발하고 있다. COSE 그룹이 개발하고 있는 CDE는 Common Desktop Environment라는 이름의 데스크탑 GUI로서 전혀 새로운 시스템을 표방하기 보다는 UNIX 공동체를 통합하고자 하는 의도가 더 크다고 보여진다. 즉 기존의 GUI 들로부터 최상의 특성(feature)들을 모아서 더 나은 시스템을 만드는 것이 이 그룹의 전략이다[4]. 예를 들어 CDE의 대부분은 OSF/Motif로 구성하고 있고, H/P 의 Visual User Environment(VUE), IBM의 Common User Access(CUA)와 Sun의 Open Look 의 특성들을 고루 반영하고 있다. [그림 2-3-1]은 COSE CDE 개발환경이다[5].



[그림 2-3-1] COSE CDE 개발환경

한편 GUI 시스템이 갖추어야 할 환경이 있는데 윈도우 시스템, 영상모형(Imaging Model), 응용 프로그래밍 인터페이스(API), GUI 개발지원도구(UIMS)들이다.

윈도우 시스템은 동시에 여러 응용 프로그램들을 보여줄 수 있도록 해주는 시스템으로써 프로그램 작업을 보다 쉽도록 도와주는 도구, 즉, 움직일 수 있고 모양 조절이 가능한 윈도우, 메뉴, 다이알로그 박스 등을 만들 수 있는 도구들을 포함하고 있다.



[그림 2-3-2] GUI의 구성

영상모형은 화면에 글자의 형태나 색상, 그래픽스 등을 어떻게 표현할 것인가에 대하여 정의한 것으로써, 이 영상모형으로 인하여 글자의 모양과 크기, 선의 종류와 색상 등의 처리가 가능하다. 응용 프로그램 인터페이스는 프로그램 언어의 함수의 집합들로서 프로그래머가 응용 프로그램을 보다 쉽게 구현할 수 있도록 지원해 준다.

GUI 개발 지원도구는 응용 프로그램과 사용자 인터페이스를 분리하여 개발.관리할 수 있도록 지원해주는 환경이다. 본 연구소가 수행한 과제는 이와 같은 GUI 개발지원도구의 프로토타잎을 개발한 것이다..

[그림 2-3-2] 는 각 GUI의 구조를 보여준다[6].

제 4 절 GUI 도구유형

1. GUI 도구 유형

최근까지 발표된 GUI 도구들을 보면 두가지 유형으로 분류된다. 한가지 유형은 사용자와 컴퓨터 간의 대화 기법들을 모아 라이브러리로 구성한 사용자 인터페이스 틀킷(Toolkits) 들로 대표적인 것으로서 X 컨소시엄의 X Windows(X11)를 들 수 있고, 사용자 편리성을 조금더 향상시켜준 OSF/Motif와 Sun의 OpenLook 등이 이 유형에 속한다. 라이브러리화된 틀킷을 이용한 사용자 인터페이스 코드들은 응용 프로그램과 밀접하게 연결되어 있어 다른 응용 프로그램에 같은 유형의 사용자 인터페이스를 재사용 함에 어려움이 있다[7].

다른 한가지 유형은 사용자 인터페이스 개발 시스템으로서 프로그래머가 여러 모양의 인터페이스를 만들고 관리하기 위하여 필요한 도구들을 통합한 시스템으

로서, 이들을 UIMS(User Interface Management System) 또는 UIDS(User Interface Development System)이라 부른다[8]. 이 시스템들은 라이브러리나 사용자 인터페이스 언어로 코딩하지 않고 화면을 보면서 사용자 인터페이스를 디자인 할 수 있도록 여러가지 기능들을 제공한다.

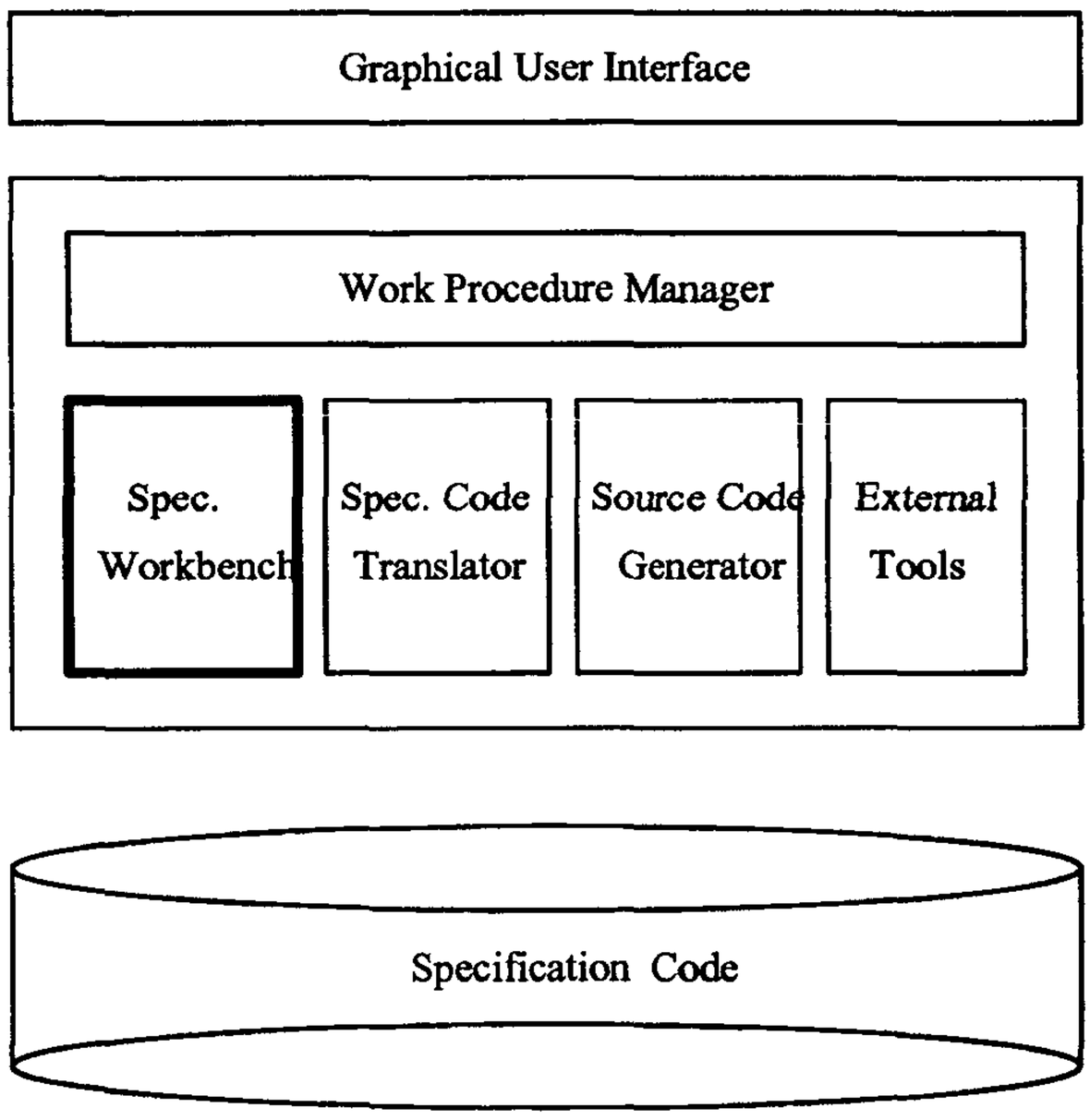
이들 사용자 인터페이스 도구들이 지원하는 기능들을 나누어 보면 세가지로 볼 수 있다. 첫째, 사용자가 대화식으로 시각적 환경에서 작업할 수 있는 명세작업대를 갖추고 있다. 둘째, 사용자의 작업내용을 표현하여 관리할 수 있는 높은 수준의 명세언어(Specification Language)를 가지고 있다. 세째, 명세언어로 표현된 코드를 최종의 컴파일 가능한 언어로 전환해 주는 코드생성기를 제공한다.

사용자 인터페이스 개발 자동화 도구는 소프트웨어 생명주기 전반에 걸쳐서 이용될 수 있는 중요한 개발 지원도구로서 인식되고 있으며, 응용 프로그램과 사용자 인터페이스 부분을 물리적, 논리적으로 구분하여 소프트웨어의 재사용 측면에서도 중요한 역할을 담당하고 있다.

2. GUI 도구 모형

GUI 도구는 사용자 인터페이스 영역에서 발생하는 순수한 객체들을 복잡한 프로그래밍 언어의 구문으로 부터 분리하여 개발자에게 제공하고 최종적으로는 컴파일 가능한 목적 원시코드를 생성하도록 구성되어 있다. [그림 2-4-1]은 GUI 도구들의 대표적인 구성을 보여준다.

<그림 2-4-1>에서 작업절차 관리기는 사용자의 작업절차를 조절하기 위한 통합적인 관리기능을 제공하며, 명세작업대는 다양한 위짓의 시각적인 합성을 지원하기 위한 그래픽 편집기 및 기타의 편집기 군으로 구성된다[9].



[그림 2-4-1] GUI 도구의 구성

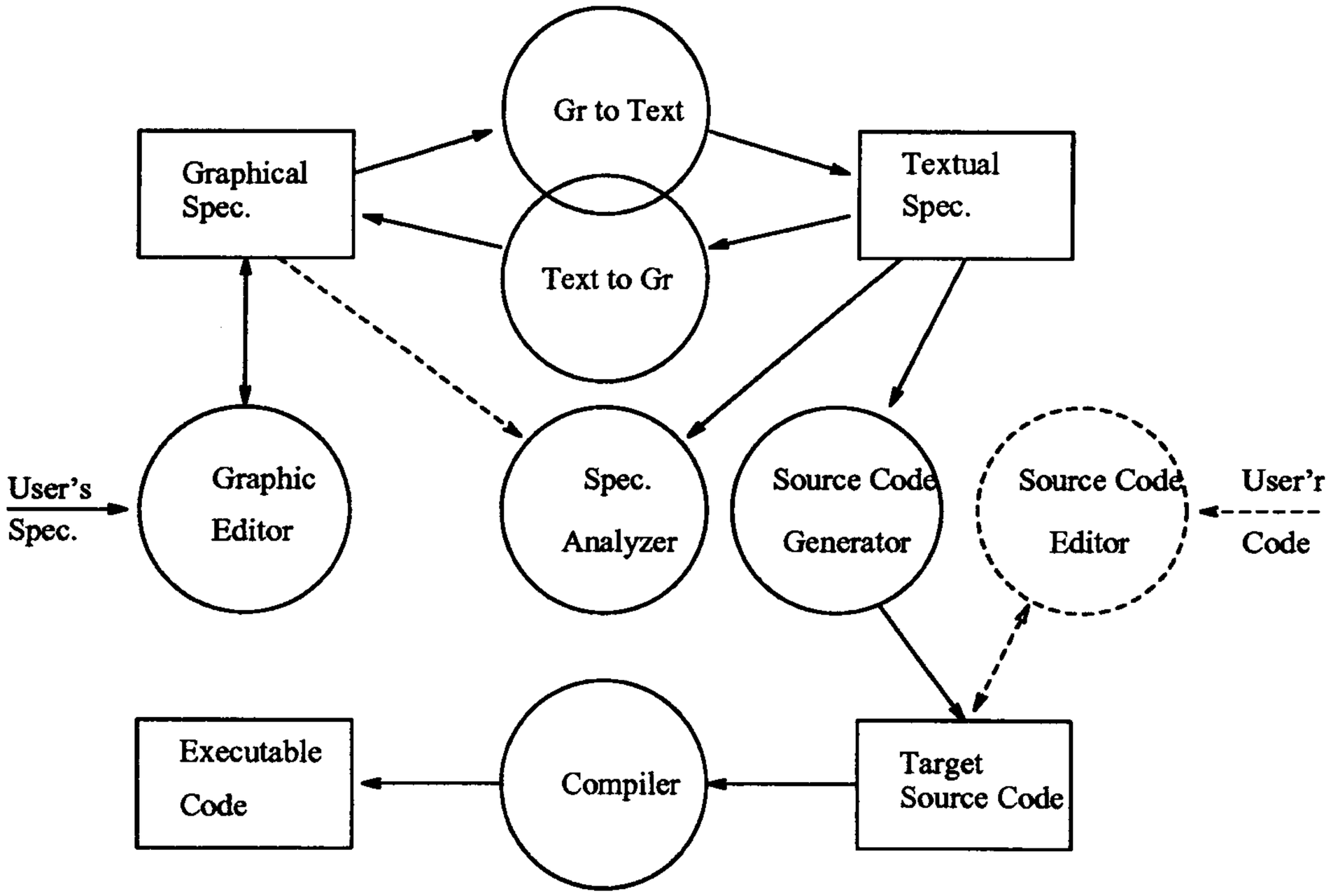
명세코드 변환기는 그래픽 명세를 텍스트 명세로 또는 텍스트 명세를 그래픽 명세로 변환해 주는 명세코드 생성기와 명세코드 번역기로 구성된다. 명세코드 생성기로 부터 만들어진 명세코드는 다시 원시코드 생성기를 거쳐 컴파일 가능한 원시코드로 생성되고, 컴파일과 라이브러리 링크를 거쳐 수행가능한 최종의 응용 프로그램을 얻게 된다.

이와 같은 구성은 스위칭 시스템 소프트웨어 개발 도구에서도 볼 수 있다.

개발자는 시스템에 대한 명세를 시각적인 환경에서 그림을 중심으로 합성 및 편집하여 그래픽 명세를 만들고, 이것을 변환기를 통하여 텍스트 명세로 변환하여 저장.관리한다. 이렇게 저장된 정보는 언제든지 다시 변환기를 통하여 그래픽

명세로 변환되어 변경을 반영할 수 있다. 텍스트 명세는 명세분석기를 통하여 그 완전성이 검증되며, 개발자가 원하는 프로그래밍 언어의 원시코드로 생성된다. 그런데 이 원시코드는 모든 기능들이 정의된 코드라기 보다는 구조적인 특성 및 제어적인 특성만을 표현하고 있는 것이 일반적이다.

이와 같은 작업절차를 갖는 도구류는 자동화를 기반으로 하고 있으므로 자동화 도구류로 분류한다[9].



[그림 2-4-2] 자동화 도구의 구성

3. GUI 도구 사례

가. UIM/X

UIM/X는 Visual Edge Software에서 개발한 GUI 개발도구로써 대상 기종은 Sun SPARC, DEC, HP, IBM등이며 OSF/Motif 툴킷을 통합하고 있으며 구성 및 기능은 다음과 같다.

1) WYSIWYG 에디터

각 위짓의 레이아웃을 실제로 확인하면서 작업을 진행하며 조작시 필요한 각 부품들을 드랙(drag) & 드롭(drop)하는 방식을 선택하고 있으며 위짓에 대한 이동, 크기 변경, 복사(copy)등의 작업이 가능하다.

2) 위짓 브라우저(Widget Browser)

레이아웃(layout)상의 각 위짓의 리소스(resource) 관계를 트리(tree) 형식으로 표시하며 각 위짓간의 계층 구조도를 설계 할 수 있으며 WYSIWYG 에디터와 연동하여 레이아웃상의 어떠한 위짓이 어디에 대응하는지 간단히 파악할 수 있게 한다.

3) 위짓 특성 편집기(Widget property Editor)

위짓의 속성을 정의하는 전용 에디터로써 각 위짓의 속성 변경, 복수의 위짓에 공통되는 속성을 모아서 변경 가능하며 틀린 문법 입력시 즉시 에러를 표시해 주기 때문에 문법적으로 정확한 입력이 가능하다. 또한 색의 속성 선정시 색상편집기(Color Editor)를 사용할 수 있다.

4) Color Viewer

기존에 정의되어 있는 색의 리스트를 표시하며 사용자가 원하는 색상이 없을

경우 색상 편집기를 사용하여 새로운 색을 만드는것도 가능하다.

5) 색상 편집기(Color Editor)

색의 다양한 요소를 조작하여 새로운 색상을 만드는 기능을 갖고 있는 편집기로써 사용자가 원하는 색상을 만들수 있다.

6) 위짓 팔레트(Widget Palette)

OSF/Motif에서 제공하는 다양한 위짓들을 표시하며 필요한 위짓을 선택하여 WYSIWYG 에디터상에 레이아웃을 작성하며 Customize 가능하기 때문에 새로이 작성된 위짓을 넣을 수 있고 개발의 내용에 맞추어 표시된 위짓을 변경시키는 것도 가능하다.

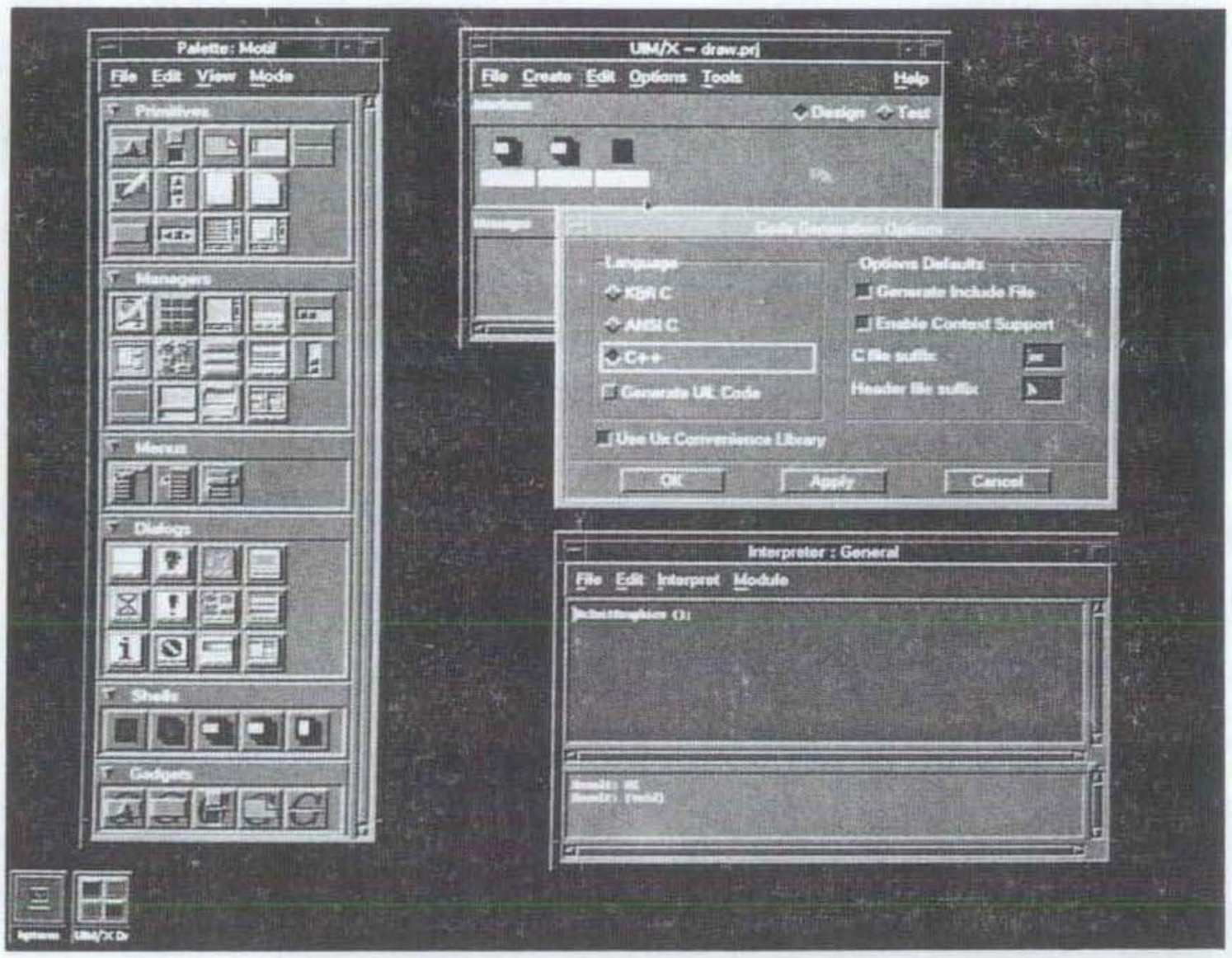
7) C 인터프리터(Interpreter)

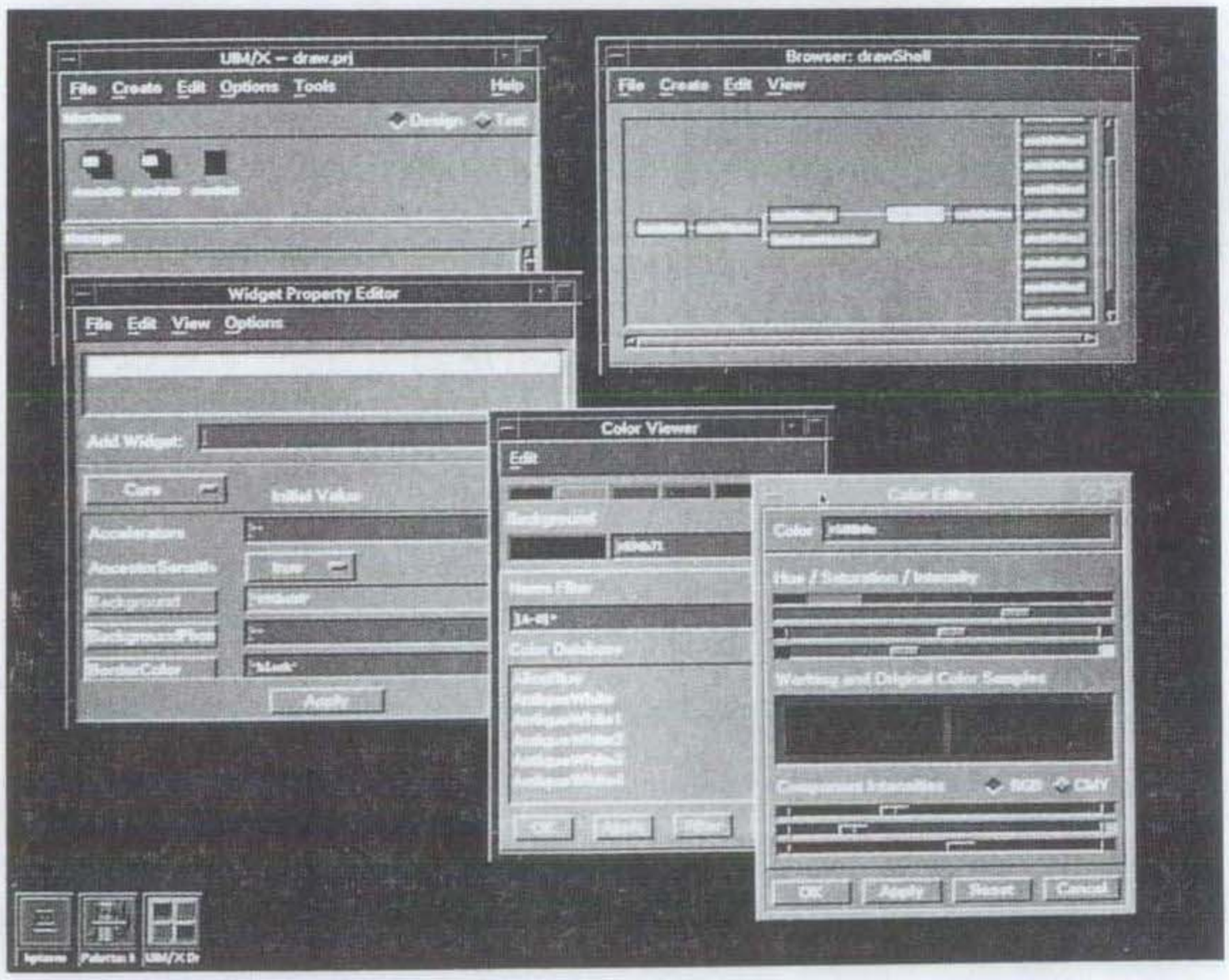
설계된 GUI 디자인의 동작을 즉시 확인 가능하며 디자인 변경시 재컴파일 등의 작업이 필요없으며 또한 실행중 에러가 검출될 경우 그 위치를 위짓이나 자원(resource)에 알려주는 기능을 갖추고 있다.

8) 소스코드 생성

K&R C, ANSI C, C++, UIL언어등 4종류의 코드를 생성할 수 있으며 이런 소스코드에는 주석(comment)도 자동생성하며 UIM/X 에서 제공한 Ux 라이브러리를 사용하여 소스코드를 생성한다.

UIM/X의 특징으로는 프로그래머가 "COMPONENT"라 불리우는 재사용 가능한 GUI 오브젝트 부분에서 인터페이스를 구성할 수 있게 해주며 부품(component)은 캡슐화, 상속(Inheritance), 동질이형(Polymorphism)의 특성을 갖고 있기 때문에 객체지향 프로그램의 장점을 활용할 수 있다. [그림 2-4-3]은 이상에서 설명한 UIM/X 도구들에 대한 화면이다.





[그림 2-4-3] UIM/X 구성화면

나. BUILDER XCESSORY

BUILDER XCESSORY는 ICS(Intergrated Computer Solutions Incorporated) 에서 개발한 Xlib/Xt/Motif Toolkit을 이용하여 C, C++, UIL, Ada등의소스코드를 생성해주는 GUI도구이며 대상 기종은 DECstation 5000, HP 9000/700,800, Sun4등이며 구성 및 기능은 다음과 같다.

1) 팔레트(Palette)

팔레트는 OSF/Motif 위짓으로 구성되어 있으며 사용자는 필요한 위짓들을 마우스로 클릭하여 원하는 GUI를 구성할 수 있으며 ICS 위짓 데이타북(Databook)에 있는 위짓이나 사용자가 정의한 위짓을 팔레트에 추가하여 사용할 수 있다. 또한 사용자가 객체지향방법으로 정의한 클래스 혹은 오브젝트를 C++ 인터페이스 형태로 팔레트에 추가할 수 있다.

2) 브라우저(Browser)

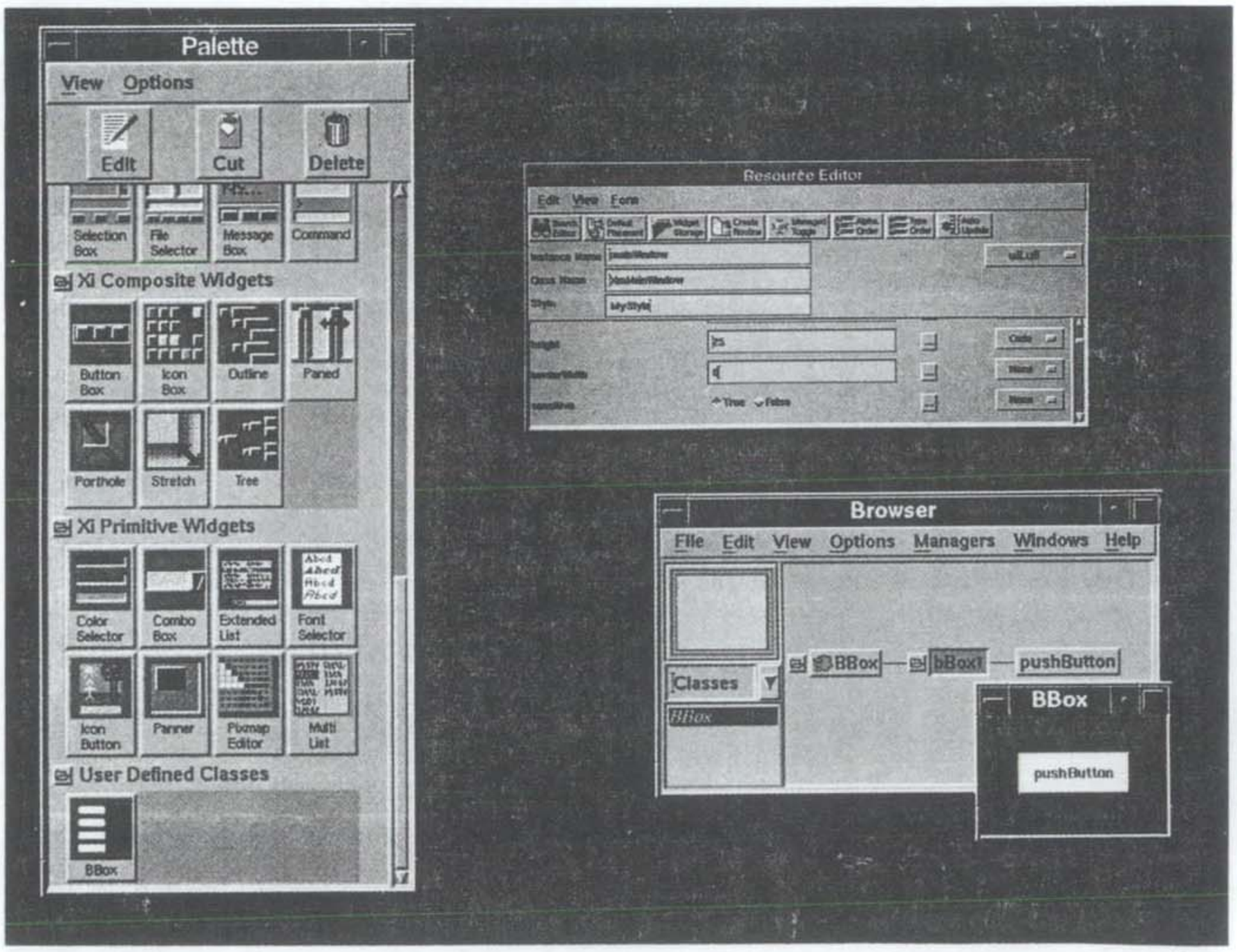
브라우저 윈도우는 사용자가 선택하여 구성한 GUI를 Motif의 위짓 계층구조 혹은 Scroll 리스트 구조로 나타내 준다. GUI 구성 설계가 완료된 후 브라우저를 이용하여 곧바로 GUI 소스 프로그램을 생성할 수 있는데 이때 생성되는 소스코드는 makefile과 C, C++, UIL, Ada등의 프로그램이다.

3) 자원 편집기(Resource Editor)

GUI를 구성하는 위짓의 속성을 편집 수정하는 기능을 가지며 하나의 위짓이 가지는 자원을 수정하게 되면 그즉시 GUI 프로그램이 영향을 받아 화면상에 다시 표시된다. 자원 편집기에서는 OSF/Motif에서 제공되는 모든 입력 형태를 지원하며 픽셀 편집기, 색상 선택기, 폰트 선택기, 콜백(Callback) 리스트 편집기, 화일 관리자등을 제공한다.

BUILDER XCESSORY의 특징으로는 객체지향 방법에 의해 재사용 가능한 코드를

생성하여 주며 코드의 캡슐화와 단위성, 독립성을 유지시켜주어 코드의 관리가 용이하다. [그림 2-4-4]는 Builder Xcessory의 구성 화면이다.



[그림 2-4-4] Builder Xcessory 구성 화면

다. ObjectBuilder

ObjectBuilder는 ParcPlace사에서 개발한 GUI도구로써 대상기종은 Sun SPARC, HP 9000/700, IBM RS6000등이며 구성 및 기능은 다음과 같다.

1) 속성 편집기(Attribute Editor)

콜백(Callback), 색상(color), 상태(positioning), 번역(translation), 기본(default) 편집기를 포함하고 있으며 색상, 폰트, 행위(action) 기타 인터페이스 속성들을 작성하는 기능을 갖는다. 이와는 별도로 새롭게 정의된 오브젝트용의 속성 편집기나 개발그룹 내에서 표준으로 준비한 독자적인 속성 편집기도 사용 가능하며 시스템에서 준비한 기본 편집기에서도 특수한것을 제외하고는 변경한 전체 오브젝트를 처리할 수 있다.

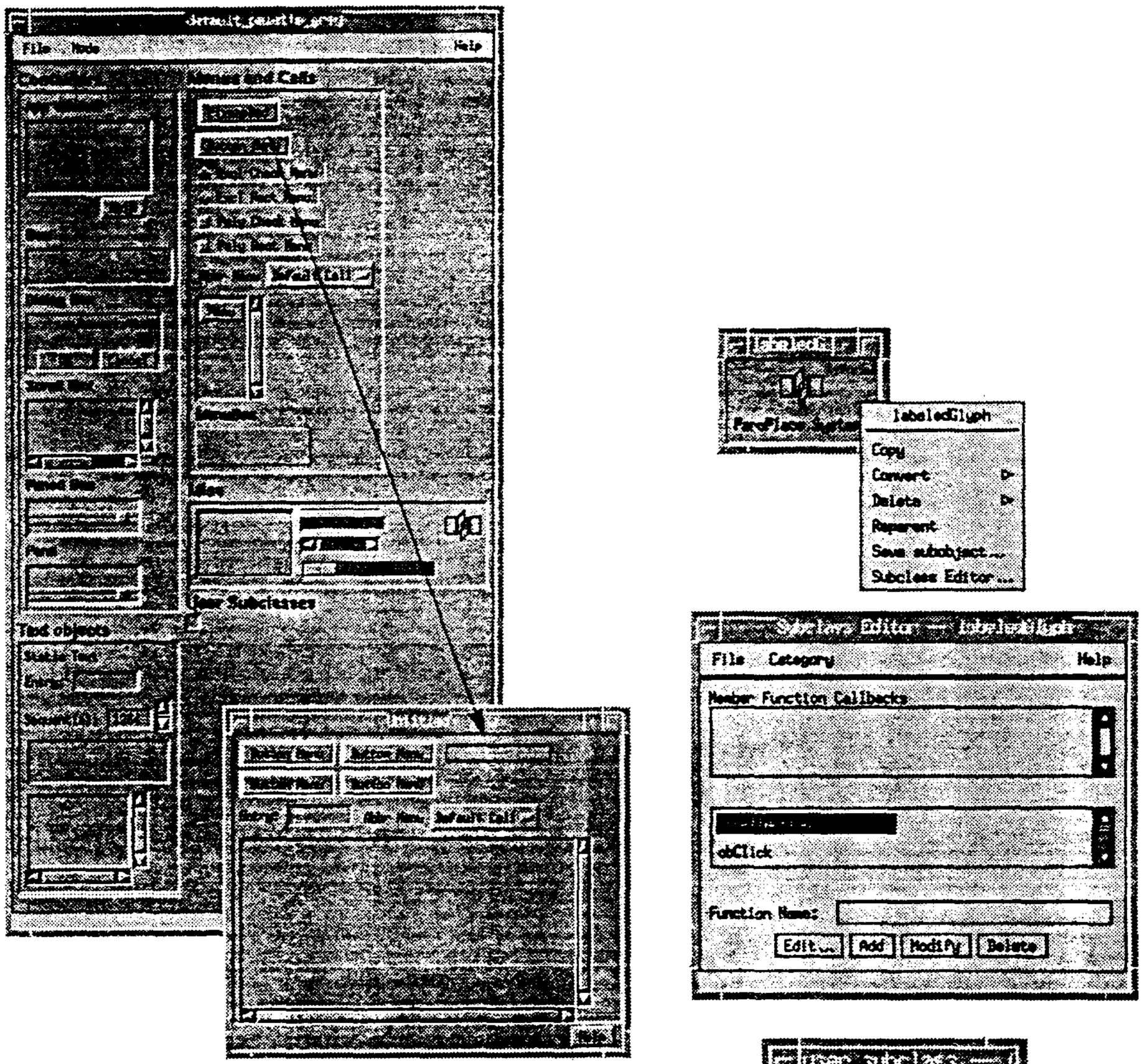
2) Custom User Interface

일반적인 어플리케이션의 보안 단계(security level)나 여러종류의 최종 사용자로부터 습득된 다양한 어플리케이션에 접근할 수 있으며 사용자의 형(type)에 따라 다양한 인터페이스를 준비한다.

3) 오브젝트 인터페이스(Object Interface)

Motif, Open Look 표준의 Super set을 C++ 클래스 계층으로 실현한 클래스 라이브러리로써 C++로 작성된 어플리케이션 코드와 일치하기 때문에 코드의 통합이 가능하다.

ObjectBuilder의 특징으로는 Visual subclassing의 기능을 이용하여 사용자 정의 인터페이스를 간단히 작성할 수 있으며 기존 부품으로부터 오브젝트를 drag and drop하여 새로운 서브클래스를 작성할 수 있으며 새로이 작성된 서브클래스는 파레트에서 자동으로 등록되며 그 즉시 재이용 가능하다. 또한 Visual subclassing은 오브젝트의 계승을 지원하고 오브젝트의 속성을 변경 하였을 경우 그 오브젝트를 베이스 오브젝트로 하는 전체의 오브젝트까지 반영 된다. [그림 2-4-5]는 ObjectBuilder의 구성 화면이다.



[그림 2-4-5] ObjectBuilder 구성 화면

라. X-Designer

X-Designer는 미국의 V.I에서 개발한 Motif 1.2 어플리케이션을 위한 그래픽 인터페이스 구축(Graphical interface building)도구로써 대상 기종은 Sun SPARC, HP 700, DECstation등이며 구성 및 기능은 다음과 같다.

1) 메인 윈도우(Main Window)

메인 윈도우는 위짓 선택(widget selection) 팔레트와 인터페이스 구축 영역(construction area)으로 구성되며 구축영역은 디자인의 상하 구조를 나타내주는 트리 뷰(tree view)를 포함하고 있다. 이 트리에 있는 위짓을 마우스를 사용하여 Cut/Copy/Paste등의 조작을 통하여 위짓의 재구성 혹은 새로운 부모(parent)로의 변경이 가능하며 이러한 디자인 구조는 PostScrip 화일 혹은 프린터로 프린트할 수 있다.

2) WYSIWYG 다이나믹 디스플레이 윈도우(Dynamic Display Window)

트리상의 위짓을 조작하여 인터페이스의 변화를 즉시 보여주는 기능을 가지고 있으며 자신이 작성한 인터페이스를 언제라도 테스트 혹은 작동시켜 볼수 있다.

3) 레이아웃 에디터(Built-in Layout Editor)

레이아웃 에디터는 Form 위짓과 같은 Motif 레이아웃 위짓의 set attachment를 구현하는 인터엑티브 도구이다. 위짓이나 attachment의 이동은 마우스의 이동으로 가능하며 자동적으로 정렬(align) 시키는 기능과 여러단계의 undo기능을 지원하여 작업의 단계를 역으로 추적할 수 있다.

4) 각종의 에디터

X-Designer에서는 XPM 형태 혹은 X Bitmap형태의 color pixmap을 작성 저장할 수 있는 Color pixmap 에디터와 여러 종류의 폰트 및 특정언어를 위해 지원되는 Compound String 에디터 및 폰트의 형태와 크기등을 선택할 수 있는 폰트 에디터를 포함하고 있다.

5) 소스코드 생성

X-Designer는 표준의 K&R, ANSI C 코드 혹은 C++나 UIL 코드 형태의 소스코드를 생성해 준다.

X-Designer의 특성으로는 온라인 도움말(On-line help)을 제공하며 부적절한 옵션에 의한 에러를 예방하는 기능을 갖추고 있어 올바른 위짓 체계(hierarchy)를 만들어 주며 생성된 소스코드의 품질을 보증해 준다.

마. TeleUSE

TeleUSE는 미국의 Alsys사에서 개발한 GUI 어플리케이션의 개발, 보수를 지원하는 UIMS로써 대상기종은 Sun SPARC, HP 9000/400/700, IBM RS/6000, DEC Station 등이며 GUI 어플리케이션을 프리젠테이션(Presentation) 계층, 다이아로그(Dialog) 계층, 어플리케이션 계층의 3단계로 계층화하였다. 그 구성은 TeleUSE 전체의 기동 또는 Third party 툴을 기동시켜주는 세션 관리기(session manager), 위짓의 계층구조를 표시하며 이 위짓들을 복사(copy), 절단(cut), 붙임(paste) 등의 편집이 가능한 트리 에디터(Tree editor), 정적인 프리젠테이션층과 동적인 다이아로그층간의 관계를 테스트하는 테스트 모드(Test mode), 복수의 위짓으로 구성되며 위짓의 계층에 대한 정보가 포함되어있는 템플레이트(template)를 사용하여 GUI를 개발, 유지보수하는 객체지향 개발 도구인 VIP, D script를 인터프리터 형태로 생성할때 인터액티브(interactive)하게 디버그할 수 있는 다이아로그 편집기 및 디버거로 되어 있다.

TeleUSE의 특징으로는 자신이 작성한 위짓 혹은 third party에서 작성한 위짓을 TeleUSE에 추가하는 것이 가능하며 Run time 라이브러리 소스코드를 지원하며, 70여 종류의 GUI 샘플을 소스코드로 제공하여 준다.

제 5 절 GUI 기술의 발전 방향

GUI 기술은 인간의 시각성을 소프트웨어의 특성에 반영해 보자는 사조로 부터 시작되었으며, 시각적인 자극은 인간의 사물 또는 문제에 대한 인식의 속도를 향상 시키는 특징을 가지고 있는 것으로 알려져 있고, 또한 경험적으로 일상 생활에서 이용되고 있다. 소프트웨어 개발 관점에서 GUI 기술은 시각프로그래밍 기술과 밀접한 관계를 가지고 있다. 따라서 시각 프로그래밍과 GUI 개발 기술을 통합하여 함께 전망한다.

시각 프로그래밍 기술은 그림과 아이콘을 이용하여 프로그래밍을 할 수 있으며, 이러한 방법은 프로그래밍을 훨씬 쉽게한다는 경험과 믿음을 바탕으로하는 기술이다.

소프트웨어 개발 기술에 있어서 앞서가고 있는 국외의 여러 기관에서의 연구 및 개발은 일정한 응용영역에서 실용성이 있는 도구의 개발이 지속적으로 이루어지고있다. 이러한 개발은 대략 세가지 범주, 언어 front-end 도구 군, 데이타베이스의 front-end 도구 군, 그리고 GUI front-end 도구 군의 유형으로 이루어 지고 있으며, 기존 CASE 기술과의 연계방법을 모색하고 있다. 이와함께 순수한 의미의, 즉 프로그래밍 언어의 관점에서의 시각 프로그래밍 언어에 대한 연구도 학계를 중심으로 개념의 형성을 위한 연구가 활발하다.

국내에서도 이와 유사한 유형의 연구 및 개발이 이루어지고 있다. 언어의 시각화에 대한 연구는 학계를 중심으로 이루어 지고 있고, GUI front-end 도구의 개발은 본 연구소에서 집중적으로 이루어지고 있으며, 데이타베이스 front-end 도구의 개발은 DBMS를 개발하고 있는 산업계에서 주로 이루어지고 있다.

외국의 경우를 살펴보면, 실용 가능한 시각 프로그래밍 도구들이 상품으로 나와 있으며, 보다 나은 기능을 제공하고 보다 다양한 응용범위를 확보하기 위한 연구가 진행중이다. 한편으로는 기존의 CASE 기술과 통합하기 방법을 모색 하고 있다.

이러한 도구들은 대략 세가지 유형으로 분류할 수 있다. 첫째는 언어 자체를 시각화하여 환경을 전체적으로 시각화한 도구 군으로서, Prograh International Inc.의 Prograph, Novell 사의 AppBuilder등이 대표적인 사례이다. 두번째 도구군은 DB의 graphical front-end 유형으로 Power-soft 사의 PowerBuilder를 들 수 있다. 세번째 도구 유형은 GUI front-end를 가진 언어로서, Microsoft 사의 Visual Basic, Visual C++ 등이 대표적인 예이다. 한편, 기존의 CASE 기술과 연계할 수 있는 방법에 대한 연구도 CASE 기술에 주력해온 큰 기업들에서 활발히 진행 중이다.

국내의 경우를 살펴보면, 국내에서의 연구 개발 사례도 외국의 사례 분류와 같이 나눌 수 있다. 그러나 국내에서의 시각 프로그래밍 연구 개발 사례는 매우 제한되어 있다. 대학을 중심으로 언어적인 관점에서 시각 언어에 대한 연구와 기존의 언어를 시각화하려는 연구가 이루어 지고있다. S/W 전문 연구소에서는 GUI를 중심으로 실용 도구의 개발에 주력하고 있으며, 산업계 에서는 DB의 front-end 도구로서, 상용 도구의 개발에 주력하여 결과물을 선보이고 있는 상태이다. 그러나 국내의 기술 수준은 선진 외국과 3-5년 정도의 격차를 보이고 있다.

이러한 연구개발 사례를 분석하면, 시각 프로그래밍 기술 분야 중에서 아직 실용적인 단계에 미치지 못하고 있는 일반적인 문제점은 다음과 같다.

- 느린 프로그램의 생성
- 사용자가 프로그램을 최적화 시킬 수 있는 방법이 부족

- 선형적인 기술
- 한정된 응용영역
- 디버깅 기능 취약

기술을 세부적으로 분석해 보면, 언어 VP 기술에 있어서 외국의 경우에는 VP의 범용성을 추구하여 일부의 상용화된 제품을 내고 있으나, 앞에서 기술한 기본적인 문제점을 실용성 있게 해결하였다기 보다는 기본적인 기술의 개발이 이루어진 상태로 볼 수 있다. 또한 응용의 품질은 해당분야 지식의 정련성에 의존적이므로, 많은 잘 정의된 응용 라이브러리를 필요로 한다. 이러한 관점에서 지식의 시각적 표현이 문제점으로 대두된다. 또한 기존의 CASE 기술 및 도구와의 연계가 원할치 못한 문제점을 가지고 있다.

국내의 경우에 언어 VP 기술을 보면, 이 기술의 개발은 주로 대학을 중심으로 학문적인 수준 에서 이루어지고 있다. 연구는 기술적으로 매우 기초기술적인 측면으로 볼 수 있는 시각 요소의 정의, 구문분석 기술, 의미해석 기술 분야에서 이루어지고 있으며, 부분적으로 객체지향 언어 및 질의어 유형의 시각화에 촛점이 주어지고 있다.

GUI VP 기술에 있어서 외국의 경우를 살펴보면, 세계적인 GUI의 표준화에 힘입어 훌륭한 상용 제품의 개발에 이 기술이 적극적으로 활요되고 있으며, 그 제품도 상당량 시장에 나와 있다. 이 기술은 크라이언트 서버형 시스템 배치의 일반화 추세에 마추어 클라이언트 소프트웨어 개발에 필수 기술로서 이용되고 있다. 그러나 이 기술 역시 자동 코드 생성으로 부터 나타나는 일반적인 문제점을 실용적으로 해결하지는 못하고 있어서 이 분야의 기술 개발이 꾸준히 진행되고 있다.

국내의 경우에 GUI VP 기술을 보면, 이 기술에 대한 국내의 연구는 국내시장의 응용 소프트웨어의 시각성이 사용자들로 부터 필수적인 요구사항으로 대두되

고 있지 못하고 있어 기술 개발의 속도가 선진 외국의 경우에 비추어 더디게 이루어지고 있다. 그럼에도 불구하고 미래의 시장 추세를 감안하여 산업계와 연구계에서 실용 기술 및 제품의 개발이 심도있게 이루어지고 있다. 국내의 기술 수준은 우선 외국의 최신 기술을 참조하여 국내의 기술을 개발하고 있는 상태로 평가된다.

DB VP 기술 분야에서 외국의 기술 수준을 보면, 대부분의 DBMS 회사에서 이 기술은 필수적인 기술로 인식하고 그들의 상용제품에 반영하고 있으며, 언어 회사에서도 언어의 범용성을 확보하기 위하여 이 기술의 개발에 많은 투자를 하고있다. 한편, 응용 소프트웨어 전문 개발사에서도 자기들의 목적에 맞게 이 기술을 개발하여 응용하고있다. 이 기술과 관련된 제품들은 시장에 나와 경합을 벌이고 있는 상태로, 일본에서도 제품을 올해 초에 시장에 내 놓았다.

국내의 기술 수준을 가늠해 보면, 이 기술에 대한 국내의 개발도 DBMS 개발회사에서 주로 이루어지고 있으며, 작은 소프트웨어 업체에서도 동시에 이루어지고 있으며 공식적으로 세개의 국내 제품이 시장에 나와 있다. 이 기술은 자료처리를 중심으로 한 국내의 응용 소프트웨어 시장에 쉽게 적응할 수 있는 기술로 연구개발이 지속적으로 진행될 것으로 예상되고 있다.

시각프로그래밍 기술의 일반적인 문제점으로 알려지고 있는 것은 다음과 같다.

- 자동 생성된 프로그램의 수행은 일반적으로 느림
- 자동 생성된 프로그램을 사용자가 최적화 시킬수 있는 방법이 미약
- 시각 프로그래밍 기술의 선형성
- 시장에 나와 있는 시각 프로그래밍 도구를 적용할 수 있는 응용의 범위는 제한적
- 디버깅 할 수 있는 기능이 미약

앞으로는 이러한 문제점을 해결하기 위한 기술 개발이 이루어 질 것이며, 기존의 CASE 기술과 연계할 수 있는 방법에 대한 연구가 진행될 것이다. 한편 상용 도구의 유형은 언어 front-end, 데이타베이스 front-end, GUI front-end 유형의 도구를 통합한 환경이 주종을 이룰 것이며, 순수한 의미의 범용적인 시각 언어에 대한 연구가 보다 가속될 것이다.

제 3 장 GUI Mosaic 분석 및 설계

여 백

제 3 장 GUI Mosaic 분석 및 설계

제 1 절 개념 및 기술

1. 개 념

GUI는 하드웨어 컴퓨팅 시스템의 발전에 힘입어 소프트웨어 전반에 걸친 필수적인 요소로 자리하고 있으며, 관련 표준화 활동 또한 활발하다. 다양한 스타일의 GUI가 출현하였으나 이제는 대략 우세한 것들의 윤곽이 드러나고 있다. UNIX 계열에서는 OSF의 Motif, PC의 MS-DOS계열에서는 Microsoft의 MS-Windows와 IBM의 OS2의 표준 GUI인 Presentation Manager등이 시장에서 그 우세성을 인정받고 있다.

이러한 GUI 스타일의 정리는 개발지원도구의 성격을 갖는 UIDS(User Interface Development System)의 개발을 유도하였다. UIDS는 성격상 GUI 개발의 전방 도구의 역할을 하며, 따라서 해당 GUI에 의존적인 것이 일반적이다. 대표적인 시스템으로 UIM/X를 들 수 있으며, Open Interface와 같은 도구의 경우는 몇 가지 대표적인 GUI 스타일에 적용할 수 있도록 개발되었다. 어떤 것이 유용할 것인가는 개발자의 상황에 따라 다를 수 있다. 즉, 일반성 대 특수성의 유효성의 판단은 상황에 따라 바뀔 수 있다. 본 연구에서 개발하고있는 GUI Mosaic은 OSF/Motif 스타일의 GUI 개발을 지원하는 UIDS의 범주에 속한다.

GUI 도구는 다음과 같은 몇가지 필수적으로 갖추어야 하는 특성들이 있다.

. 사용자의 작업환경은 시각적이며, 대화식이어야 한다.

. 작업의 결과는 즉시 검증되어 반영되어야 한다.

. 사용자에 의한 프로그래밍 언어의 사용은 가능한 배제되어야 한다.

. GUI는 프로그래밍 언어 코드가 아닌 명세 코드 수준에서 관리되어야 한다.

. 응용 프로그램과의 연결 방법이 정의되어야 한다.

. 목적 프로그래밍 언어 코드를 생성할 수 있어야 한다.

. 생성된 코드는 바로 컴파일하여 수행될 수 있어야 한다.

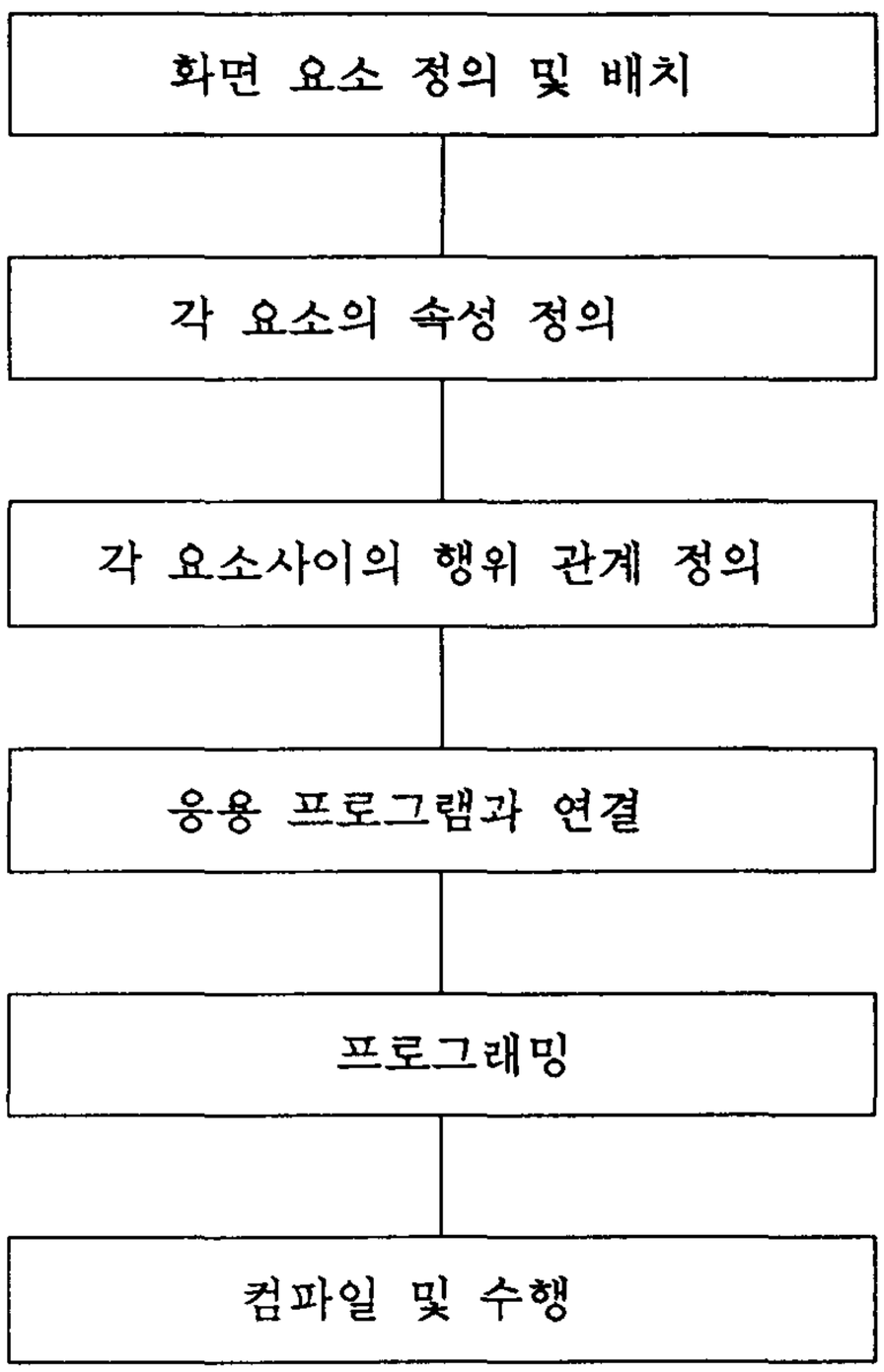
대부분의 GUI 도구들이 이러한 특성을 만족하고 있으나, 응용 프로그램과의 연결 특성을 만족스럽게 지원하고 있지는 못하다. 이부분은 기존의 CASE 도구와의 통합을 통하여 해결하려는 경향이 지배적이다. GUI Mosaic도 이러한 특성들이 만족될 수 있도록 개발되고 있으며, 프로토타잎에서 이미 가능성 검증은 마친 상태이다.

GUI Mosaic이 배경으로 하는 Motif는 소프트웨어를 GUI부와 응용부로 나누어서 GUI부를 모형화 한 점에서 가장 기본적인 영역분석 관점을 볼 수 있다. GUI부는 한정된 몇가지 방법을 통하여 응용부와 연결될 수 있도록 모형화 되었다.

GUI 개발 작업은 대체로 [그림 3-1-1]과 같은 과정들로 이루어진다. [그림 3-1-1]에서 화면 요소의 정의 및 배치 과정에서는 Motif 위짓의 합성과 배치를 통하여 사용자가 원하는 화면의 틀을 편집하는 과정이며, 이러한 배치가 끝나면 각 화면 요소의 내적 속성을 편집하는 과정이며, 다음으로 각요소의 외적 속성, 즉 다른 요소에 영향을 미치는 속성과 행위를 정의하는 과정이 이루어지며, GUI와 응용 프로그램을 연결하는 과정이 이어지고, 앞의 과정들에서 정의한 내용을 프로그래밍하는 과정이 있고, 마지막으로 코드를 컴파일하여 검증하는 과정이 이어진다. 이러한 일련의 과정을 마치게 되면 사용자가 개발하고 있는 소프트웨어의 GUI부 개발이 끝나게 된다. GUI 도구는 이러한 과정을 지원하도록 구성되어

있다.

GUI Mosaic은 위에서 열거한 과정들을 자동화의 개념을 바탕으로 여러가지 기능들을 적절히 배분하여 체계화한 도구이다. 사용자는 최종적으로는 코드생성을 통하여 코드와 컴파일 가능한 환경을 얻게되고, 이어서 컴파일을 수행하면 원하는 GUI를 얻도록 구성되었다.



[그림 3-1-1] 일반적인 GUI 개발 과정

2. 기 술

GUI 도구와 관련된 기술들은 대부분 자동화에 필요한 기술들에 속하며, 도구

의 대상 영역에 해당하는 GUI 기술이 주요 기술들이다. 자동화 기술은 재사용 기술과 명세 정형화 기술을 바탕으로 하며, 시각 프로그래밍 기술이 관련되어 있다. 또한 컴파일러 기술에 해당하는 번역 기술이 명세와 목적 원시 코드 사이의 교량 역할을 한다. 그외에 기본적인 기술로는 그래픽을 들 수 있다.

재사용 기술은 협소한 의미의 기술적 재사용이 적용된다. 재사용 기술은 일반적으로 기술적인 측면과 비 기술적인 측면으로 나눌 수 있다. 후자의 경우는 재사용의 하부구조로서 주로 조직적인 차원에서 재사용의 활성화를 위한 방법들과 관련되는 반면에, 전자는 개발과 직접적으로 관계를 갖는 기술들을 말한다. 자동화와 관련된 재사용 기술은 생성 기술류라고 할 수 있다. 생성 기술은 다시 변환 기술과 합성 기술로 나누어 생각할 수 있다.[1]

변환 기술은 한 모형에서 다른 모형으로의 천이를 의미한다. 소프트웨어 개발 과정의 앞뒤 과정을 고려하면 변환은 다시 앞 과정의 명세에서 뒷 과정의 명세로 변환할 경우를 수직 순 변환이라고 하며, 그 반대의 경우는 수직 역 변환이라고 한다. 예를들면 설계 명세로 부터 구현 언어로 된 코드로의 변환이 수직 순 변환이며 그 반대의 경우는 수직 역 변환에 해당한다. 한편 같은 수준에서 모형을 변환할 경우를 수평 변환이라고 하며, C 프로그램을 C++ 프로그램으로 바꾸거나 그 반대로 바꿀 경우가 이러한 변환에 해당한다. 자동화 도구류에서는 주로 수직 순 변환 기술과 수평 변환 기술이 이용된다. 정형화된 명세 코드로 부터 원시코드를 만들어 내는 기술은 수직 순 변환 기술에 해당하며, 명세코드를 그래픽 정보로 바꾸거나 그 반대로 할 경우는 같은 수준에서 일어나는 모형의 변환으로 이해할 수 있으므로 수평 변환에 해당한다고 할 수 있다.

합성 기술은 미리 준비된 여러가지 라이브러리 요소들을 모아서 목적하는 소프트웨어 요소를 만들어 가는 기술을 의미한다. 합성 기술의 대상이 되는 라이브러리 요소를 모듈이라고 하며, 완성형 모듈과 수정형 모듈로 구분할 수 있다. 완

성형 모듈이란 한번 만들어 진 뒤에는 변경을 가하지 않는 모듈이다. 반면에 수정형 모듈이란 만들어질 때 이미 변경을 전제로 한 모듈이다. 따라서 이러한 두 가지 종류의 모듈을 사용하는 방법에 있어서도 차이가 있다. 완성형 모듈을 사용하는 기술을 완성형 합성이라고 하며, 수정형 모듈을 사용하는 기술을 수정형 합성 기술이라고 한다. 자동화 기술에서는 수정을 가하지 않아도 되는 완성형 합성 기술을 주로 이용하며 수직 순 변환기술의 하부 기술로서 이용된다. 즉, 명세로부터 원시 코드를 생성할 때 미리 준비된 라이브러리의 요소를 이용하는 방식으로 사용된다.

명세 정형화는 일정한 영역의 지식을 정형화된 구문의 집합으로 표현함으로써 목적 시스템에 대한 표현이 보다 분명하게 이루어지며, 또한 기계적으로도 검증이 가능하도록 한다. 경우에 따라서는 이러한 명세를 수행시킬 수 있는 수행기의 개발도 가능하다. 정형화는 그래픽 명세 정형화와 텍스트 명세 정형화등으로 나눌 수 있으나 일반적으로 정보의 내용은 같고 흔히 두가지를 함께 사용한다. 그래픽 명세는 사용자의 편리를 제공하며, 텍스트 정보는 정보의 저장과 검증에 편리하기 때문이다. 자동화 도구에서는 정형적인 텍스트 및 그래픽 명세언어를 정의하여 사용하는 것이 일반적이나 반드시 그래야만 하는 것은 아니며, 서로의 이용성에 따라 보완적으로 또는 어떤 유형 한가지 만이 사용되기도 한다.

컴파일러 분야의 번역 기술은 재사용 기술의 변환 기술과 유사하며 여기에 명세를 수행시켜 결과를 검증할 수 있도록 한다. 자동화 기술에서 이기술의 중요성은 명세 단계에서 시스템의 기능성 및 운용성을 포함한 외적 특성을 검증할 수 있도록 하는 중요한 기술이다.

그래픽 기술은 사용자의 이용성을 향상 시킬 수 있는 핵심 기술이다. 그래픽 기술은 사용자의 명세 작업이 시각적으로 이루어 질 수 있도록 한다.

GUI Mosaic이 지원하는 OSF/Motif는 이 도구의 응용 영역 지식에 해당한다.

Motif는 복합 위짓을 포함하여 대략 50개의 기본 위짓을 라이브러리로서 제공하며 이와 관련된 다양한 기능을 제공하는 객체지향적인 구조를 가지고 있다. 그러나 엄밀히 말하면 캡슐화가 되어있지 않으므로 객체지향의 개념을 제공한다고 볼 수는 없다. 현재는 버젼 1.2가 제공되고 있으며 사용자의 수가 늘어 감에 따라 UNIX 기계의 표준 GUI로 자리잡고 있으며, 점점 안정적으로 세계화를 지향하고 있다.

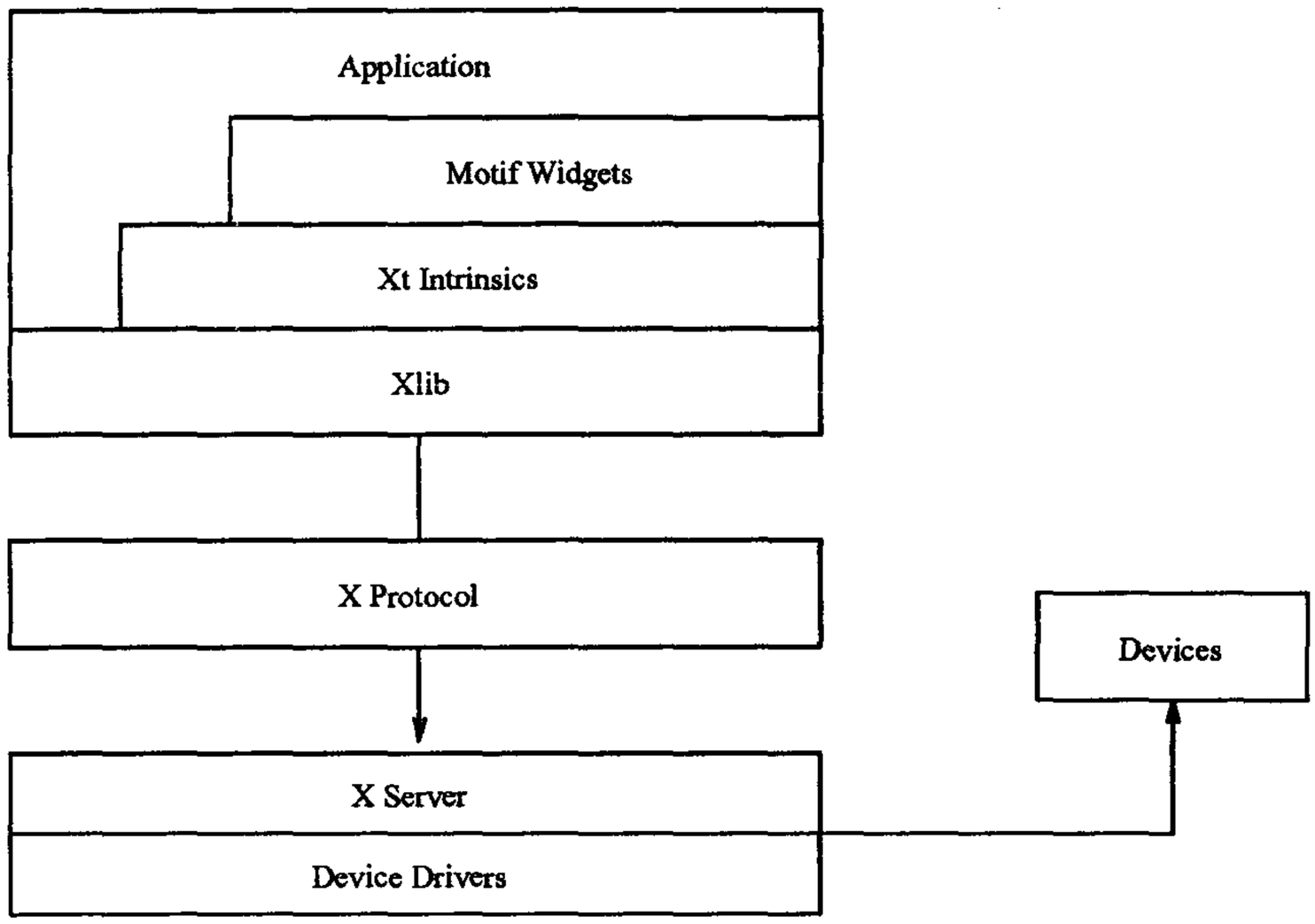
제 2 절 Motif 모형

1. Motif 응용 개발 모형

Motif 위짓은 Xt Intrinsics의 위짓에 기초하고 있으며, Xt Intrinsics는 Xlib에 기초하고 있다. Xlib의 기능은 X Protocol에 따라 X Server와 교신하며, X Server는 하드웨어 디바이스 제어기를 통해 사용자와 교신하게 된다. 사용자가 Motif 응용 소프트웨어를 개발하면 [그림 3-2-1]과 같은 환경에서 그 응용 소프트웨어는 운영된다[13].

2. Motif의 구조

Motif는 세가지 의미를 갖는 시스템으로서, 윈도우 관리기이며, 위짓 라이브러리를 제공하며, 특수 전용 언어를 제공한다. 윈도우 관리기는 화면 상의 각종 윈도우 객체를 보기 좋게 치장하며 동시에 그들을 관리할 수 있는 기능을 제공한다.

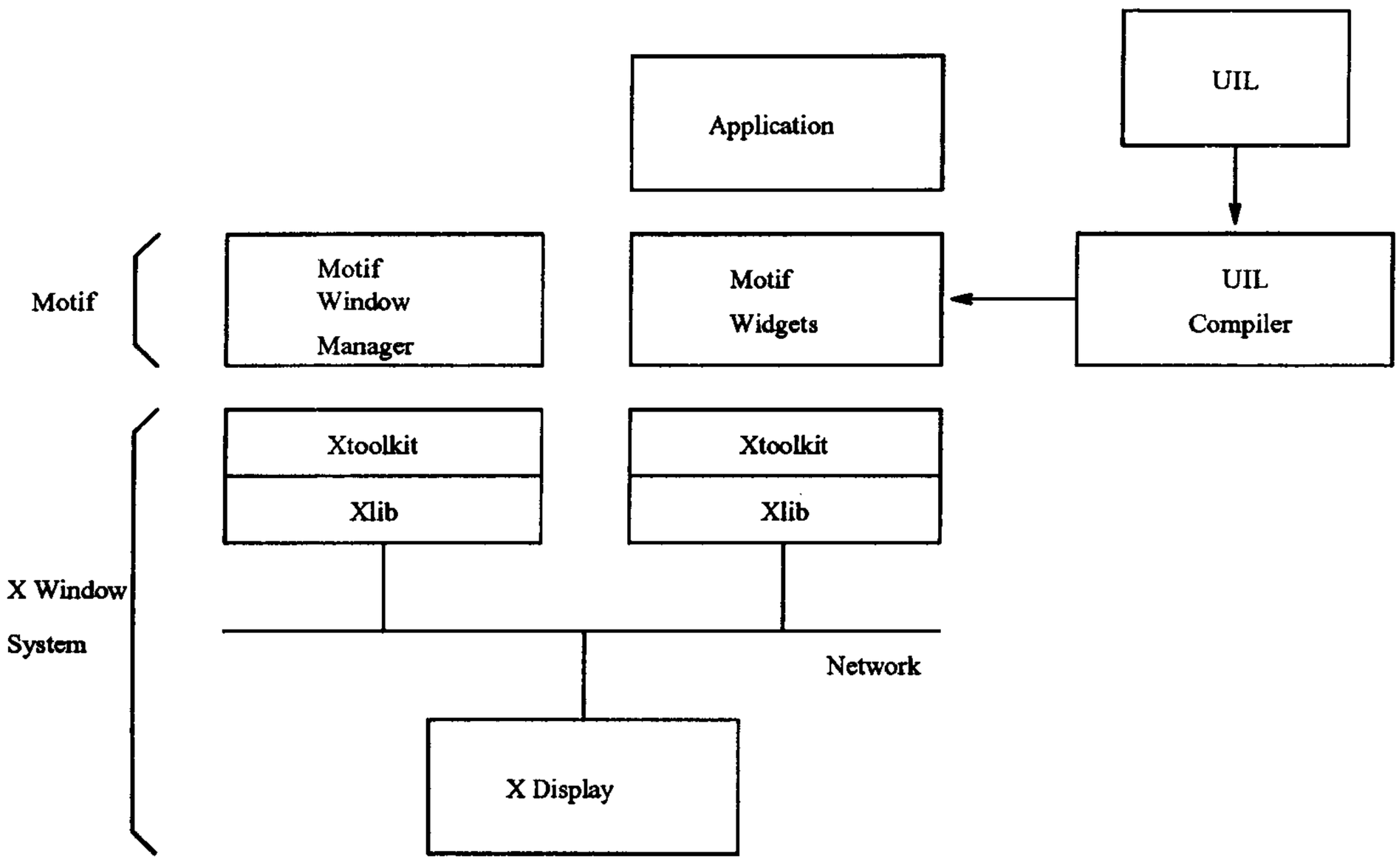


[그림 3-2-1] Motif의 응용 소프트웨어 개발 모형

Motif 위짓 라이브러리는 Xt Intrinsics의 위짓 들에 기초하여 재구성한 위짓들의 집합으로서 Motif 만이 갖는 독특한 윈도우를 제공하며, 그와 관련된 다양한 기능들을 제공한다.

전용 사용자 접속부 명세 언어인 UIL(User Interface Language)은 Motif 스타일의 사용자 접속부를 기존의 C 언어보다 편하게 명세할 수 있도록 만든 언어로서 컴파일러가 따로이 준비되어 있다. 사용자는 UIL 코드를 작성하고 UIL 컴파일러로 해당 코드를 컴파일하여 C 또는 C++로 작성된 프로그램에서 불러 사용하면 된다.

[그림 3-2-2]는 Motif의 구성요소를 보여준다.



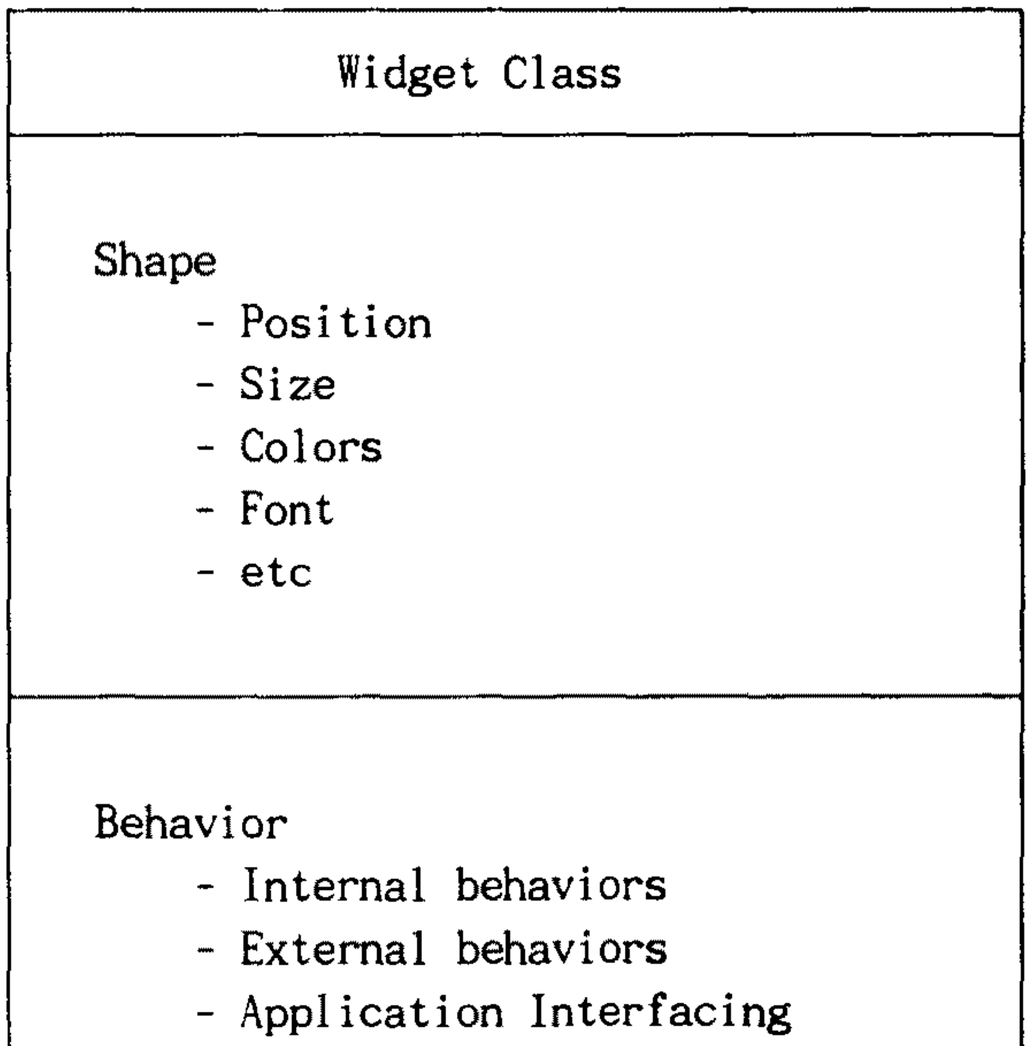
[그림 3-2-2] Motif의 구성 요소

3. Motif 위짓의 모형

Motif 위짓은 실제적으로는 객체라고 볼 수 없으나, 개념적으로는 객체의 관점에서 바라보면 보다 이해하기 쉽다. 또한 Motif 위짓은 내적으로 Xlib에 기초한 Xt Intrinsics에 기초하고 있으므로, 하나의 Motif 위짓에는 상당히 많은 요소들이 관여되어 있으나 단순하게 Motif에서 보여지고 있는 위짓의 특성만을 하나의 객체형 모형으로 표현하면 [그림 3-2-3]과 같다. [그림 3-2-3]에서와 같이 객체의 이름을 나타내는 이름부와 여러가지 동적 특성을 나타내는 모양부와 동적

특성을 나타내는 행위부로 나누어 표현할 수 있다.

행위부에서 내부 행위(internal behaviors)란 위짓의 사적인 행위들로서 사용자에게는 제공되지 않으나 위짓의 특성을 나타내기 위해 미리 정의되어 있는 행위들을 말한다. 외부 행위(external behavior)란 위짓 외부에서 위짓의 특성을 변경할 수 있는 기능들의 집합을 의미한다.예를 들면 위짓의 색상을 변경하는 기능을 들 수 있다. 마지막으로 응용 접속 행위란 해당 위짓에서 발생한 서버의 사건에 대응하여 발생하는 동적 기능의 집합을 의미한다. Motif에서는 Action, EventHandler, Callback 등의 세가지 방법이 제공된다. Callback의 방법은 어떤 특정한 하나의 사건에 대응하여 지정된 기능을 수행하기 편리하게 정의하여 제공하는 접속 행위의 방법이고, EventHandler의 경우는 이보다는 많으나 역시 몇가



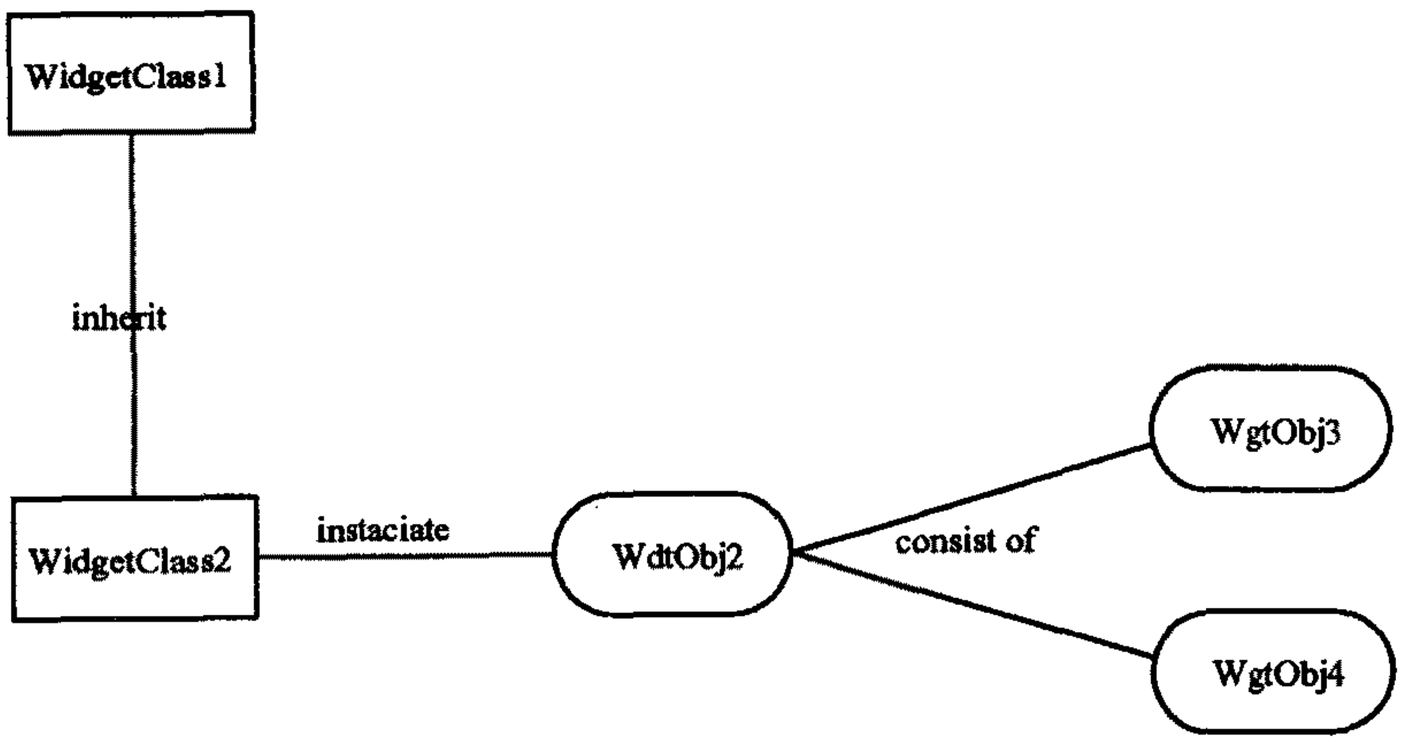
[그림 3-2-3] Motif 위짓의 구성 모형

지의 사건을 받아들여 분기하여 기능을 수행할 수 있도록 제공하며, Action의 경우는 Translation이라는 방법을 통하여 보다 일반적인 사건들을 구애없이 다룰 수 있도록 제공하는 접속 행위의 방법이다. 이러한 접속 방법들을 통하여 GUI부는 전체적으로 사용자가 원하는 화면의 관리와, 응용 프로그램부가 편리하게 연결되어 사용자가 원하는 소프트웨어의 개발을 쉽게 도와준다.

4. Motif 위짓 사이의 관계 모형

Motif 위짓 사이의 관계는 미리 정의된 관계인 상속 관계와, 사용자 접속부의 구조와 관련된 구축(construction) 관계의 두가지 관계가 있다.

상속 관계는 위짓의 속성을 나타내는 자원의 포함 관계를 나타내며 이 관계로서 위짓의 개략적인 특성을 파악할 수 있다. 구축 관계는 사용자 접속부의 구조를 보여주는 관계로서 접속부의 개략적인 모양을 알 수 있다. 그런데 모든 위짓이 모든 위짓과 구축의 관계를 가질 수 있는 것은 아니고 그 관계의 가능성은 상속의 관계에 비해 매우 복잡하다. 이 관계는 제 4 절의 GUI Mosaic 명세에서 설명한다. [그림 3-2-4]는 위짓 사이의 관계를 일반화하여 보여준다.



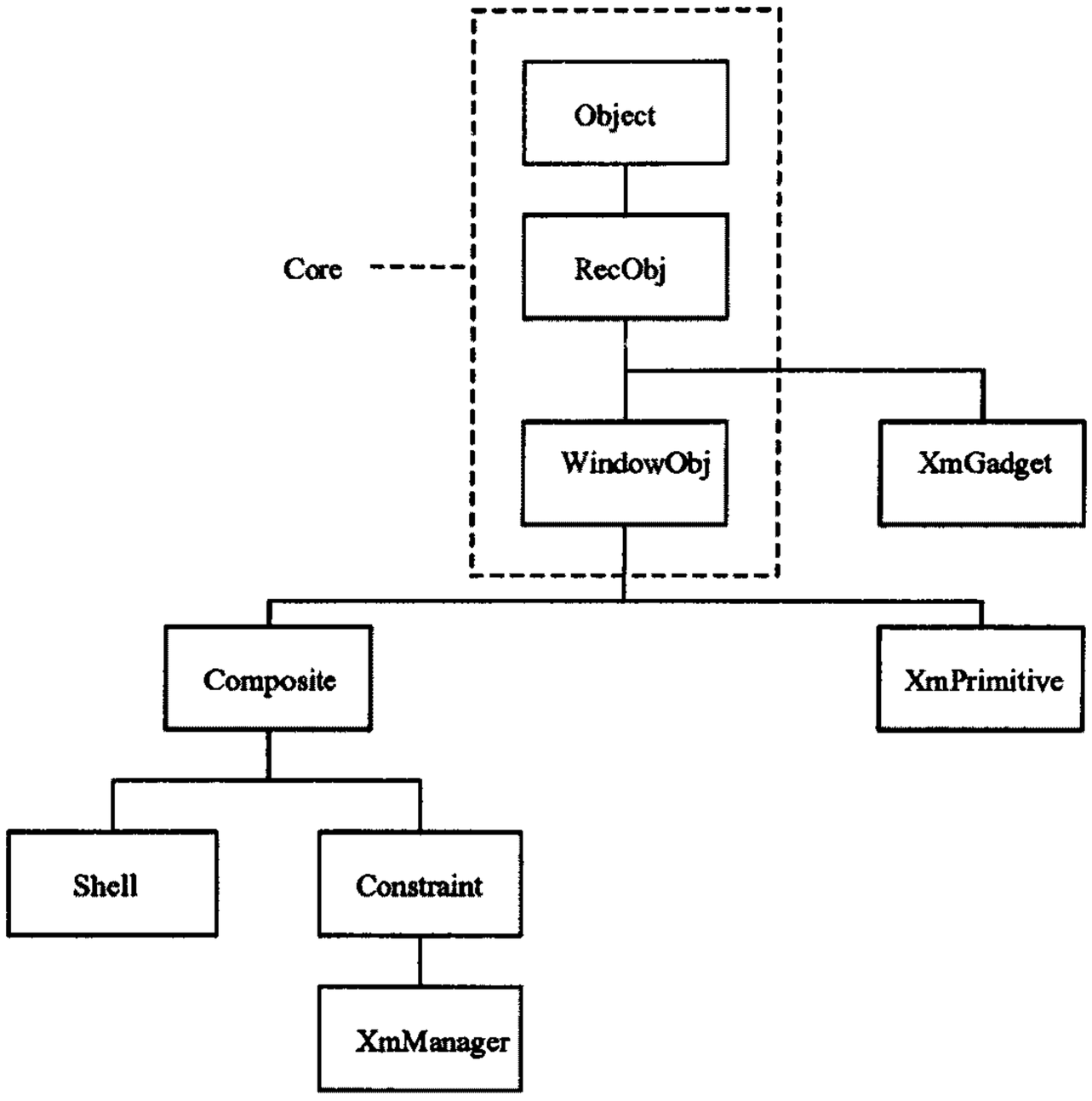
[그림 3-2-4] Motif 위짓 사이의 일반적인 관계

[표 3-2-1] Motif 위짓의 특성에 따른 분류

종 류	클 래 스 이 름
Shell Widgets	. XmDialogShell . XmMenuShell . VendorShell
Display Widgets	. Core . XmPrimitive . XmArrowButton . XmDrawnButton . XmLabel . XmList . XmPushButton . XmScrollBar . XmSeperator . XmText . XmToggleButton
Container Widgets	. XmManager . XmDrawingArea . XmFrame . XmMainWindow . XmPanedWindow . XmRowColumn . XmScale . XmScrolledWindow
Dialog Widgets	. XmBulletinBoard . XmCommand . XmFileSelectionBox . XmForm . XmMessageBox . XmSelectionBox
Menu Widgets	XmCascadeButton XmCascadeButtonGadget
Gadgets	. Object . RectObj . XmGadget . XmArrowButtonGadget . XmLabelGadget . XmPushButtonGadget . XmSeperatorGadget . XmToggleButtonGadget

5. Motif 위짓의 분류

Motif 위짓은 특성에 따른 분류와 상속 관계에 따른 분류의 두가지로 나눌 수 있다. <표 3-2-1]은 특성에 따른 분류를 보여주며, [그림 3-2-5]는 상속 관계에 따른 분류를 보여준다[13].



[그림 3-2-5] Motif 위짓의 상속 관계에 따른 분류

제 3 절 GUI Mosaic의 개념적 모형

1. 시스템 정의

가. GUI Mosaic과 들무새 시스템

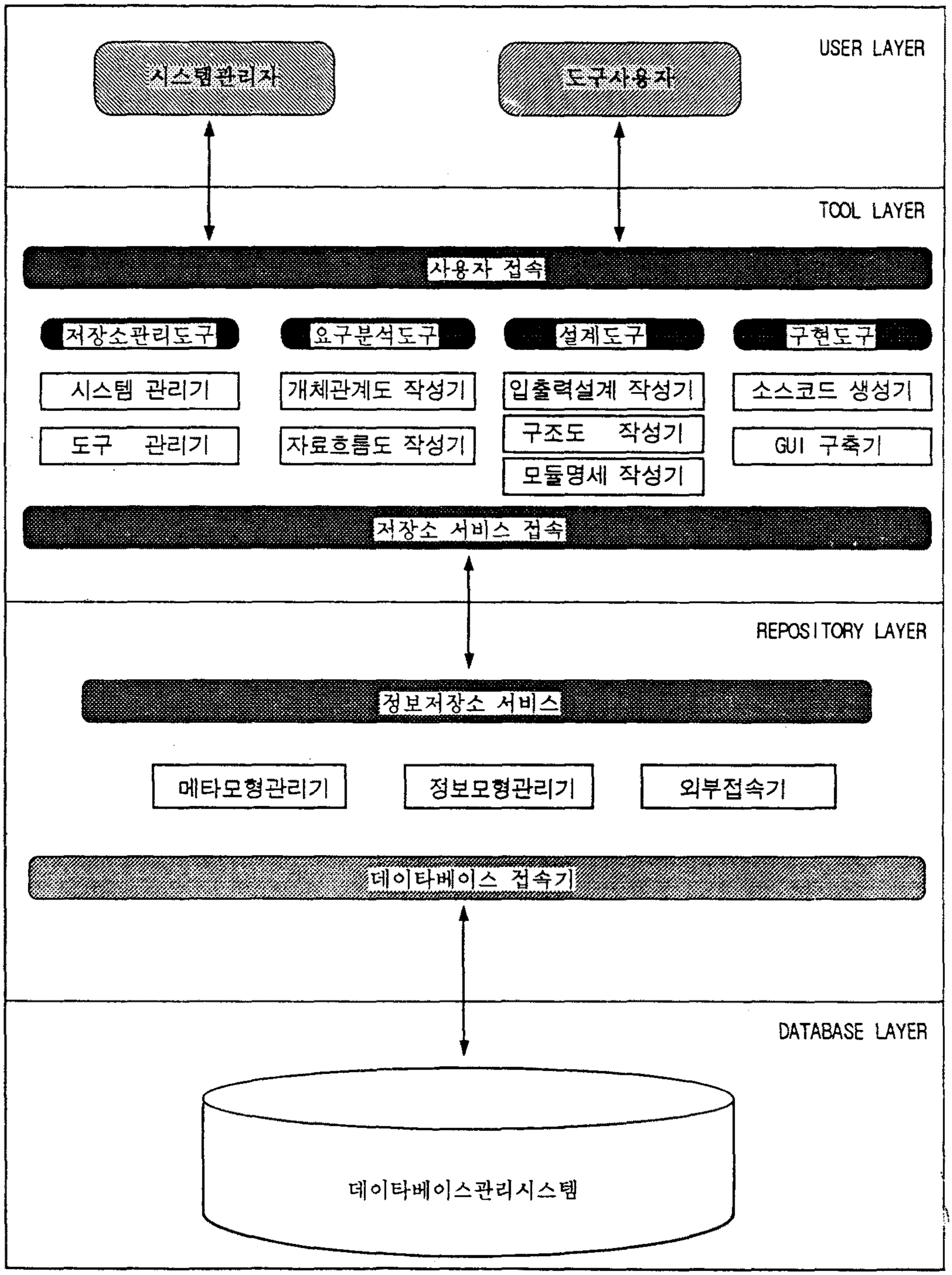
GUI Mosaic은 Motif 스타일을 따르는 GUI 개발을 지원하는 도구이다. 이도구는 정보저장소 중심의 통합 소프트웨어 개발 지원도구로 개발 중인 들무새

(Deulmusae) 시스템의 단위 시스템으로, 자료의 관리는 정보 저장소의 지원을 받으며, GUI는 타 도구와 공통으로 사용한다. 그러나 정보저장소를 통하지 않는 타 도구와의 직접적인 자료 교환은 없다. 즉 들무새와의 통합은 정보저장소와 GUI 수준에서 이루어 진다. [그림 3-3-1]은 들무새 시스템의 계층구조를 보여주며, [그림 3-3-2]는 들무새 시스템의 기능의 모형을 보여주며, [그림 3-3-3]은 자료 흐름의 모형을 보여준다.

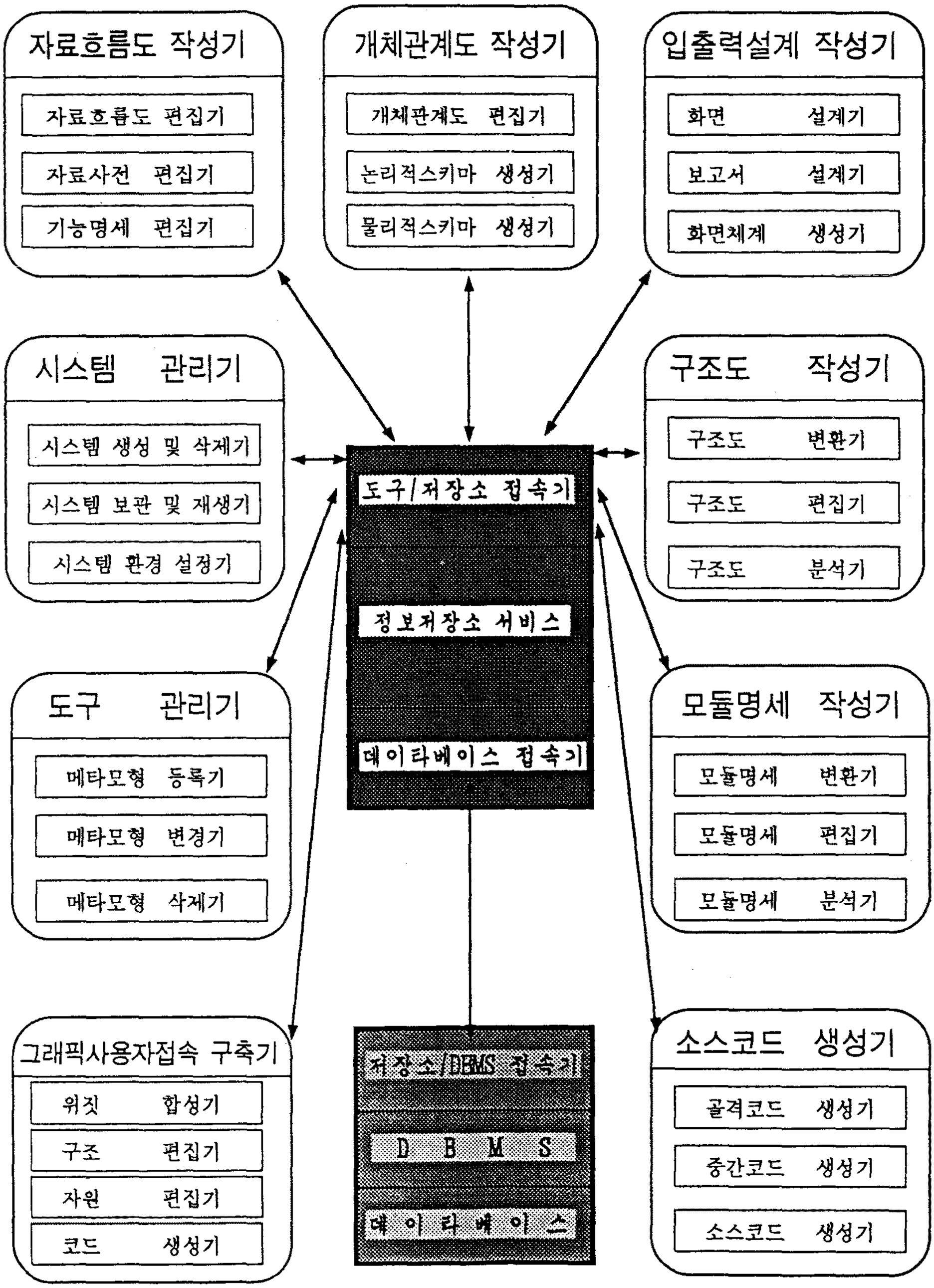
들무새의 구조는 [그림 3-3-1]에서와 같이 사용자 계층(User Layer), 도구 계층(Tool Layer), 저장소 계층(Repository Layer), 그리고 데이타 베이스 계층(Database Layer)의 네개의 계층으로 이루어져 있다. 사용자 계층은 도구의 사용자와 시스템 관리자로 구분되고, 도구 계층은 단위 도구에 공통적인 사용자 접속과 저장소 관리 도구 군, 요구 분석 도구 군, 설계도구 군, 그리고 구현도구 군으로 도구군이 구성되며, 이러한 단위 도구들은 저장소 서비스 접속을 지원 받는다. 저장소 계층은 도구에 대한 정보 저장소 서비스 부와 서비스를 지원하는 외부 접속기와 저장소 자체의 관리를 위한 메타모형 관리기, 정보모형 관리기와 데이타베이스와 접속하기 위한 데이타베이스 접속기로 구성된다. 마지막으로 모든 논리적인 절차를 거친 물리적인 자료는 DBMS에 의해 저장되어 관리된다. 들무새에서 이용하는 DB는 관계형이다.

나. GUI Mosaic 정의

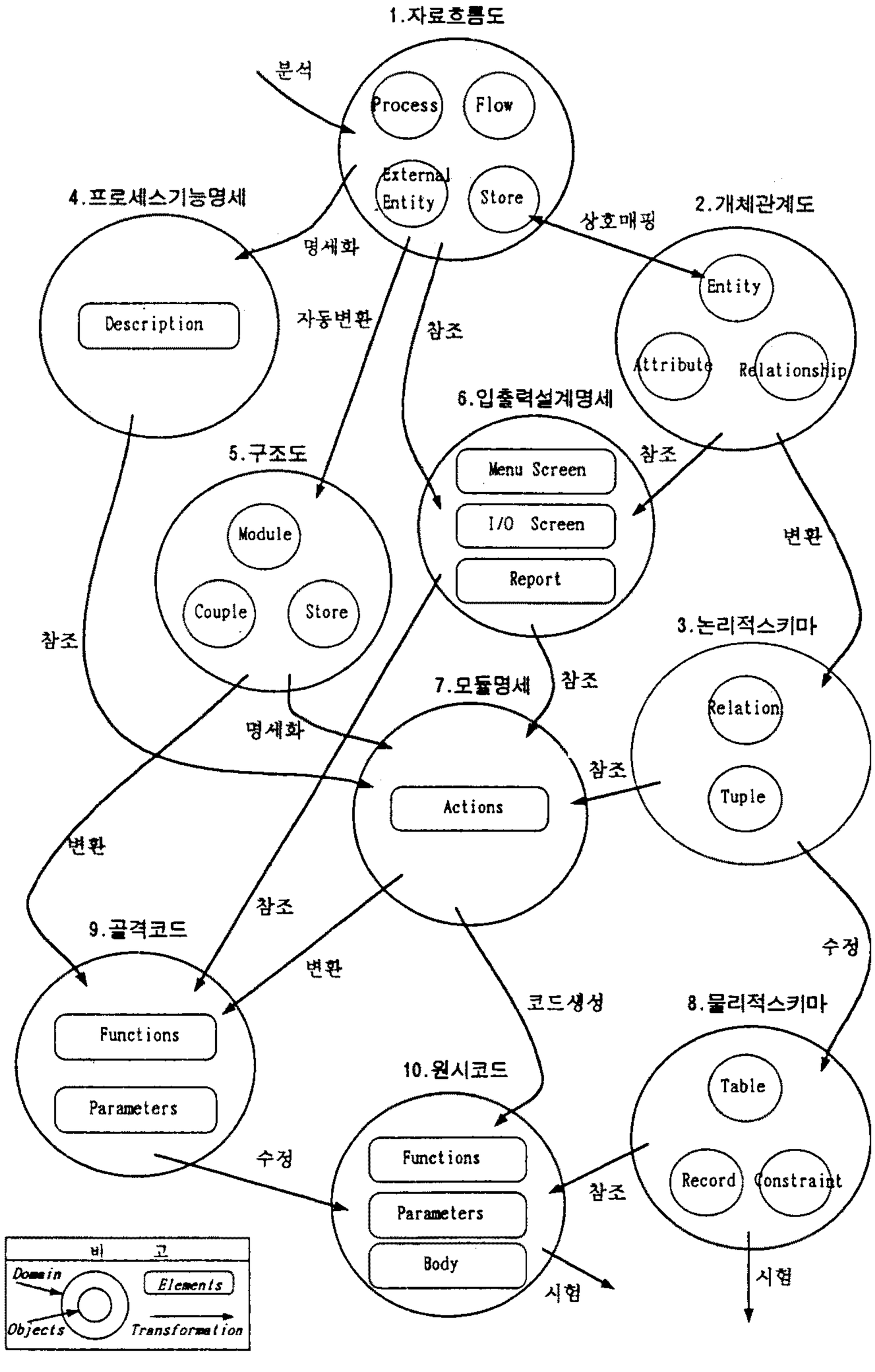
GUI Mosaic은 Motif 스타일의 GUI 개발을 최대한 자동화하려는 도구로서, 이 도구의 사용자는 GUI 분석을 하는 분석가, GUI의 구체적인 모양 및 행위를 결정하는 설계자, 설계대로 구현을 하는 프로그래머, 그리고 유지보수를 하는 유지보수자가 될 수 있다.



[그림 3-3-1] 들무새 시스템의 계층구조



[그림 3-3-2] 들무새 시스템 기능 모형



[그림 3-3-3] 들무새 시스템 자료흐름 모형

이 도구는 분석 단계에서는 대략적인 GUI의 구성을 지원하며, 또한 그 결과는 실제 화면으로 검증할 수 있도록 지원한다. 설계 단계에서는 상세한 명세 작업을 지원하며, 구현 단계는 코드 생성을 통하여 목적 원시코드 및 컴파일 환경을 생성하는 완전한 자동화로 구현자를 지원한다. 즉, 설계자와 프로그래머의 구분이 이 도구에서는 없다.

사용자의 작업은 다양한 위짓을 사용하여 여러개 단위 접속부를 구성하는 작업, 각 접속부의 구조를 조정하는 작업, 각 접속부의 구성 위짓의 속성을 정의하는 작업, 위짓과 위짓, 위짓과 접속부, 또는 위짓 자체의 행위를 정의하는 작업으로 이루어 진다. 이러한 접속부에 대한 제반 정의 작업이 완료되면 정의된 명세로 부터 목적하는 원시코드 및 컴파일 환경을 생성한후 컴파일 과정을 거쳐 최종의 수행코드를 얻는다. 이후로는 변경에 의한 유지보수 작업이 명세 수준에서 이루어 진다. 이러한 일련의 모든작업은 시각적인 환경에서 이루어지며 대화식으로 진행되어야 한다.

접속부를 구성하는 작업은 위짓을 근간으로 이루어지며, 작업은 다음과 같은 방법으로 일어난다. 사용자는 선택 가능한 모든 위짓을 모아 놓은 위짓의 팔레트(palette)에서 가장 상위의 위짓에 해당하는 쉘(Shell) 위짓 중의 하나를 선택하여 접속부의 정의를 시작한다. 이후로 가능한 위짓을 선택하고 부모 위짓 객체를 정의한 후 마우스를 이용하여 대략적인 위치와 크기를 조정한후 또다시 마우스의 버튼을 이용하여 화면상에 원하는 위짓의 객체를 만든다. 이 객체는 프로그램의 수행에 의해 나타나는 객체와 모양에 있어서 동일하다. 이러한 단위작업을 반복적으로 수행하여 원하는 단위 접속부를 만든다. 이러한 단위 접속부를 여러개 만들 수도 있다.

접속부의 틀을 만드는 작업이 끝나면 각 단위 접속부의 구조를 조정한다. 사용자는 단위 접속부의 구조를 보여주는 조회기(Browser)에서 위짓의 이름을 기준

으로 선택할 수 있다. 모든 작업은 대상 위짓을 선택한 후 원하는 작업을 할 수 있는 방식으로 구성된다. 가능한 작업은 단일 위짓에 대한 이름의 변경, 클래스의 변경등이 있으며, 모든 하위 위짓을 포함하는 작업으로는 복사, 삭제, 자르기, 붙이기 등이 있으며, 모든 작업은 복구가 가능하도록 회복(Undo) 작업이 가능하다.

접속부의 구조를 조정하는 작업이 끝나면 각 위짓의 속성을 정의한다. 사용자는 자원 편집기에서 위치 및 모양과 관련된 속성들을 편집한다. 이때 미리 정의된 속성 요소의 값들에 대해서는 나열에 의한 선택 방법을 사용한다. 색상의 편집은 전문적인 색상 편집기를 이용한다. 행위를 정의하는 작업중 콜백(Callback)의 정의는 콜백 편집기에서 콜백 함수와 그 파라미터를 정의함으로써 이루어지고, 사건처리(EventHandler)는 사건처리 편집기에서 사건과 처리 함수 및 그 파라미터를 정의함으로써 이루어지고, 액션(Action)의 경우는 번역(Translation) 편집기에서 사건과 함수의 대칭 표 작성과 함수를 정의하는 작업으로 구성된다.

일련의 명세 작업이 완료되면 명세는 명세 언어로 된 명세코드로 생성 및 관리되고, 이후의 GUI에 대한 변경은 명세코드를 시스템에 로드(load)하고 번역(interpretation)하여 앞서 설명한 작업들로서 이루어 진다. 또한 명세는 C 또는 C++ 원시코드로 생성된다. 생성되는 원시코드는 수정없이 컴파일 및 수행될 수 있도록 Makefile도 함께 생성되며, 사용자의 일부 변경을 위해 자원 화일도 생성한다.

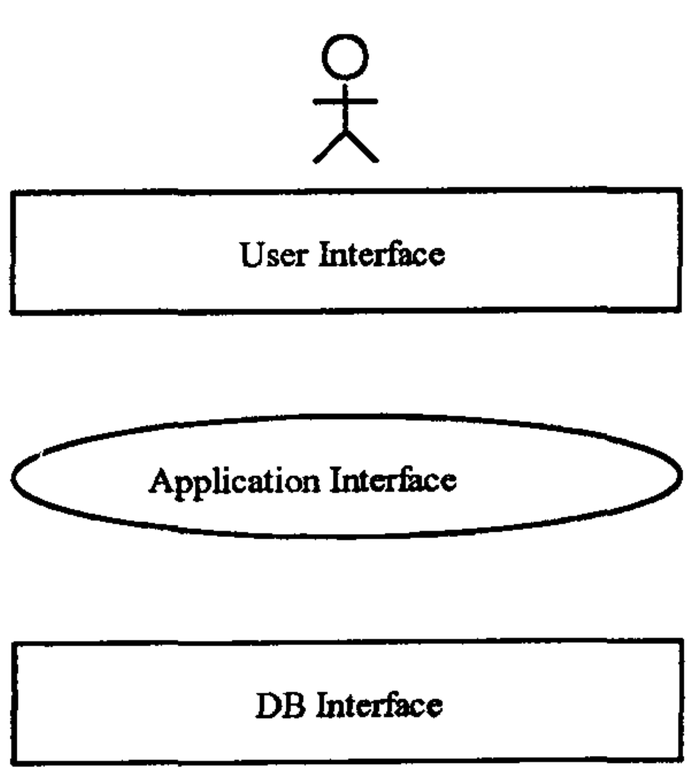
마지막 과정으로 사용자는 컴파일러 및 로더의 종류와 옵션을 선택할 수 있다. 이는 최종 소프트웨어의 이식성을 향상 시키는 데에 그 목적이 있다.

2. 소프트웨어 모형

현재 소프트웨어라는 단어는 프로그래밍 언어로 구성된 코드의 범위를 넘어서서 관련 문서를 포함한 컴퓨터로 접근 가능한 모든 인공물을 지칭하는 경향이다. 여기서 소프트웨어란 전통적으로 말하는 프로그램 코드로 제한한다.

소프트웨어는 기술적인 관점에서 사용자 접속부, 응용 접속부, 자료 접속부 및 데이타베이스로 구성되었다고 해석할 수 있다[1]. 이러한 해석의 배경에는 사용자 접속부 관리 시스템, 데이타베이스 관리 시스템 및 응용 라이브러리의 존재를 가정하고 있다.

[그림 3-3-4]는 GUI Mosaic의 관점에서 보는 소프트웨어의 구성을 보여준다.

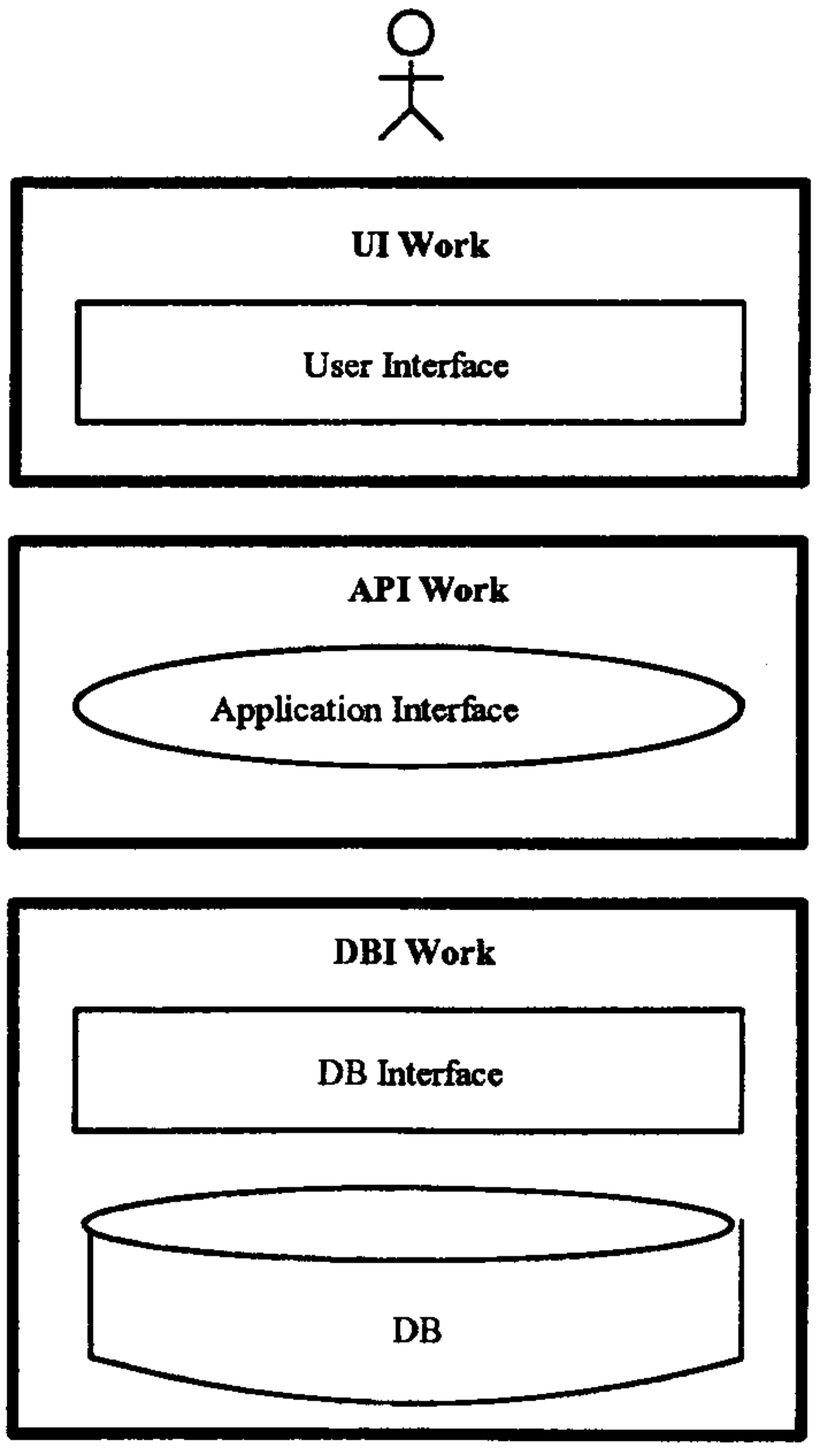


[그림 3-3-4] GUI Mosaic 관점에서 소프트웨어의 구성

3. 소프트웨어 개발작업 모형

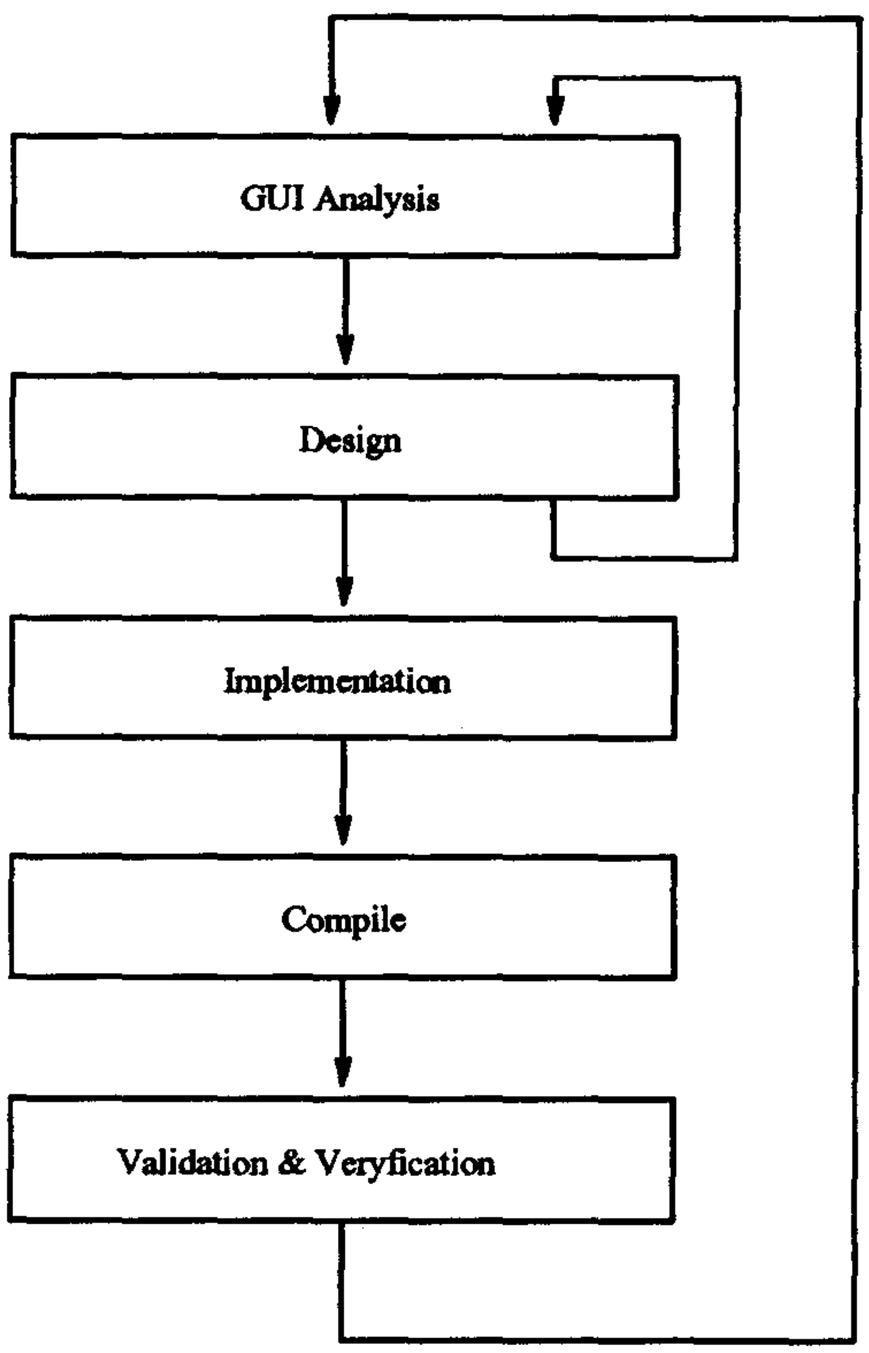
소프트웨어의 개발 작업은 소프트웨어의 구성 모형에 근거하여 사용자 접속부 개발 작업(UI Work), 응용 접속부 개발작업(Application Interface Work), 데이타베이스 접속부 개발작업(Database Interface Work)의 세가지 접속부 작업으로 구성되며, 접속 대상의 개발 작업으로 응용 라이브러리 개발 작업, 데이타베이스 구축 작업 두가지로 구분할 수 있다. GUI Mosaic에서는 이러한 다섯 가지의 작업들을 GUI 관련 작업과 그 외의 작업으로 구분하여 우선적으로 GUI 개발작업의 지원을 개발의 범위로 한다.

[그림 3-2-5]는 정적 관점에서의 소프트웨어 개발 작업 모형을 보여준다.



[그림 3-3-5] 소프트웨어 구성 관점에서 개발 작업의 분할

[그림 3-2-5]는 기술적인 관점에서 소프트웨어의 개발 작업을 구분한 그림이고, 개발 절차적인 관점에서는 [그림 3-3-6]과 같이 GUI 분석 작업, 설계 작업, 구현 작업, 컴파일 작업 및 확인 검증 작업으로 구성되는 순서적인 일련의 작업으로 구성된다. 도구는 두가지 관점에서 사용자의 개발 작업을 지원할 수 있어야 한다.



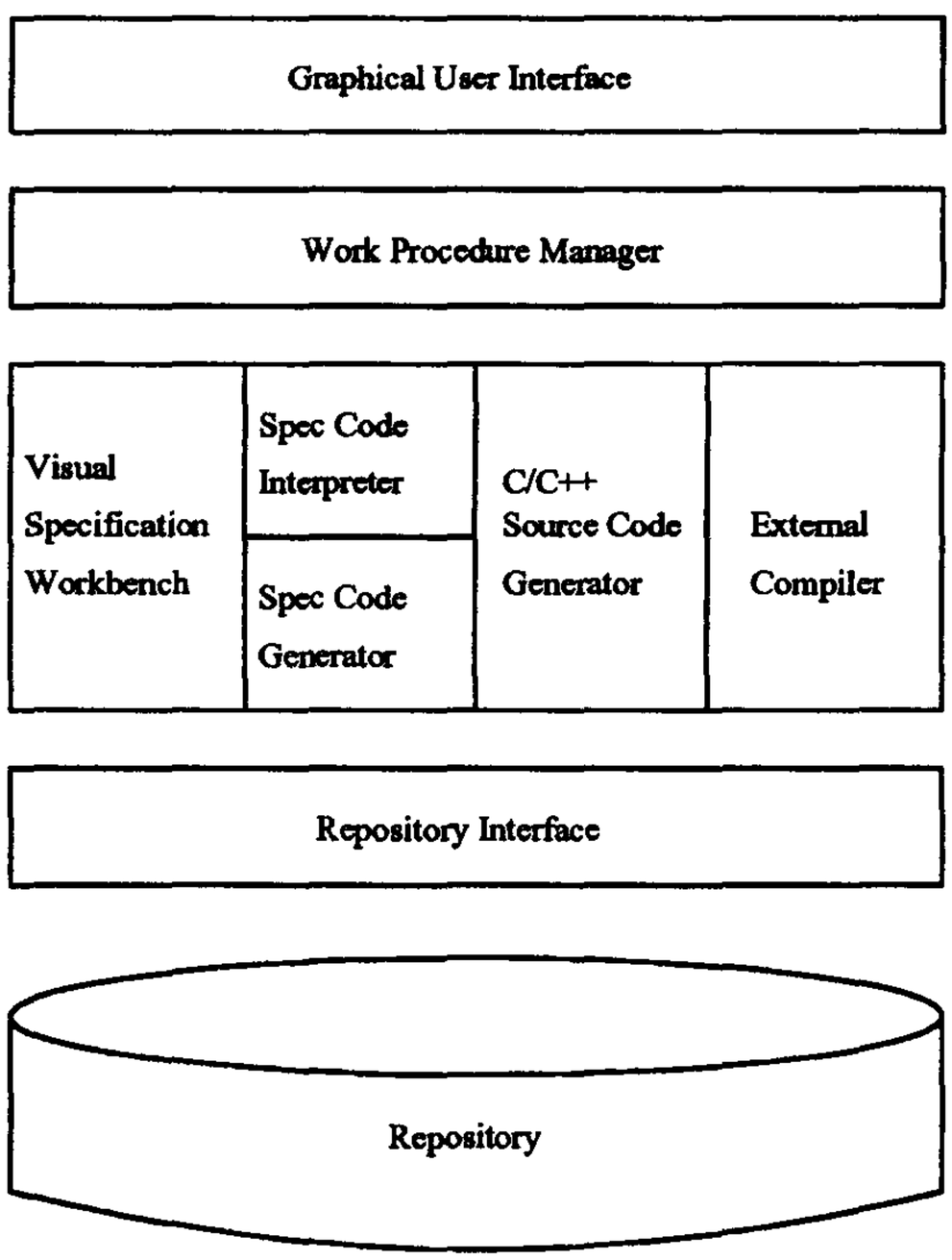
[그림 3-3-6] 소프트웨어 개발 절차 관점에서 개발 작업의 분할

4. 시스템 구성 모형

GUI Mosaic은 들무새 시스템의 단위 시스템으로 자료는 정보저장소에서 관리되고, GUI는 들무새의 화면으로 통합된다. 그러나 도구 수준에서 타 도구와의 정

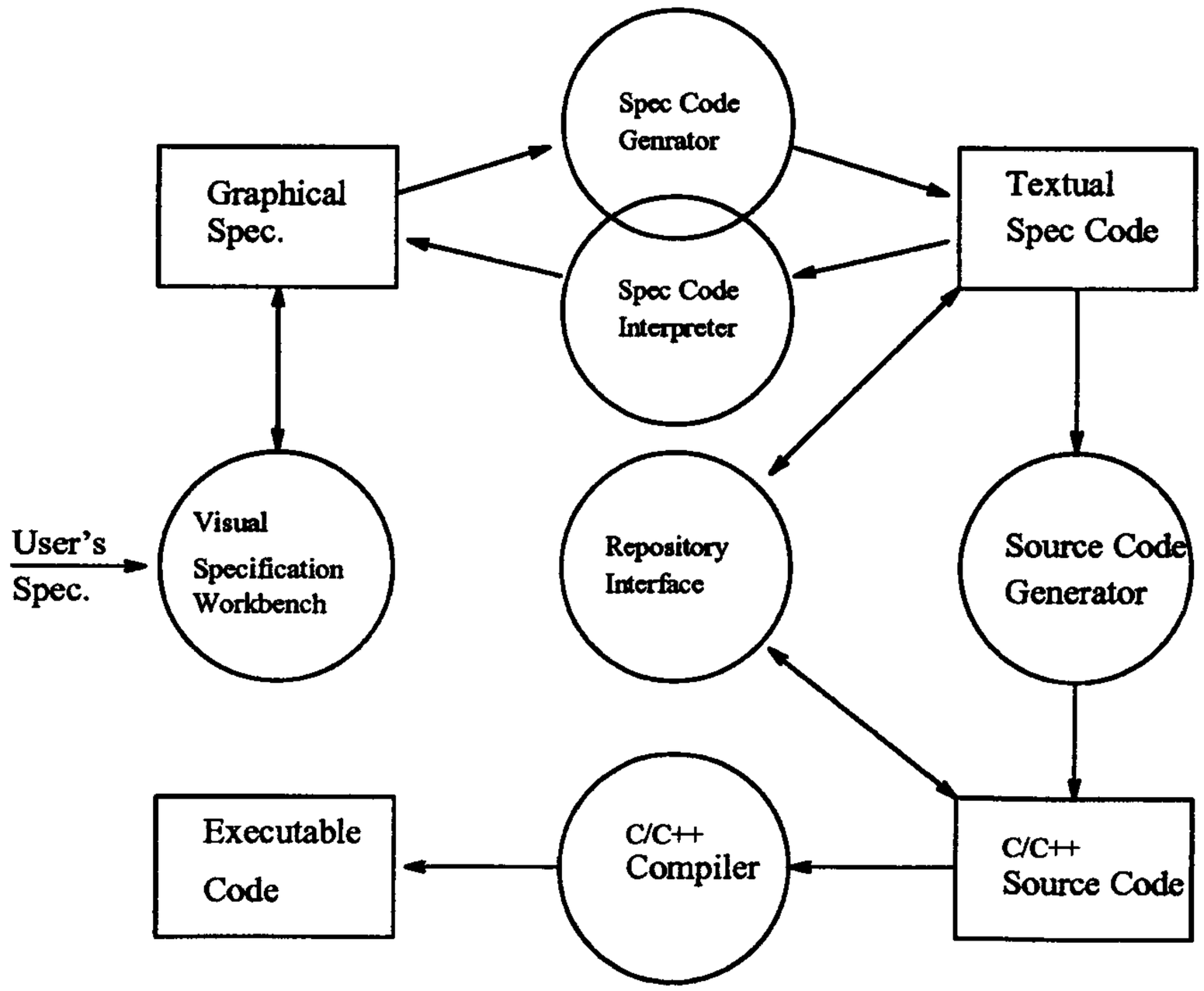
보 교환은 없다. [그림 3-3-7]은 시스템 구성 모형을 보여준다[11].

[그림 3-3-7]에서와 같이 GUI Mosaic의 주요 구성 요소는 사용자의 GUI 개발 작업 절차를 관리해 주며, 도구의 하부 시스템들을 통합한 작업절차관리기(Work Procedure Manager), 시각적인 명세작업을 지원해 주는 시각 명세작업대(Visual Specification Workbench), 명세작업의 결과를 명세 언어의 코드로 바꾸어주는 명세코드생성기(Spec Code Generator), 반대로 명세코드를 작업 영역으로 올리고 번역하는 명세코드번역기(Spec Code Interpreter), 명세를 입력으로 목적 C/C++ 원시코드를 생성하는 C/C++ 원시코드생성기(C/C++ Source Code Generator), 원시코드를 컴파일하는 외부의 C/C++ 컴파일러(Compiler), 그리고 명세정보를 정보저장소에 저장하거나 읽어오기 위한 저장소접속기(Repository Interface) 등이다.



[그림 3-3-7] GUI Mosaic의 주요 구성

각 하부 시스템 사이에는 [그림 3-3-8]과 같은 관계를 가지고 있다. 사용자는 시각 명세작업대에서 자신의 GUI를 정의하여 그래픽 명세(Graphical Spec)를 만들고, 이 그래픽 정보는 명세코드생성기에서 텍스트 형식의 명세코드로 바뀌고, 반대로 명세코드 번역기에서 그래픽 명세로 전환되어 재 편집된다. 텍스트 명세는 원시코드 생성기(Source Code Generator)에서 C 또는 C++ 원시코드로 바뀌고, 이 원시코드는 해당 언어 컴파일러(Compiler)에 의한 컴파일 과정을 거쳐 수행 가능한 코드(Executable Code)로 바뀐다. 텍스트 명세와 원시코드는 저장소 접속을 통해 저장소에 관리된다.

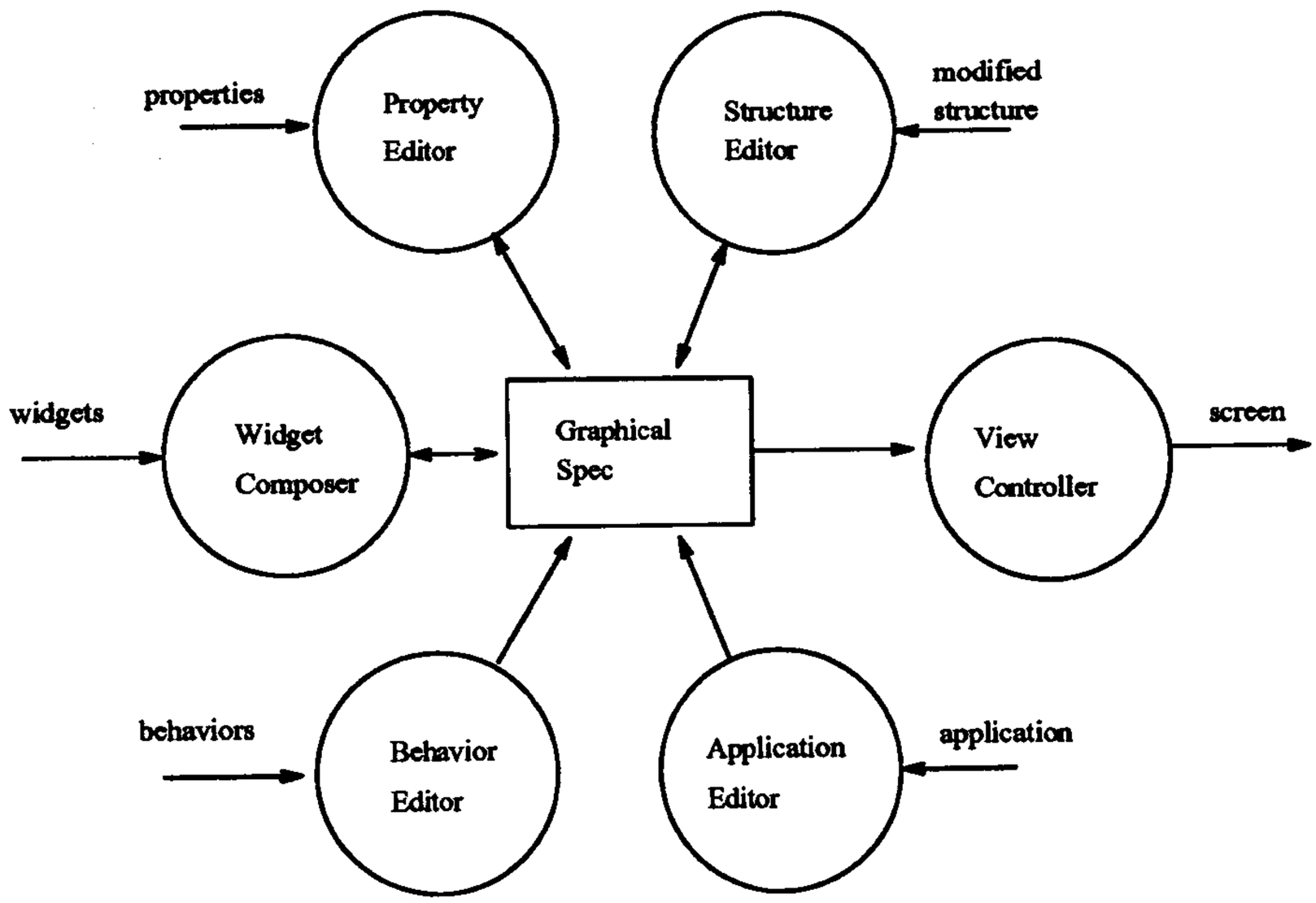


[그림 3-3-8] GUI Mosaic 하부 시스템 사이의 관계

5. 명세 작업 모형

명세 작업은 사용자와 가장 많은 대화가 이루어 지며, 도구의 운용성이 가장 중요시 되는 작업이다. 명세 작업은 위짓의 합성 및 결과의 반영, 위짓의 속성 편집 및 결과의 반영, 구조의 편집 및 결과의 반영, 행위의 편집, 그리고 응용 프로그램의 편집 등으로 구성된다. 이러한 작업들은 위짓을 중심으로 유기적으로 이루어 진다. 따라서 명세작업은 시각적이며 운용이 단순해야 하며, 대화식 작업이며 즉시 검증성이 있어야 한다.

[그림 3-3-9]는 이러한 작업들 사이의 관계를 보여주는 그림이다.



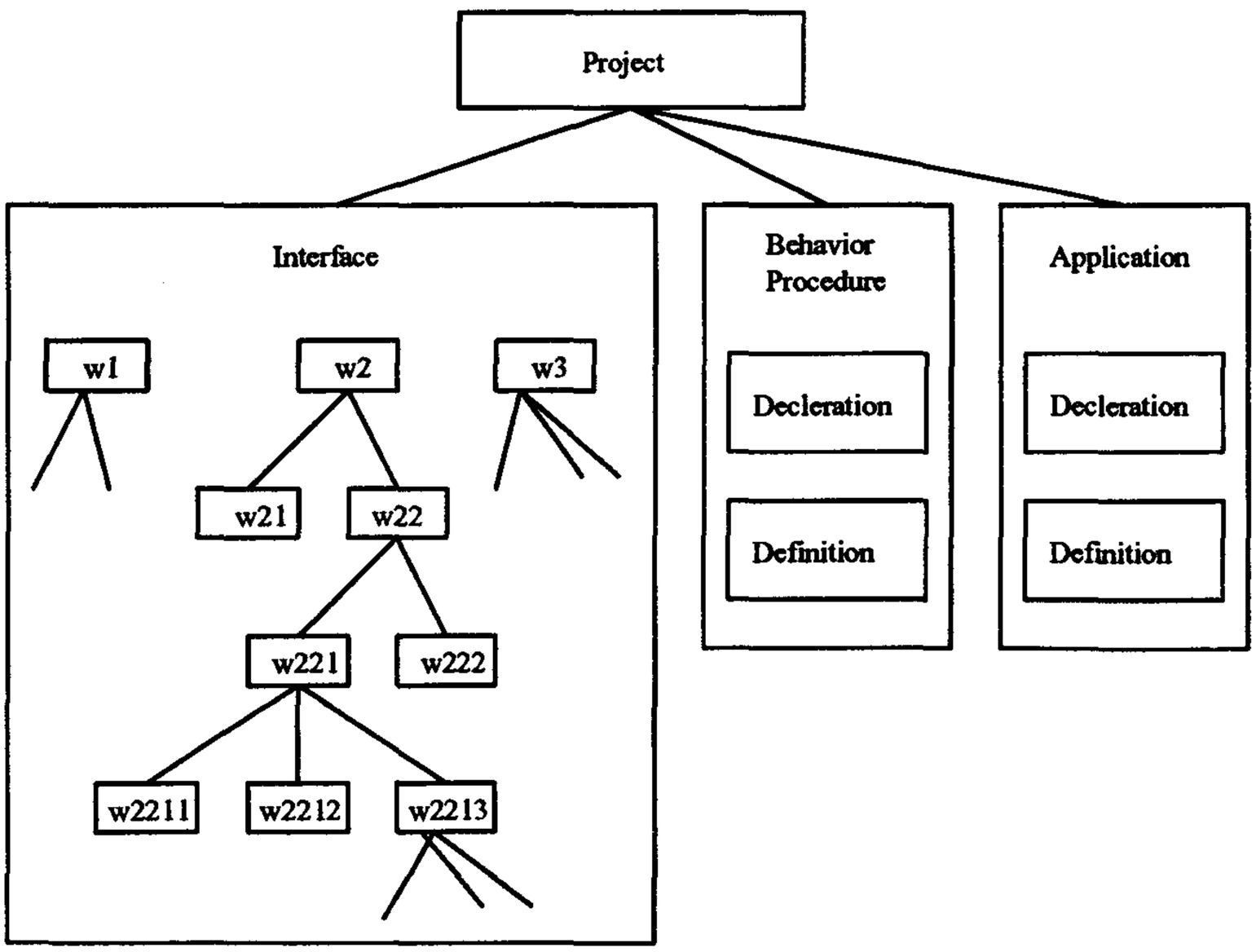
[그림 3-3-9] GUI Mosaic의 명세 작업 모형

6. 명세 표현 모형

명세의 모음을 프로젝트라고 하며, 프로젝트(Project)는 인터페이스(Interface), 위짓의 행위(Behavior), 그리고 응용(Application) 정보로 구성된다. 각 명세 정보는 각기 다른 객체로 표현한다. 프로젝트 정보는 프로젝트 클래스, 인터페이스 정보는 위짓 클래스, 행위 정보는 행위 선언 클래스와 행위 정의 클래스, 응용 정보는 응용 선언과 응용 정의 클래스로 구분한다.

프로젝트 클래스는 한 프로젝트를 대표하며, 인터페이스 클래스는 인터페이스의 구조 및 특성을 표현하며, 행위 클래스들은 위짓이 갖는 행위들을 표현하며, 응용 클래스는 응용 프로그램을 표현한다.

[그림 3-3-10]은 명세의 표현 모형을 보여준다.

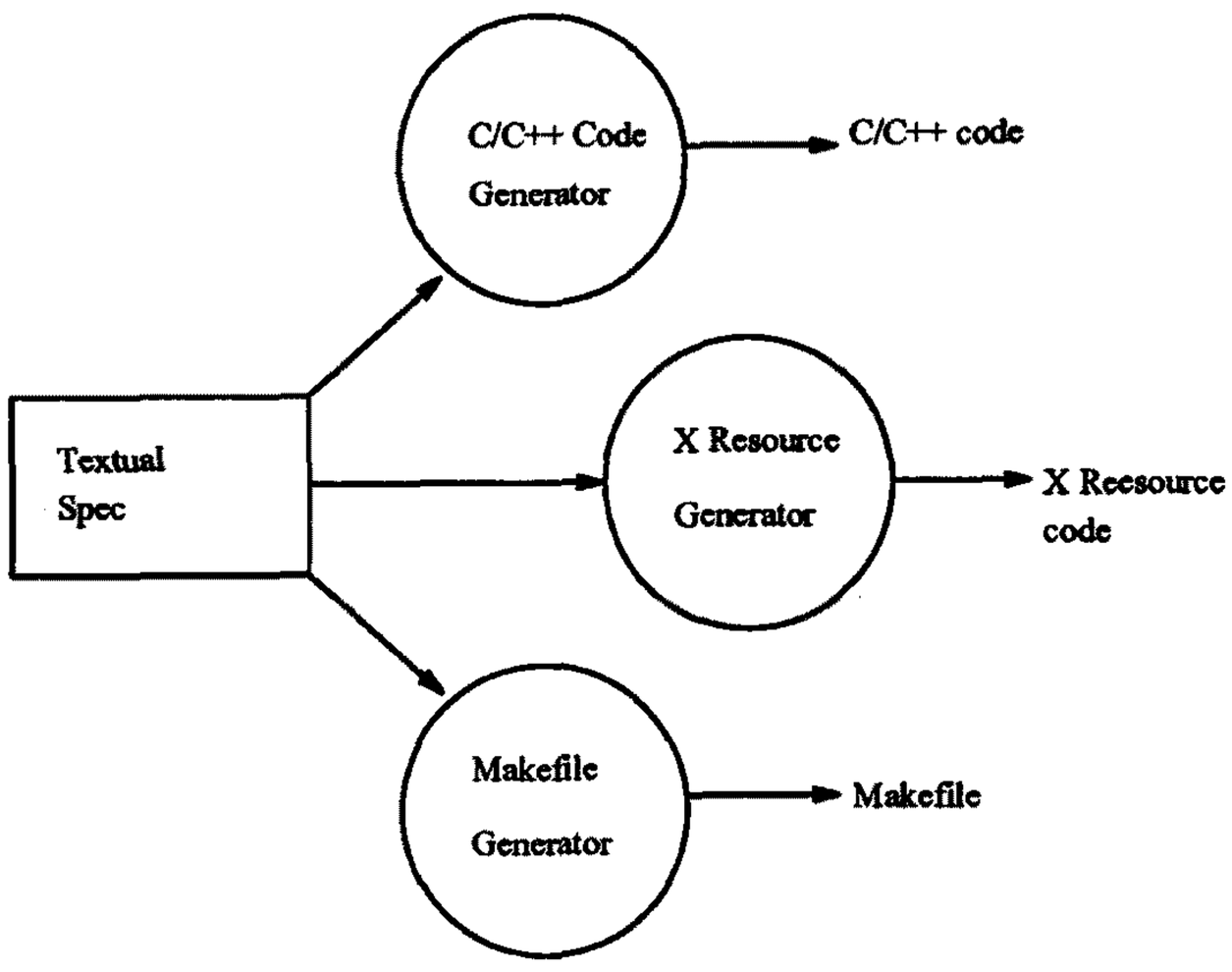


[그림 3-3-10] GUI Mosaic의 명세 표현 모형

7. 원시코드 생성 모형

원시코드는 사용자가 원하는 최종의 산물로서 생성된 원시코드 자체가 컴파일 가능하여야 하며, C 또는 C++ 코드, X 자원 코드와 Makefile을 생성한다. C 또는 C++ 코드는 프로젝트 전체에 하나만 존재하는 main 프로그램, 여러개의 인터페이스의 구조를 각기 대표하는 구조(structure) 프로그램들, 인터페이스의 각 위짓의 행위와 관련된 프로시듀어(procedure) 프로그램들과 응용 프로그램들로 구성된다.

[그림 3-3-11]은 원시코드의 생성 모형을 보여 주며, [그림 3-3-12]는 C 또는 C++ 코드가 가지는 프로그램들의 유형을 보여준다.



[그림 3-3-11] GUI Mosaic에서 원시코드 생성 모형

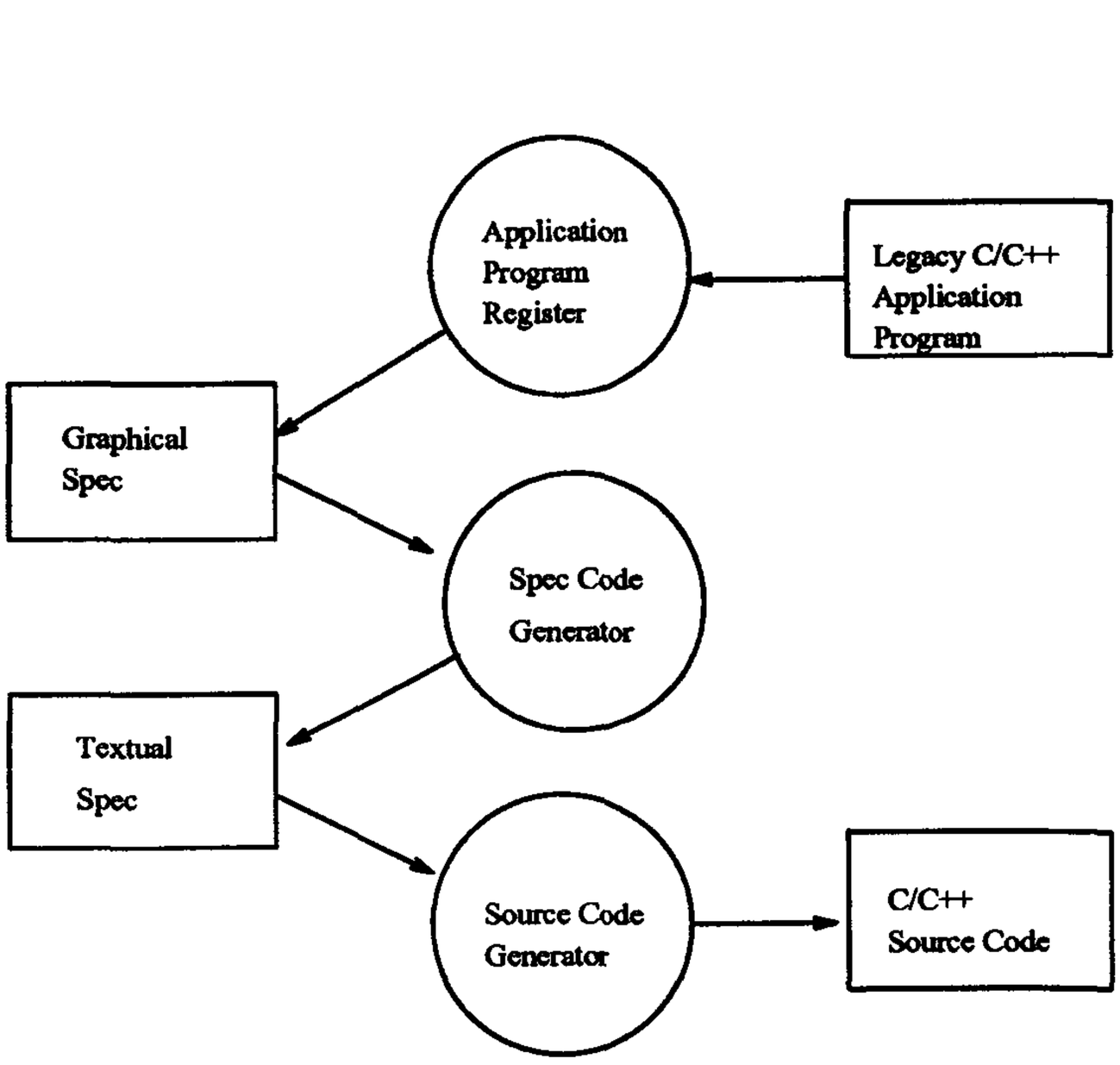
[그림 3-3-12] C/C++ 코드의 구성 요소

8. 기존 응용 프로그램과의 통합 모형

기존에 이미 개발되어진 소프트웨어의 응용 영역의 프로그램들은 응용에 대한 전문적인 지식을 가지고 있다는 점에서 매우 중요하다. 이러한 프로그램들을 다시 사용할 수 있다면 개발 생산성 측면에서 매우 긍정적인 결과를 낳을 것이다.

기존의 응용 프로그램을 재사용하기 위해서는 이들을 도구의 정보화하는 작업이 선행되어야 한다. 이 작업을 응용 프로그램 등록 작업(legacy application program registration)이라 하며, 이 작업의 결과는 도구에서 명세코드 생성과 원시코드 생성의 과정과 자연스럽게 연결되어 재사용된다.

[그림 3-3-13]은 기존 응용 프로그램의 도구 안으로의 통합 모형을 보여준다.



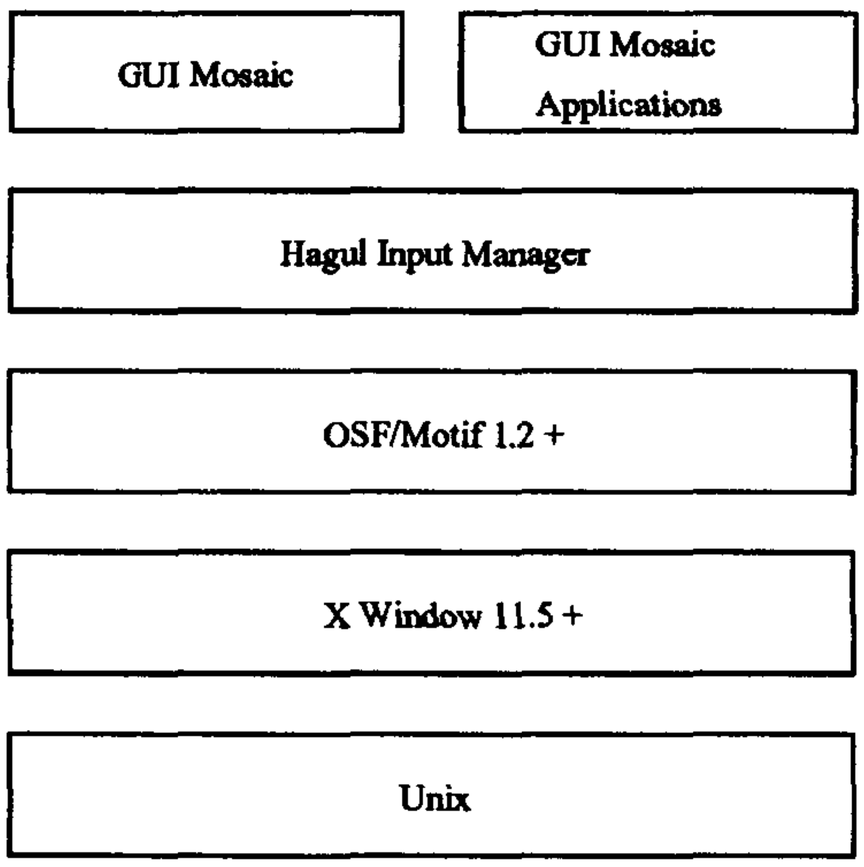
[그림 3-3-13] GUI Mosaic에서 기존 응용 프로그램과의 통합 모형

제 4 절 GUI Mosaic의 개발 명세

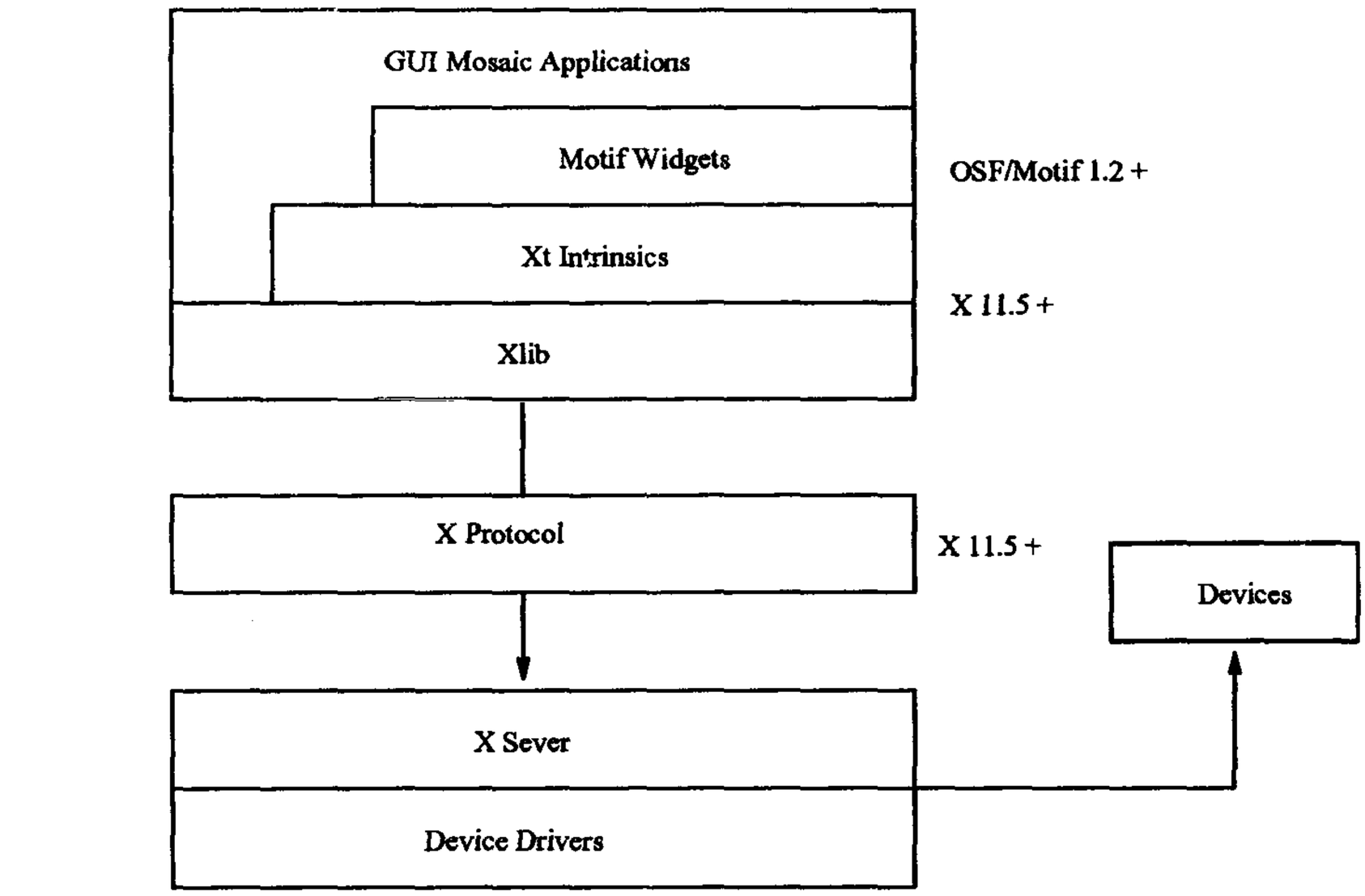
1. 환경

가. 목적 환경

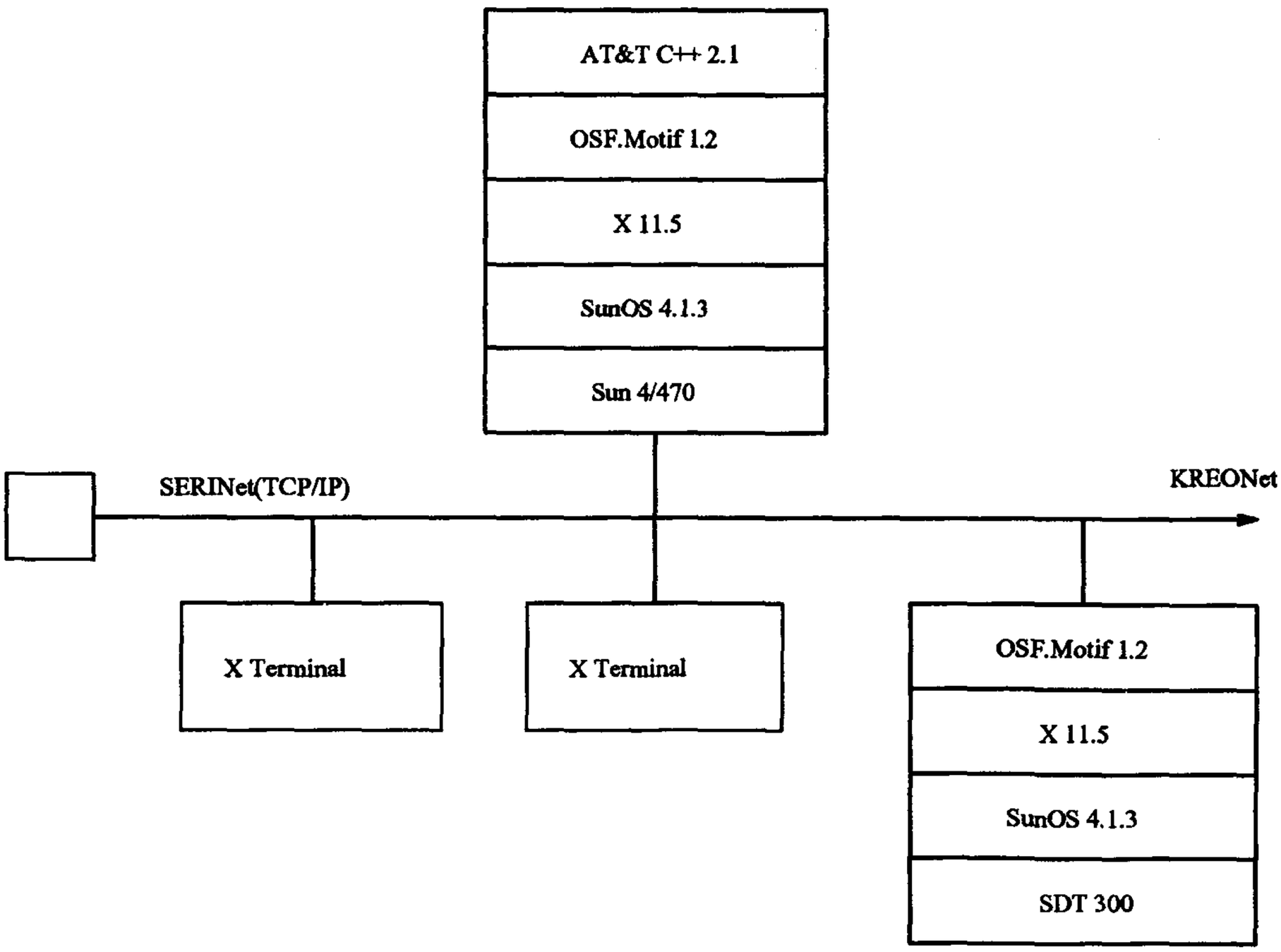
GUI Mosaic은 SunOS 4.1.3, X 11R5, Motif 1.2에서 운영되며, 이 환경에 맞는 응용 수행 코드를 만든다. [그림 3-4-1]은 GUI Mosaic의 운영 환경 및 생성된 소프트웨어의 운영 환경이기도 한 시스템 환경을 보여준다. GUI Mosaic에서 만들어진 응용 소프트웨어의 작동 방식은 [그림 3-4-2]와 같다.



[그림 3-4-1] GUI Mosaic의 목적 환경



[그림 3-4-2] GUI Mosaic Application의 작동 방식



[그림 3-4-3] 개발 환경 구성

나. 개발 환경

1) 개발 H/W 및 S/W 환경

개발은 Sun 4/470을 주 기계로하고 SERINet에 연결하여, 두대의 X 터미날과 한대의 SDT 300과 통신하여 개발하는 환경이다. [그림 4-3-3]은 개발환경을 보여주는 그림이다.

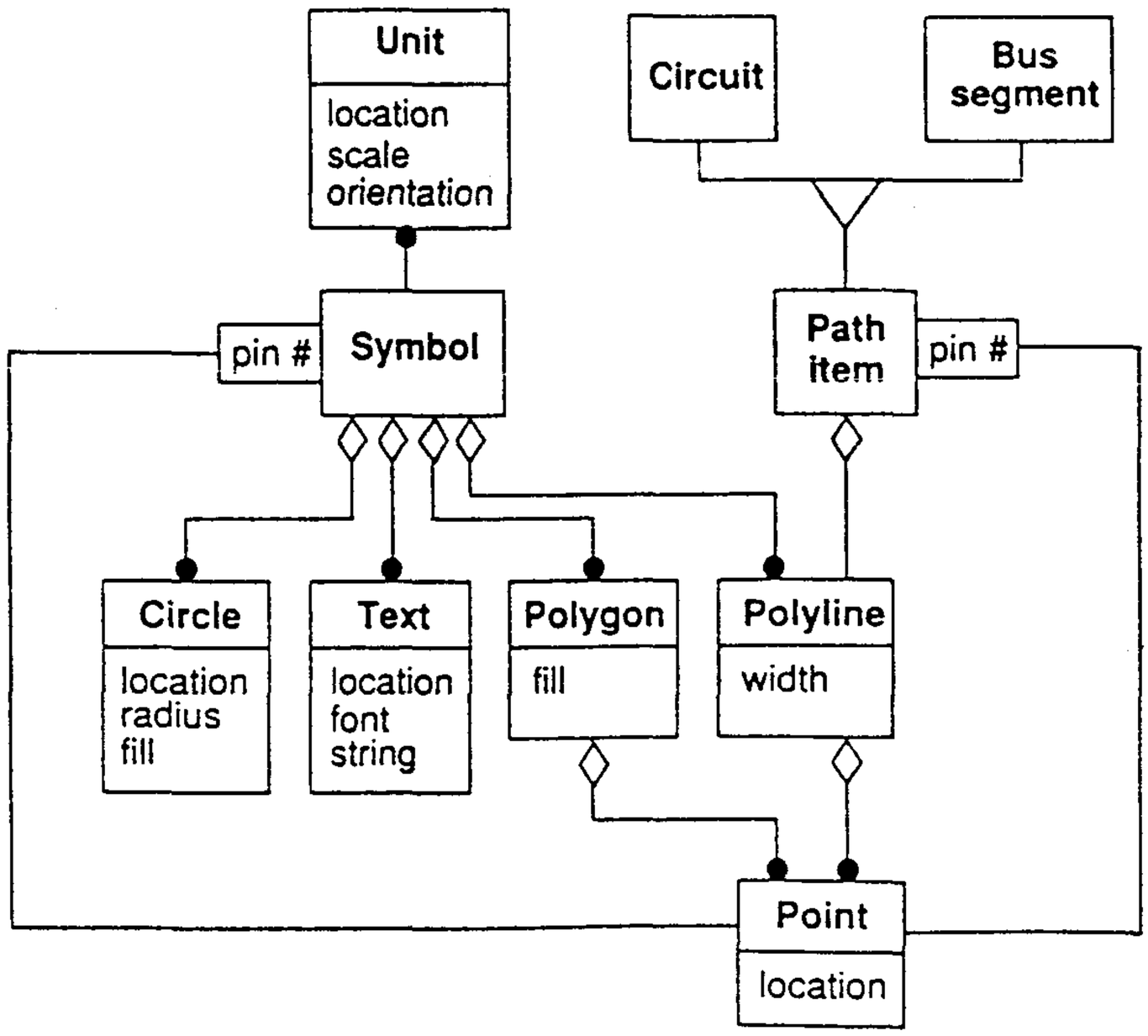
2) 개발 방법론

GUI Mosaic의 개발은 객체지향적인 방법의 적용을 기본으로 한다. 근간이 되는 방법은 Rumbaugh의 OMT 방법을 사용한다. 즉, 시스템을 자료(data)의 관점, 기능(function)의 관점, 상태(state)의 관점에서 보고, 각기 객체도(object diagram), 자료흐름도(dataflow diagram), 상태도(state diagram)로 표현한다. 표현 방법은 참고문헌 [10]의 양식을 따른다. [그림 3-4-4]는 객체도를 보여주고, [그림 3-4-5]는 자료흐름도를, [그림 3-4-6]은 상태도를 보여주는 예이다.

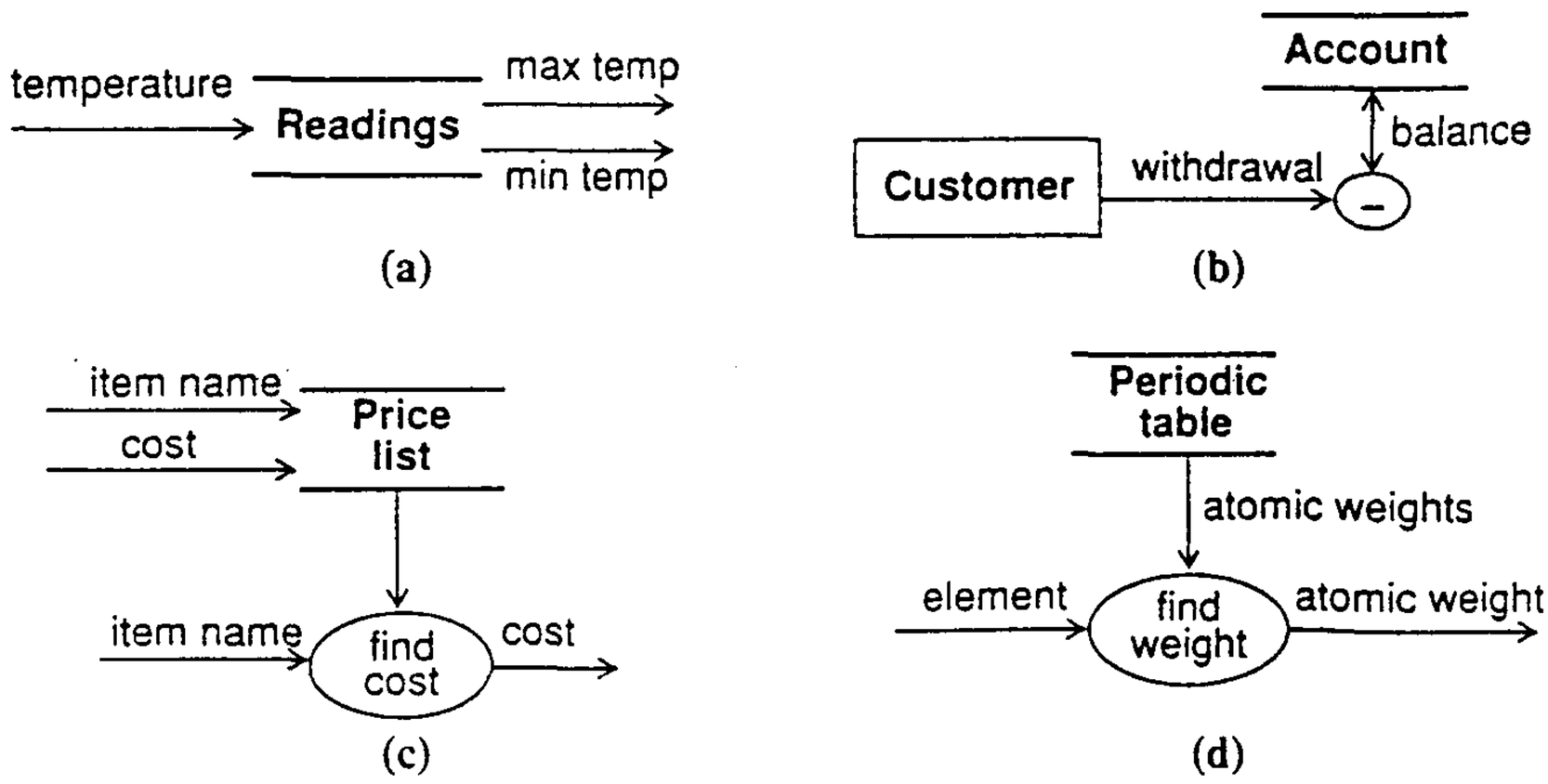
2. 도구의 구성

가. 도구의 기능적(functional) 구성

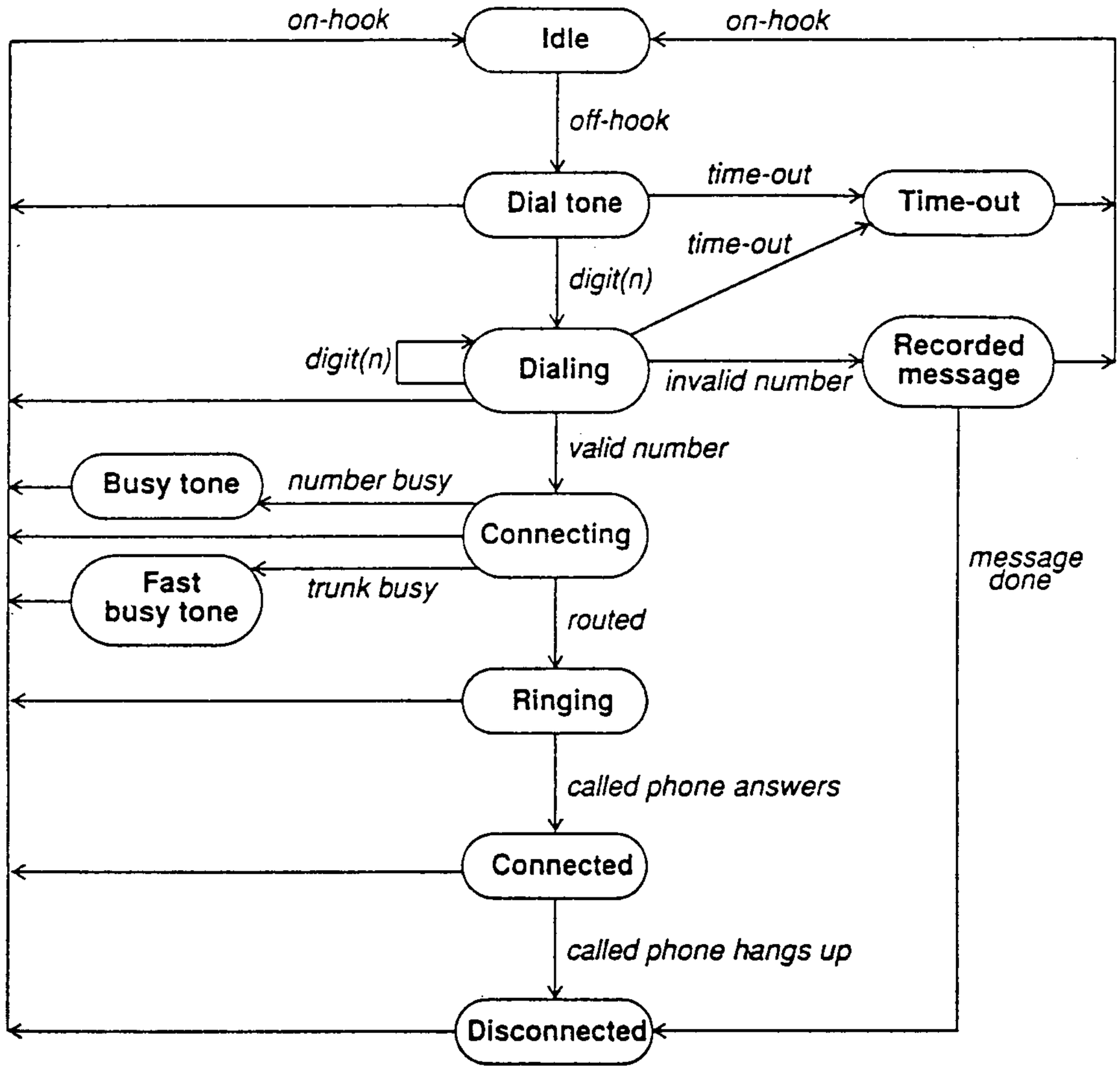
GUI Mosaic의 기능은 주요한 8가지의 기능으로 나눌 수 있다. 다음은 이러한 기능들이며, [그림 3-4-7]은 제어적 성격의 작업 절차 관리를 제외한 7가지의 기능의 구성을 보여준다.



[그림 3-4-4] OMT 형식의 객체도 예



[그림 3-4-5] OMT 형식의 자료흐름도 예



[그림 3-4-6] OMT 형식의 상태도 예

- 작업 절차 관리
- 프로젝트 명세화
- 프로젝트 이미지 명세의 번역
- 프로젝트 그래픽 이미지 명세의 텍스트 명세로의 변환
- 프로젝트 텍스트 명세의 그래픽 이미지 명세로의 변환
- 프로젝트 그래픽 이미지 명세로 부터 목적 원시코드의 생성
- 정보 저장소와의 접속

- 목적 원시 코드의 컴파일

작업 절차의 관리는 사용자의 작업 순서를 제어함을 목적으로 하는 제어 기능으로서 각각의 작업을 통합해 주는 역할을 수행한다. 작업의 시작은 기존 프로젝트 명세의 유무에 따라 구분되며, 중간 과정은 주로 프로젝트 그래픽 이미지 명세의 유무에 따라 구분되고, 컴파일 과정은 원시 코드의 유무에 의존한다.

프로젝트 명세화는 사용자의 마음속에 있는 프로젝트를 문서를 통하여 구체화하여 프로젝트 이미지를 만드는 과정이다. 이 과정은 시각적이며 대화식으로 발생한다.

프로젝트 이미지 명세의 번역은 사용자에게 컴퓨터 메모리 안의 프로젝트 형상을 사용자의 화면에 실현하는 기능이다. 사용자는 작업 과정에서 이 기능의 도움으로 자신이 하고 있는 작업의 결과를 검증할 수 있다.

프로젝트 이미지 명세의 텍스트 명세로의 변환은 일종의 코드 생성으로 메모리의 정보를 화일 정보화하는 작업으로, 도구의 명령으로는 주로 "저장"이라는 표현이 사용된다. 경우에 따라서는 이 명세 코드를 편집함으로서 프로젝트의 명세를 수정할 수도 있다. 그러나 이러한 수정 방법은 바람직하지 못한 것으로 피하는 것이 좋다.

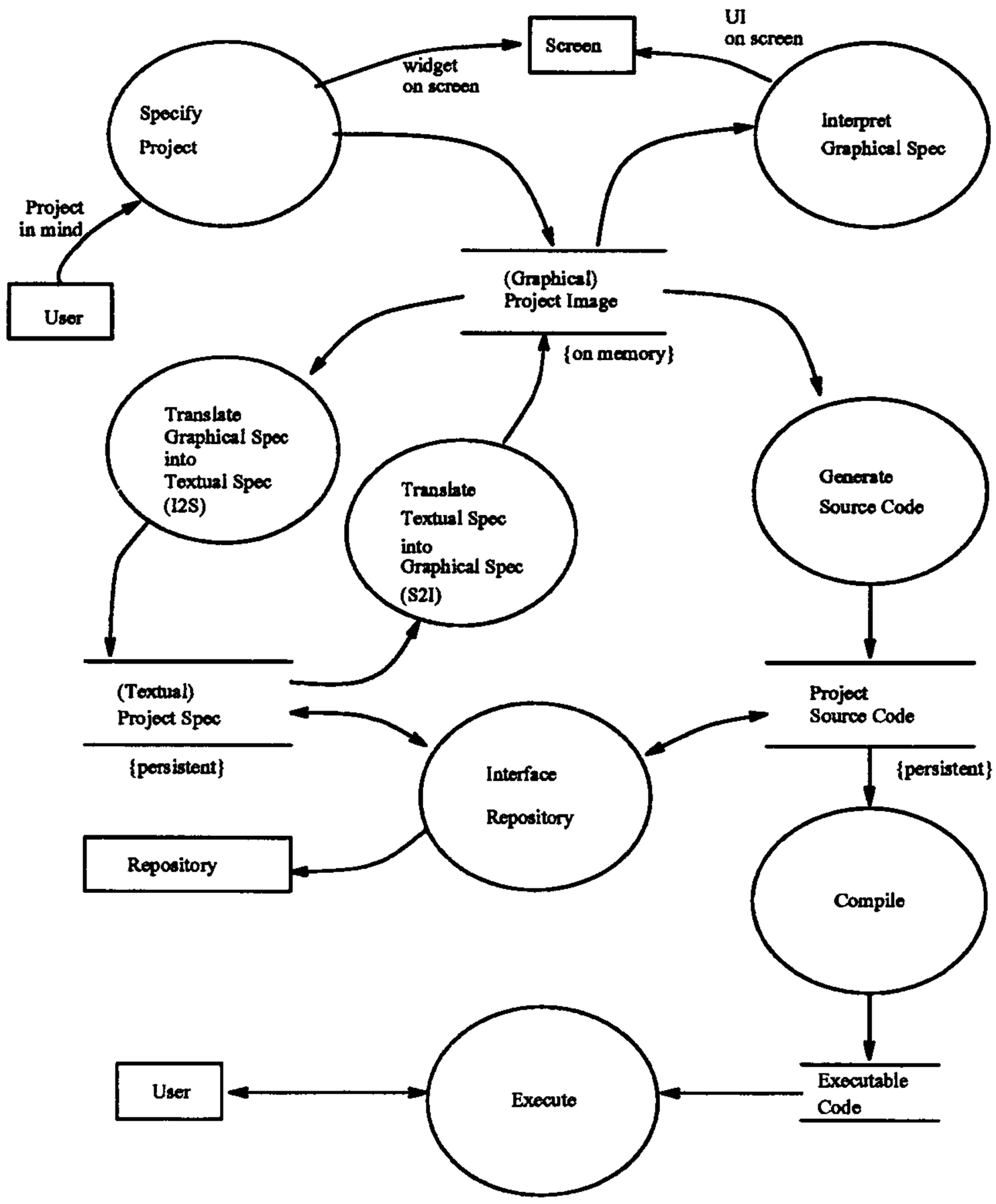
프로젝트 텍스트 명세의 그래픽 이미지 명세로의 변환은 화일의 정보를 메모리 정보화하는 기능으로, 도구의 명령으로는 주로 "읽어오기"라는 표현이 사용된다. 이 기능의 주요 목적은 기존의 프로젝트 명세에 대한 변경을 가하기 위한 준비이다.

프로젝트 그래픽 이미지 명세로 부터 목적 원시 코드의 생성은 메모리의 이미지를 컴파일 가능한 프로그래밍 언어인 C 또는 C++로 바꾸는 기능이다. 이 기능에서는 C 또는 C++ 원시코드 뿐만 아니라 필요에 따라서는 X 자원 화일을 만들어 내며, 목적 언어에 맞는 Make 화일도 만들어 낸다. 즉 바로 컴파일이 가능하도록

환경을 만들어 낸다.

목적 원시코드의 컴파일은 수행 코드를 만드는 기능으로 컴파일러는 기존에 시장에서 제공되는 것을 통합하여 이용한다.

마지막으로 정보 저장소와의 접속은 프로젝트의 텍스트 명세와 원시 코드를 저장소 정보화하거나 또는 반대의 기능을 수행한다.

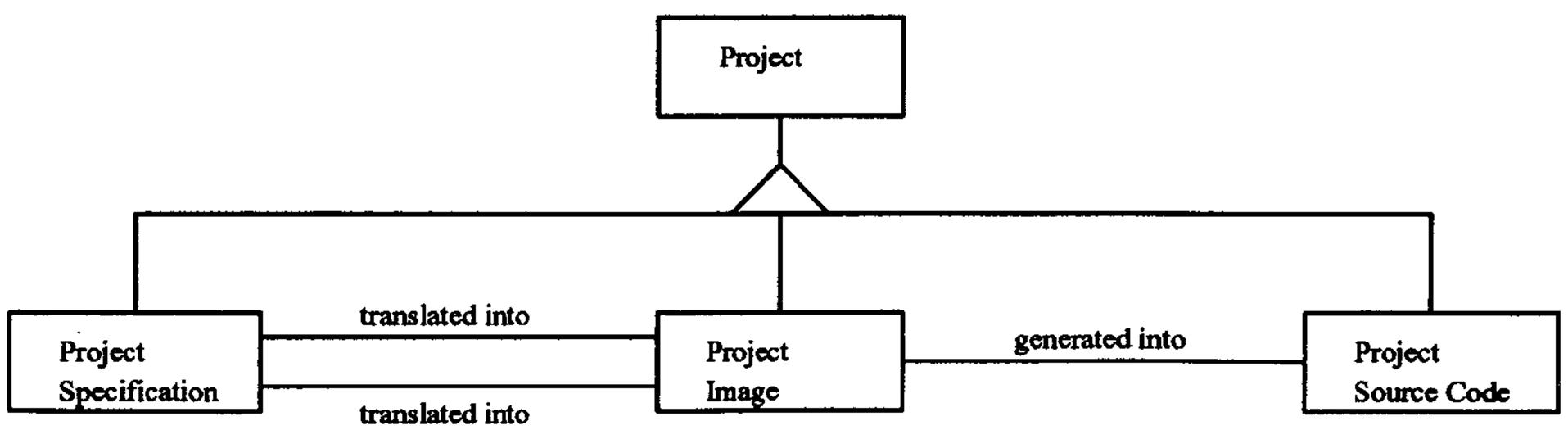


[그림 3-4-7] 도구의 기능적 구성

나. 도구의 자료(data)적 구성

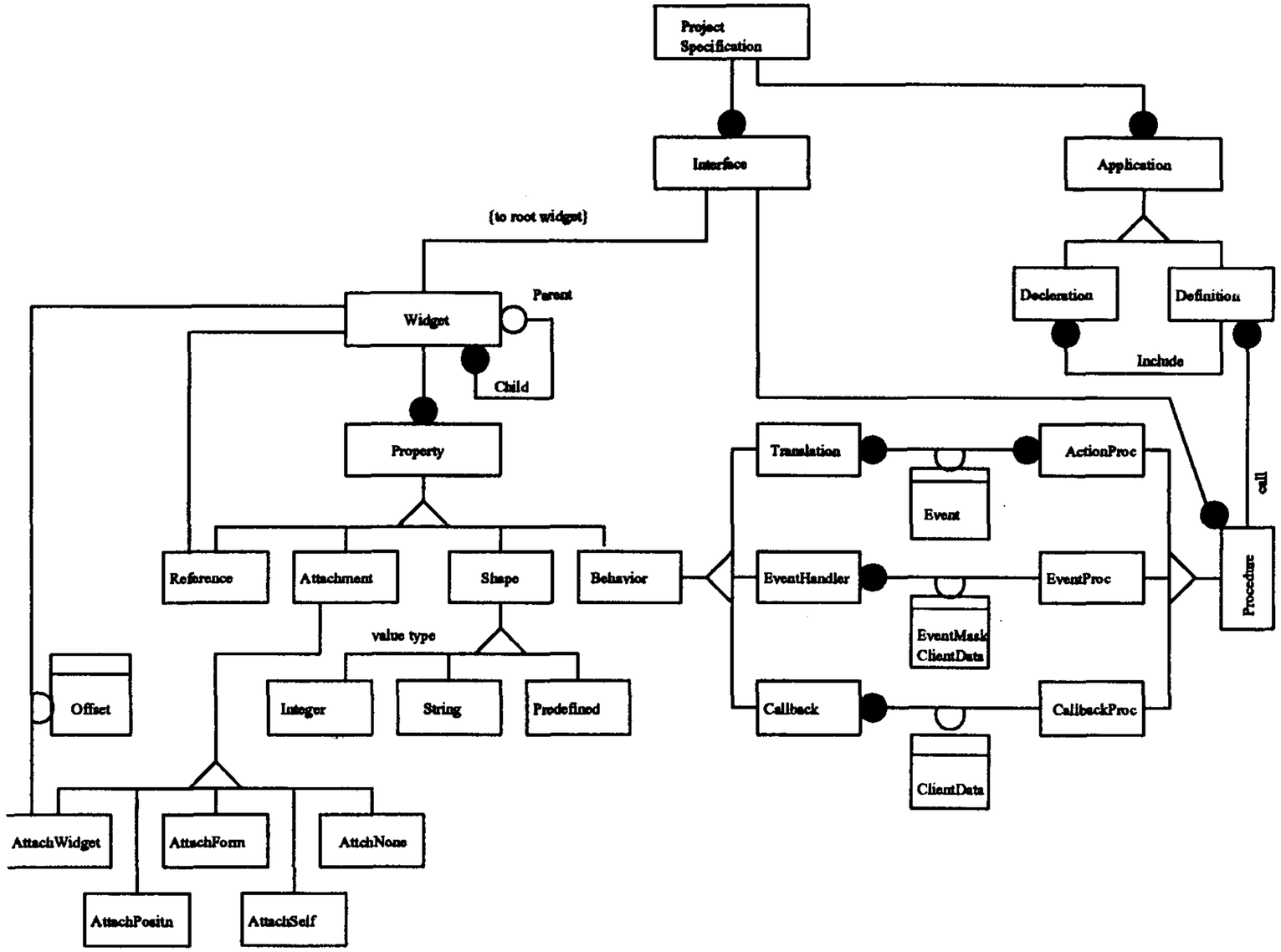
GUI Mosaic은 한번에 하나의 프로젝트에 대한 작업을 수행 할 수 있으며, 프로젝트의 정보는 [그림 3-4-7]에서 보여주듯이 내용면에서는 동일하나 표현 방법 또는 표현 매체가 다른 네가지 종류, 텍스트 프로젝트 명세, 그래픽 프로젝트 이미지, 프로젝트 원시 코드, 그리고 수행 코드가 있다. 그러나 수행 코드는 컴파일러를 고려한 원시코드에 완전히 의존적이며, 원시코드 보다 우수한 이식성으로 관리 대상에서 제외하므로 실제적으로 관리하여야하는 프로젝트 정보의 종류는 세가지이다.

[그림 3-4-8]은 프로젝트 정보의 종류를 보여주는 그림으로, 이 그림에서 프로젝트 명세(Project Specification)는 프로젝트 정보를 화일에 유지하기 위한 정형적인 명세어로 표현된 코드이며, 프로젝트 이미지(Project Image)는 프로젝트 정보를 도구에서 다루기 위해 메모리의 정보로 표현된 것이며, 프로젝트 원시코드(Project Source Code)는 목적 프로그래밍 언어로 표현된 프로젝트 정보이다.

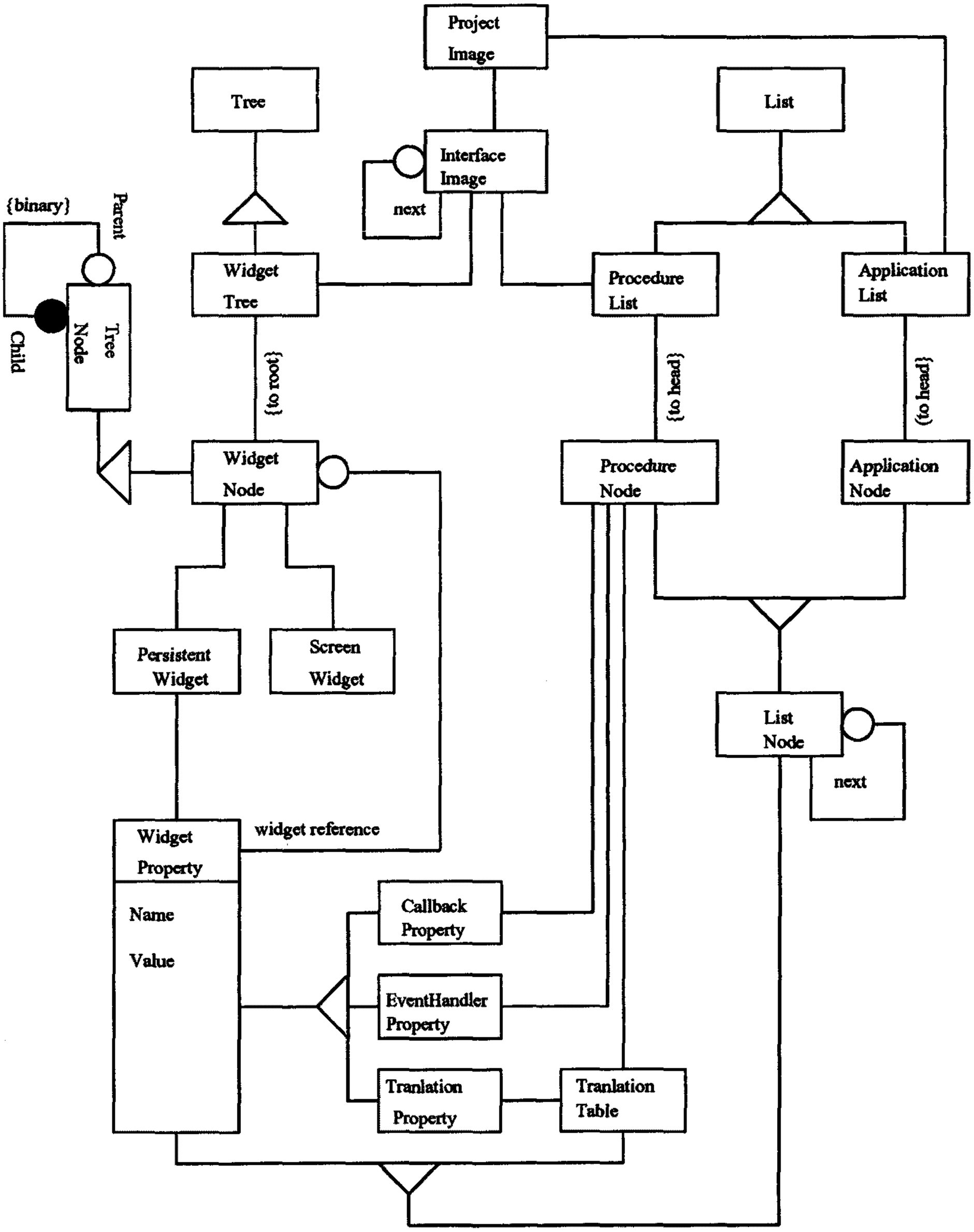


[그림 3-4-8] 도구의 자료적 구성

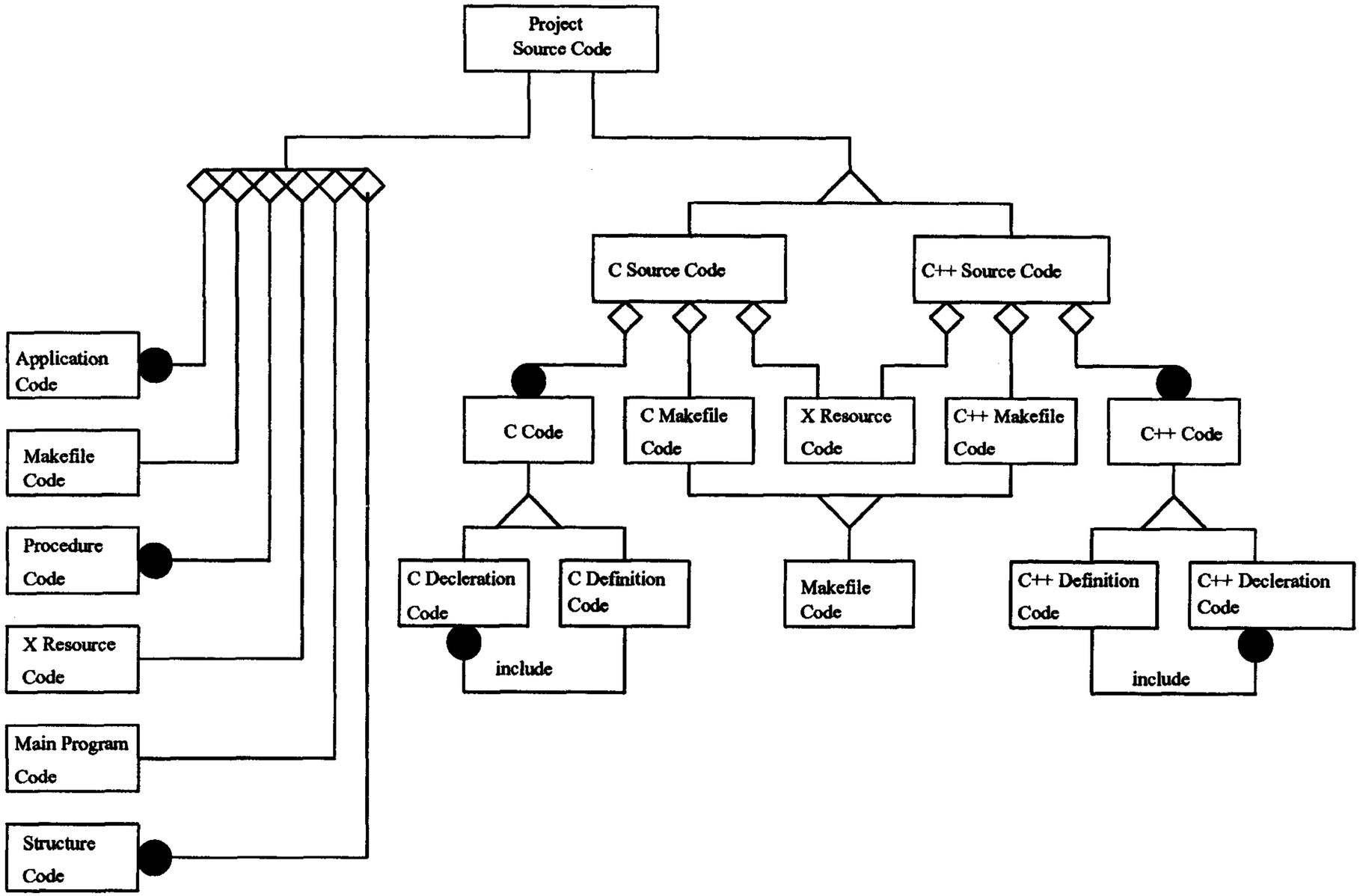
[그림 3-4-8]에 나타나는 각 프로젝트 정보의 구성은 각기 [그림 3-4-9,10 , 11]과 같다.



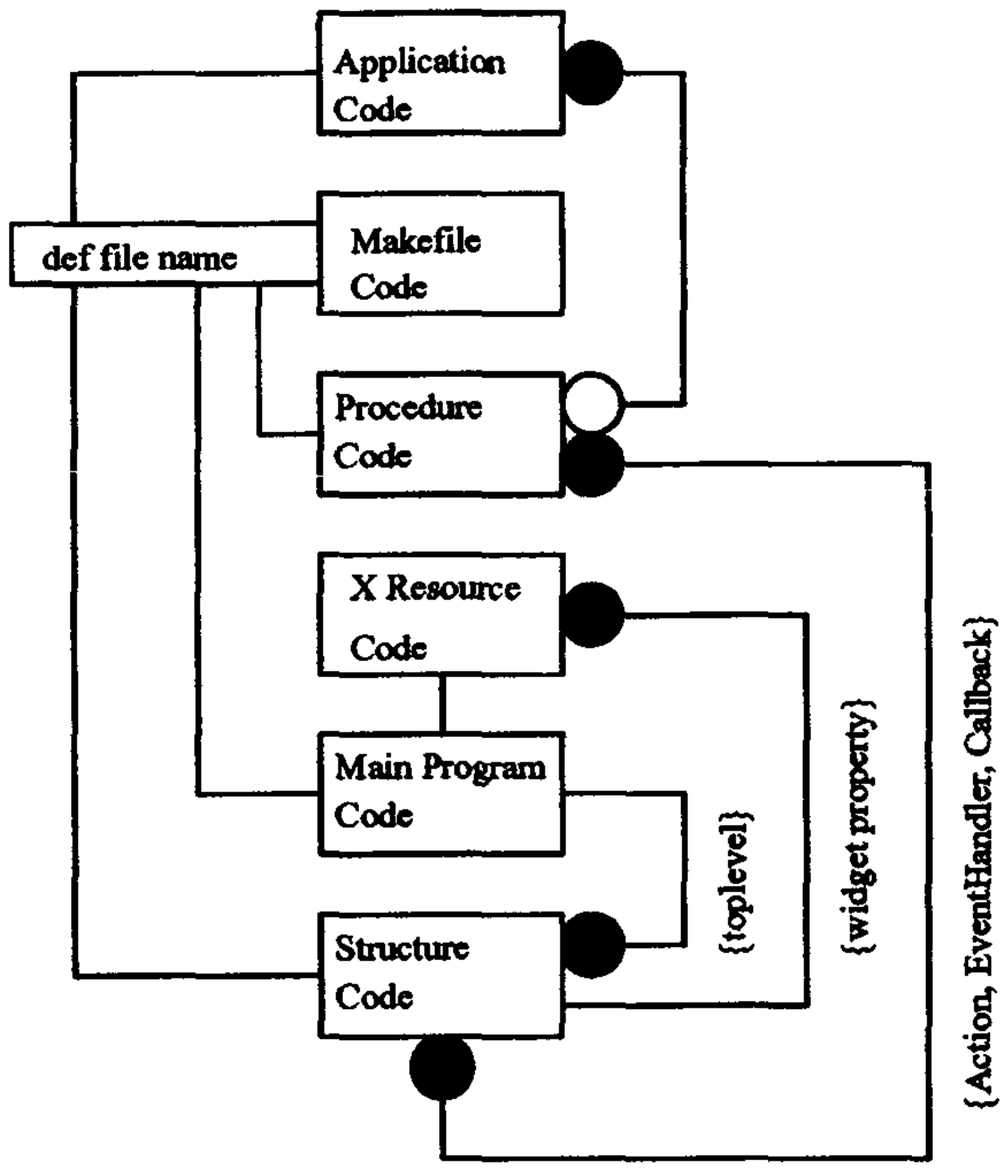
[그림 3-4-9] 텍스트 프로젝트 명세의 구성



[그림 3-4-10] 그래픽 프로젝트 이미지의 구성



[그림 3-4-11] 프로젝트 원시 코드의 구성



[그림 3-4-12] 프로젝트 원시코드 요소 간의 관계

다. 도구의 상태적(state) 구성

GUI Mosaic은 도구와 사용자 사이에 극히 대화적인 방법을 사용하는 시스템으로서 상태의 천이가 사용자의 요구에 따라 다양하게 발생한다. [그림 3-4-7]에 나타난 기능들은 사용자의 요구에 따라 자신의 기능을 수행한다. 이러한 상태의 천이는 사용자가 발생시킨 사건이나 수행의 결과에 따른 사건으로 대별된다. GUI Mosaic의 개괄적인 상태의 천이는 [그림 3-4-13]과 같다.

GUI Mosaic 프로그램을 수행시키면 시스템을 초기화 시킨후 초기 대기(Idle) 상태인 Work Procedure Managing 상태에 들어간다. 이 상태는 사용자로 부터의 사건 발생을 대기하는 상태이며, 사건이 요구하는 기능이 현재 상태에서 수행 가능한 절차인 지를 검증한다. 합당한 요구이면 기능의 수행 상태로 천이하고 수행

이 끝나면 원래의 상태로 다시 천이한다. 아니면 합당한 절차가 아님을 알리는 메시지를 보이고 Message Displayed 상태로 천이하고 사용자의 확인을 받고 원래의 상태로 천이한다.

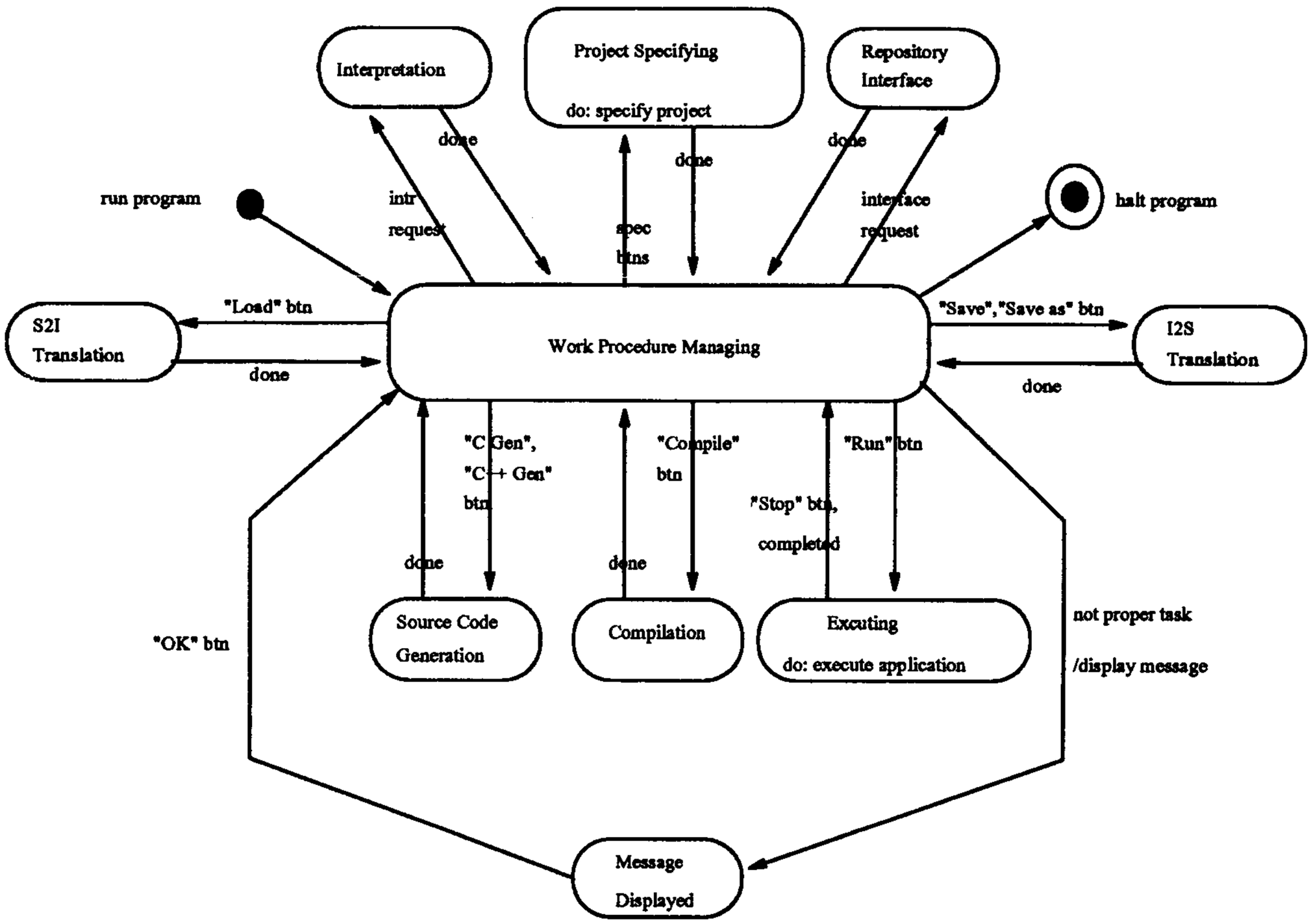
Project Specifying 상태는 프로젝트 명세화를 수행하는 모든 상태를 대표하는 상태이며, I2S/S2I Translation은 명세의 변환 상태를 나타내며, Compilation은 원시 코드의 변환 상태를, Source Code Generation은 원시 코드의 생성 상태를, Executing은 생성된 응용 시스템의 수행 상태를 나타내며, 이 상태의 내부 상태의 구성은 응용마다 다르기 때문에 더이상 나누어 표현할 수 없는 특징이 있다.

그 외에도 X Window 시스템 및 Motif 환경에서 기본적으로 제공하는 기능에 의한 시스템의 상태 천이가 있다. 예를 들면, 시스템의 아이콘화, 위치 변경, 크기 변경 등과 같은 기능들에 의한 외적 상태 천이가 존재한다. 그러나 이러한 상태들은 시스템 외적이므로 고려에 넣지 않는다.

3. 작업 절차 관리기

작업 절차 관리기는 사용자의 작업이 합당한 절차를 통하여 이루어지도록 하는 안내자 기능을 수행하는 모듈이다. 따라서 [그림 3-4-7]에 나타나는 모든 기능의 수행은 이 관리기의 안내를 받는다. 즉, 해당 기능을 요구하는 요구 사건이 발생하면 작업 절차 관리기에서는 해당 기능의 수행이 가능한 상황인 지를 검사하고 합당하면 해당 기능을 수행시키고, 아니면 불가능하다는 메시지를 보여주는 사건을 발생시켜 메시지를 화면에 보여주는 기능을 수행시킨다. 이러한 검사의 과정은 요구하는 기능과 현재 자료의 준비 상황에 의해 결정된다. 이러한 결정 과정은 [그림 3-4-7]의 기능적 구성도와 [그림 3-4-13]의 도구의 상태적 구성도

를 통하여 이해할 수 있다.



[그림 3-4-13] 도구의 상태적 구성

사용자가 “Specify Project” 기능 중에서 Project의 변경을 요구하는 사건이 발생하면 작업 절차 관리기는 Project Image가 존재하는 지를 확인한다. 존재하는 경우에는 요구된 기능을 수행할 수 있는 허가를 내주며, 반대의 경우에는 불허를 하며 메시지를 띄운다.

사용자가 “Interpret Graphical Spec” 기능을 요구하면 관리기는 Project Image의 화면 요소가 존재하는 지를 확인한다. 존재하면 허가를 하고 반대의 경우는 불허하며 메시를 띄운다.

사용자가 “Translate Graphical Spec into Textual Spec” 기능을 요구하면 관

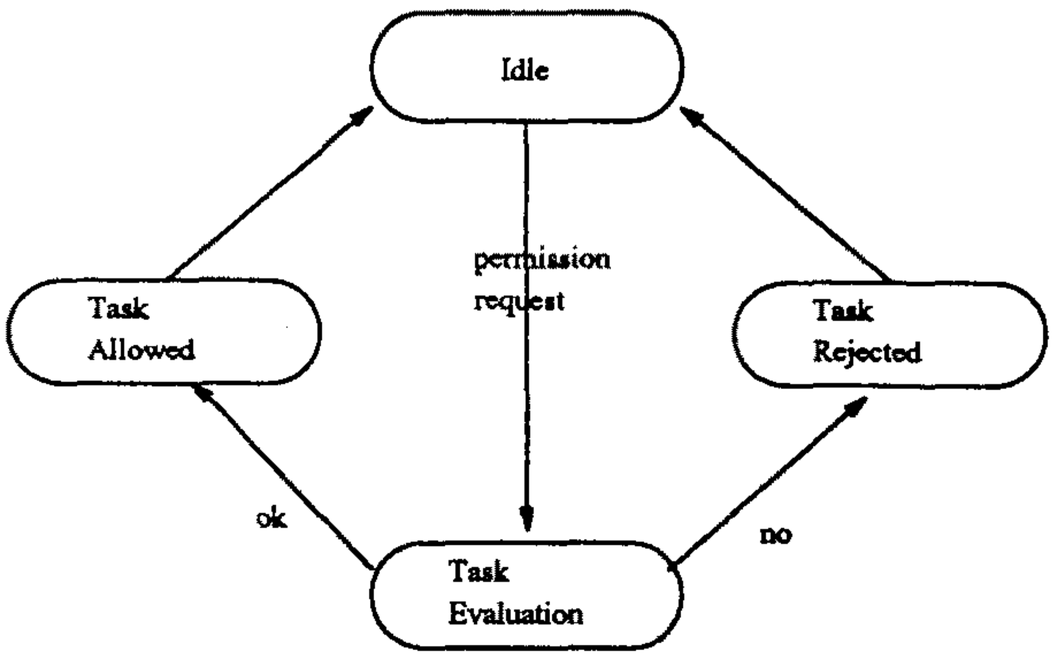
리기는 Project Image와 Project Spec의 존재를 확인한다. 우선 Project Image가 존재하지 않을 경우에는 불허를 하며, 존재할 경우에는 Project Spec의 변경에 대한 확인을 받고 허가한다. "Translate Textual Spec into Graphical Spec"의 경우는 반대의 확인 과정을 거쳐 허가 여부를 결정한다.

사용자가 "Generate Source Code"를 요구하면 Project Image의 존재를 확인하고, 없으면 불허하고, 있는 경우에는 Project Source Code의 존재를 확인하고 변경에 대한 확인을 받고 허가한다.

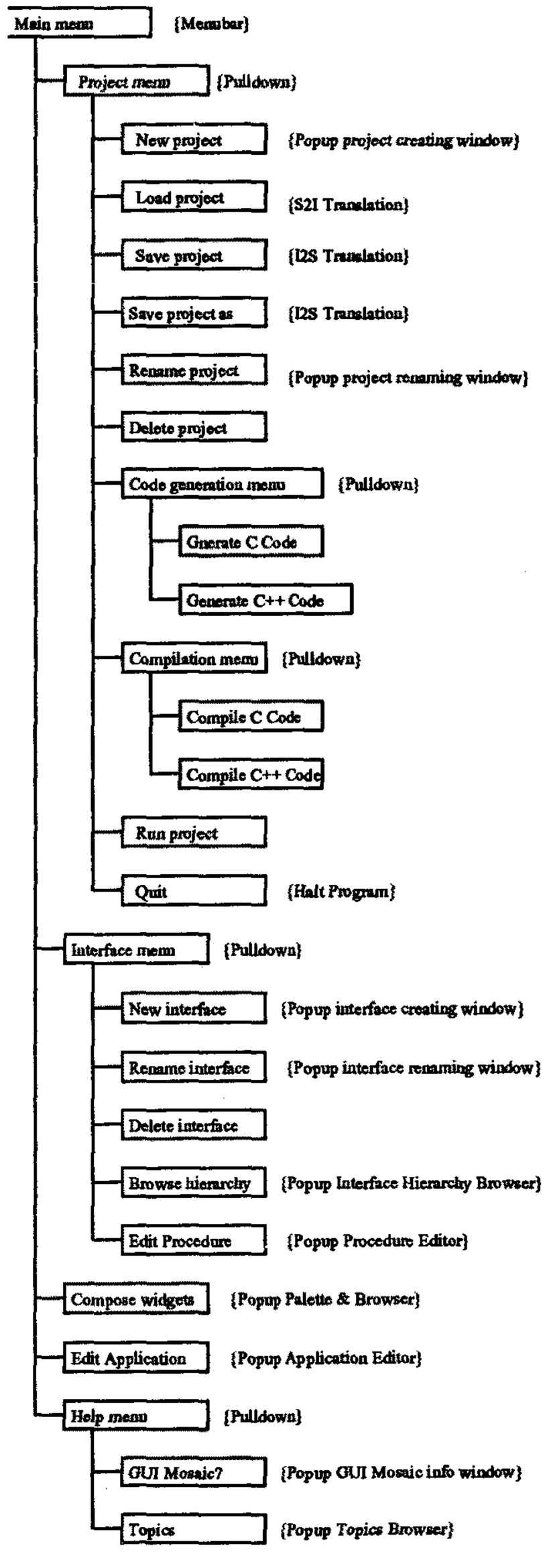
"Compile"의 경우는 Project Source Code의 존재를 확인하고, 있는 경우에 허가하며, "Execute"의 경우는 Executable Code를 확인하고 허가 또는 불허한다.

"Interface Repository"의 경우는 Project Spec과 Project Source Code의 존재 및 버젼의 일치성을 확인한 후 허가하며, 마지막으로 도구의 수행을 종료할 경우에는 Project Spec의 변경 유무를 확인하고 변경된 경우에는 Repository에 변경된 내용을 저장할 것인가를 확인하고 "Interface Repository" 기능 중에서 저장 기능을 수행하는 허가를 주고, 완료된 후에 프로그램을 종료한다.

[그림 3-4-14(a)]는 작업절차 관리기의 상태 천이를 보여주며,.[그림 3-4-14(b)]는 작업 절차 관리기가 제어하는 기능들을 중심으로 하는 GUI Mosaic의 주 메뉴 체계를 보여준다.



(a)



(b)

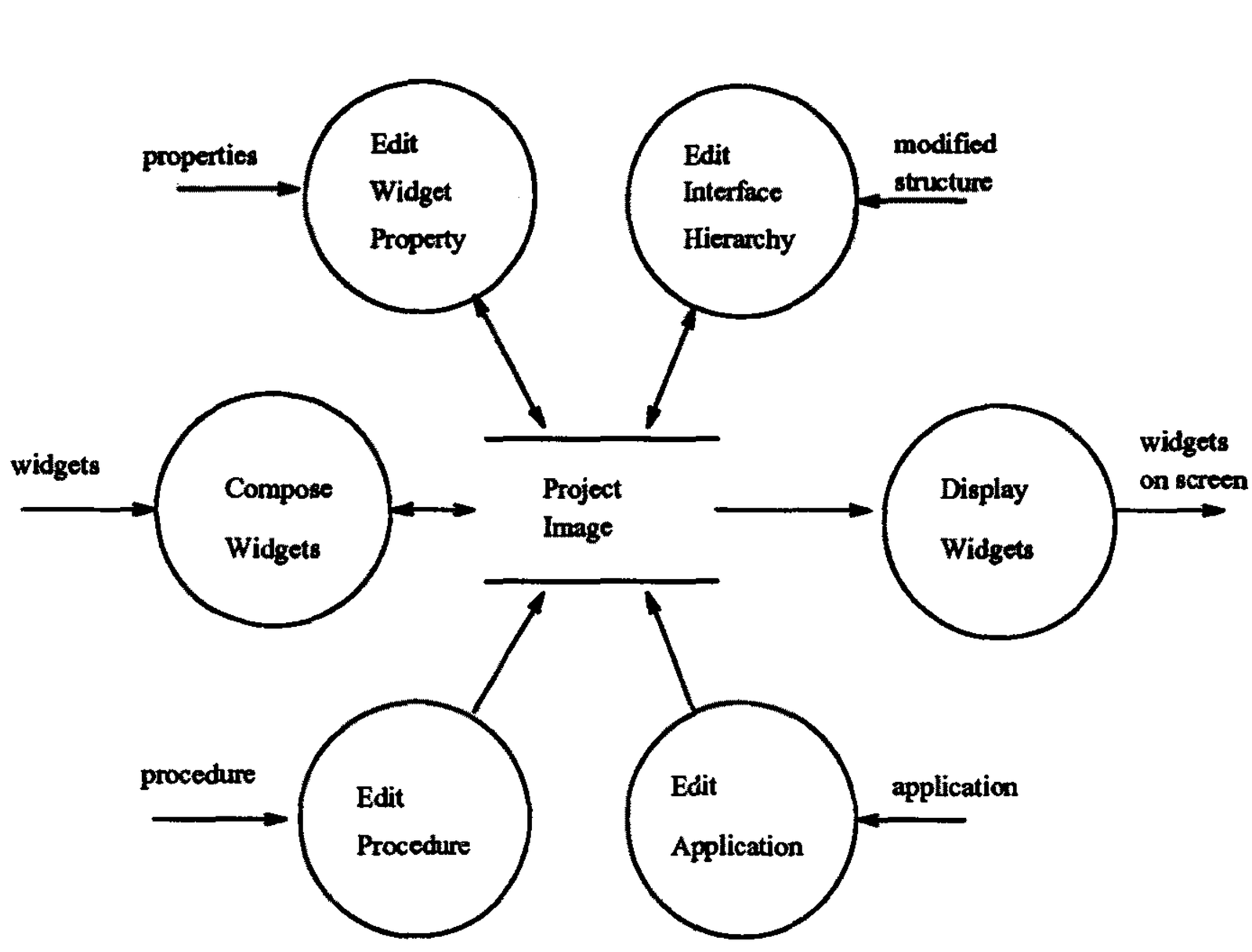
[그림 3-4-14] 작업 절차 관리기의 (a) 상태 천이 (b) 메뉴 상태 체계

4. 시각 사용자 명세 작업대

시각 사용자 명세 작업대는 [그림 3-4-7]의 "Specify Project"에 해당하는 기능을 수행하는 단위 도구의 이름으로 다음과 같은 기능으로 구성된다.

- 위짓을 합성하는 기능(Compose Widgets)
- 위짓의 특성을 편집하는 기능(Edit Widget Property)
- 위짓 합성체인 인터페이스를 편집하는 기능(Edit Interface Hierarchy)
- 위짓의 행위부인 프로시듀어를 편집하는 기능(Edit Procedure)
- 응용 프로그램을 편집하는 기능(Edit Application)
- 화면에 위짓을 구현하는 기능(Display Widgets)

[그림 3-4-15]는 시각 사용자 명세 작업대의 주요 기능을 보여준다.



[그림 3-4-15] 시각 사용자 명세 작업대의 주요 기능적 구성

위짓 합성은 하나의 위짓을 다른 하나의 위짓의 자식으로 하여 화면을 구성하는 작업으로 위짓의 트리를 구성하는 과정과 동시에 화면상의 위치와 크기를 결정한다. 위치 및 크기를 조정하는 작업은 화면에서 마우스를 이용하여 하게 되며 부모와 자식 사이의 관계가 합당한 지를 검증한다. 작업절차는 다음과 같다.

위짓 팔레트에서 위짓 클래스를 선정한다.

자신의 이름과 local인지 global인지를 입력한다.

이름이 이미 이미지 위짓 트리의 이름과 중복되면

오류 메시지를 내고 처음 상태로 돌아 간다.

고유하면 부모 위짓을 결정한다.

부모 위짓과의 관계가 합당한 가를 검증한다.

합당치 않으면 오류 메시지를 내고 종료한다.

합당하면 위짓의 위치 및 크기를 알리는 윤곽선을

현재의 커서 위치에 보여준다.

마우스를 이용하여 윤곽선의 위치 및 크기를 결정한다.

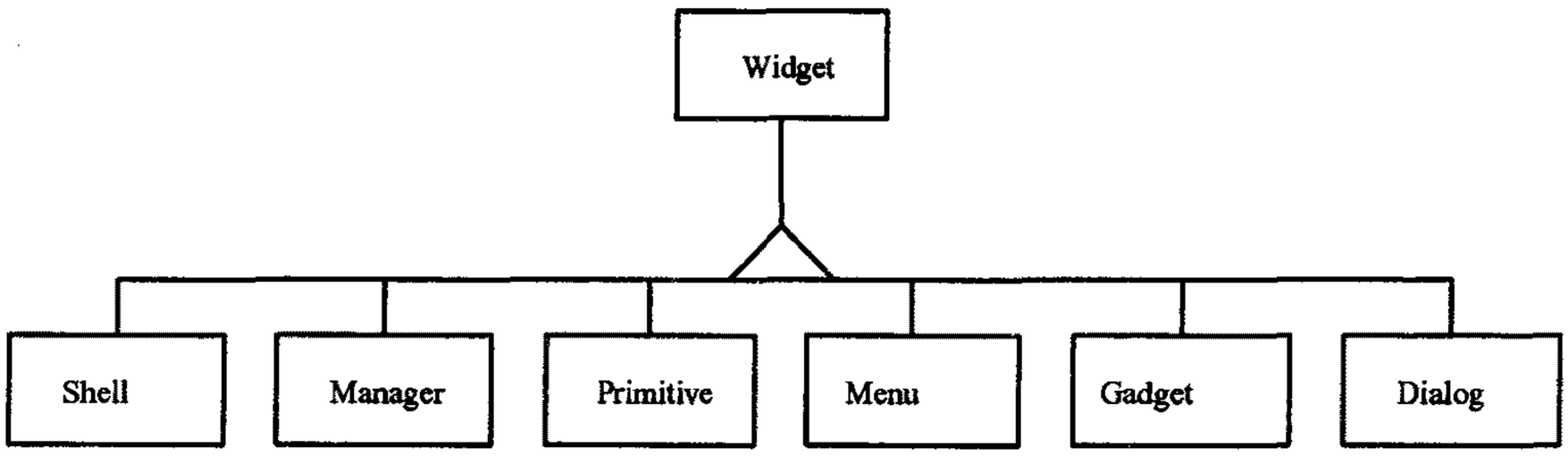
완성 버튼이 눌려지면

윤곽선 위짓을 프로젝트 이미지의 위짓 트리에 추가한다.

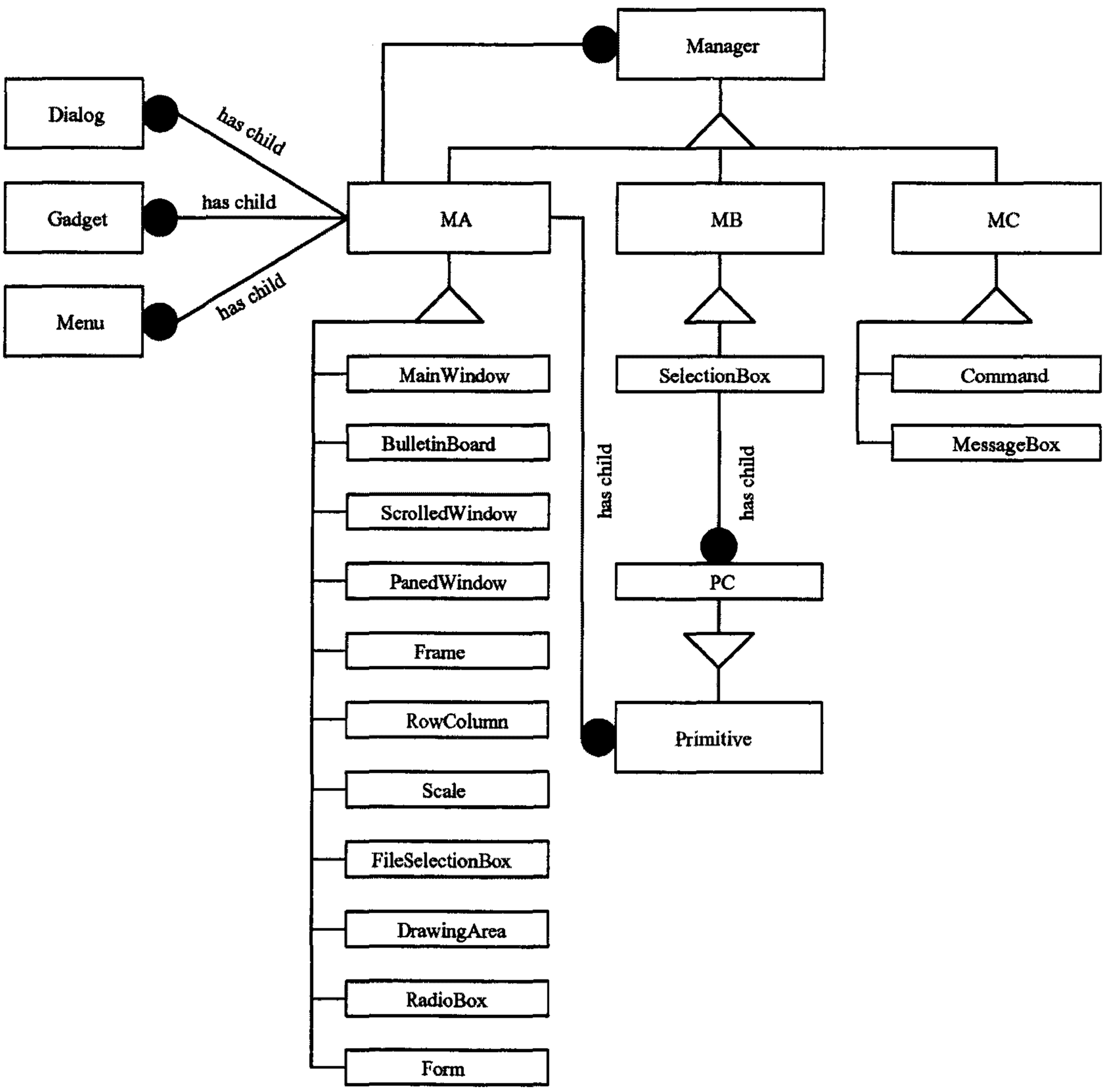
변경된 트리를 화면에 다시 구현하고,

종료한다.

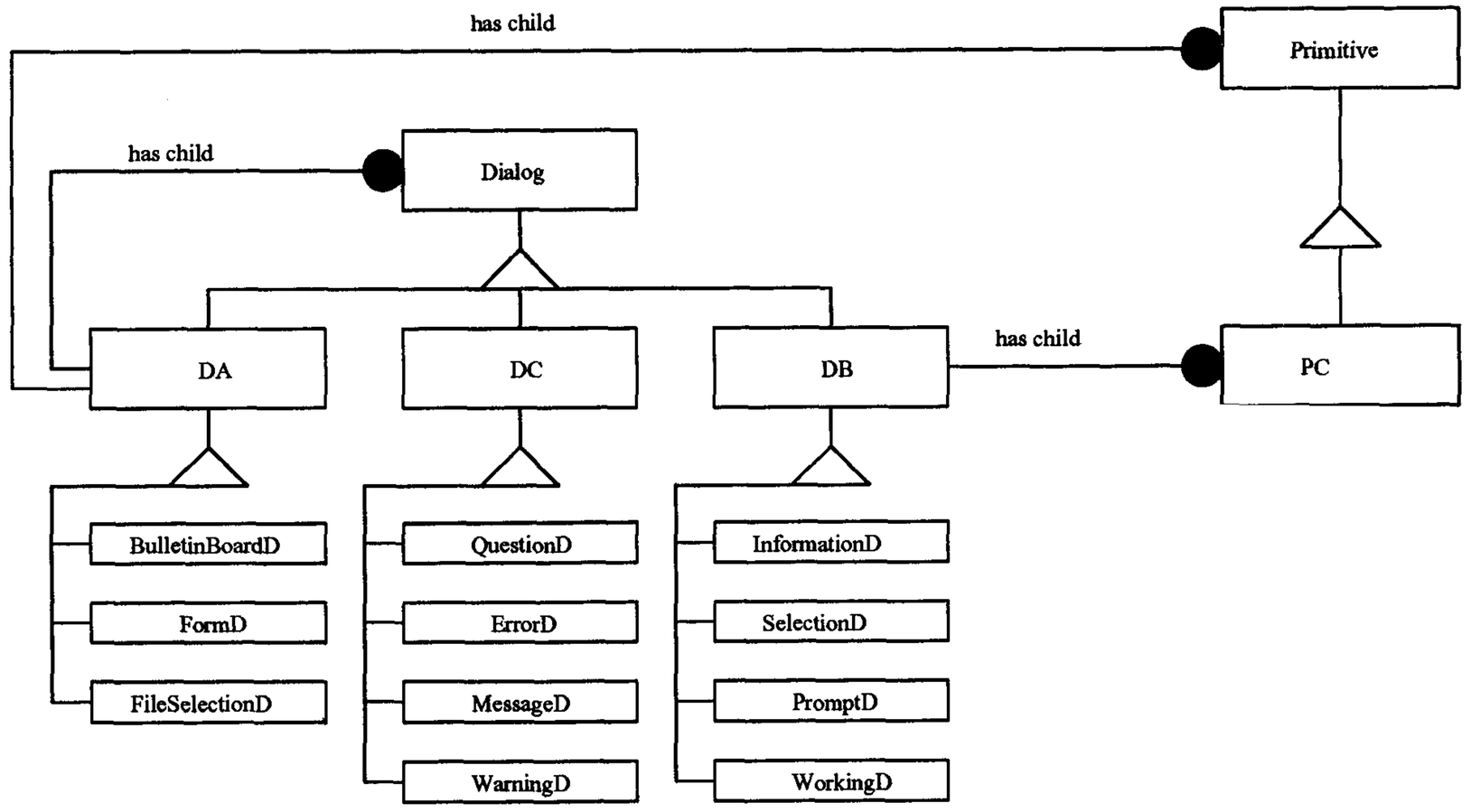
부모 위짓과의 관계가 합당한 지에 관한 검증은 [그림 3-4-16, 17, 18, 19, 20, 21, 22]에서 보여주는 규칙을 따른다[9].



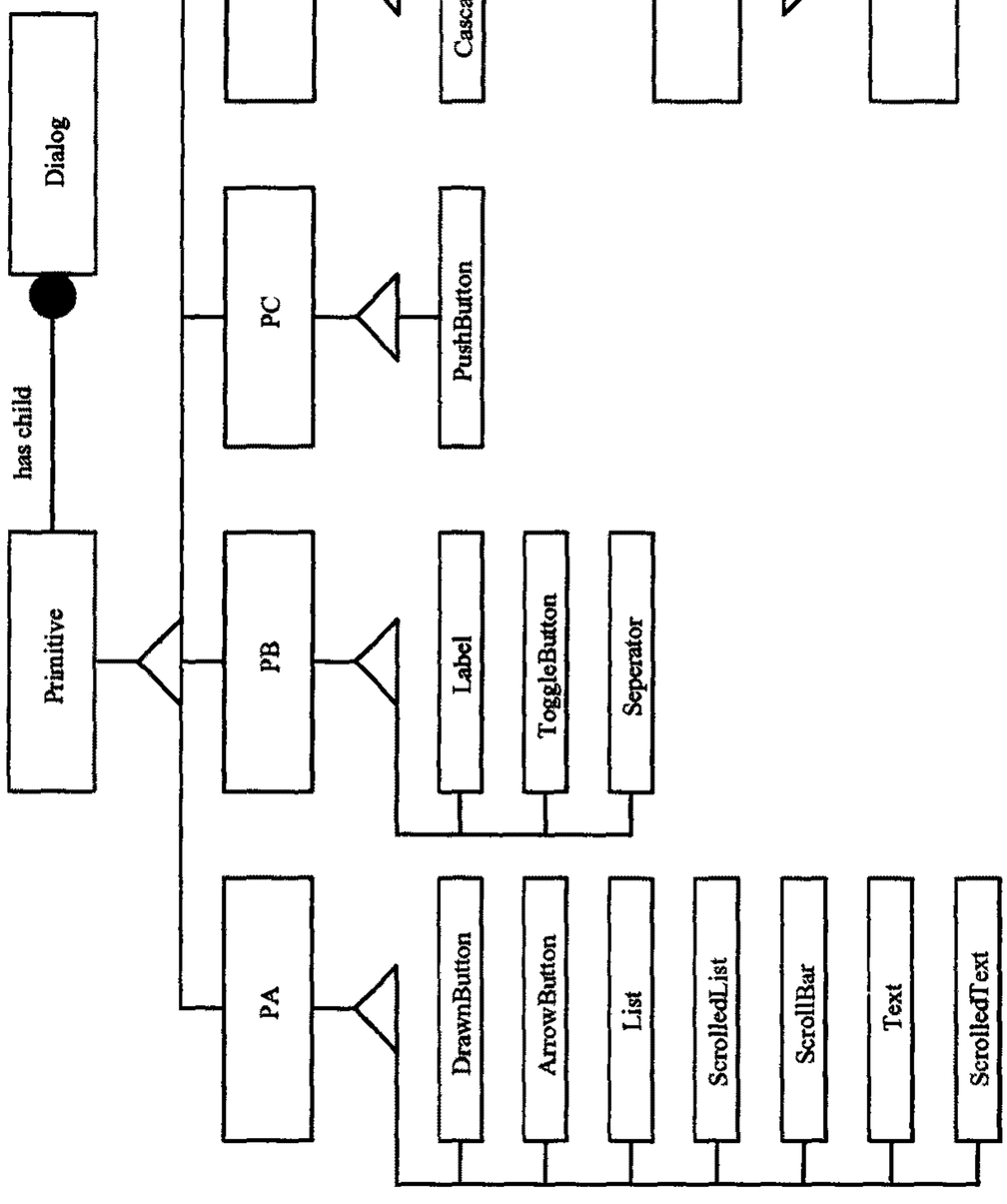
[그림 3-4-16] 위짓의 종류



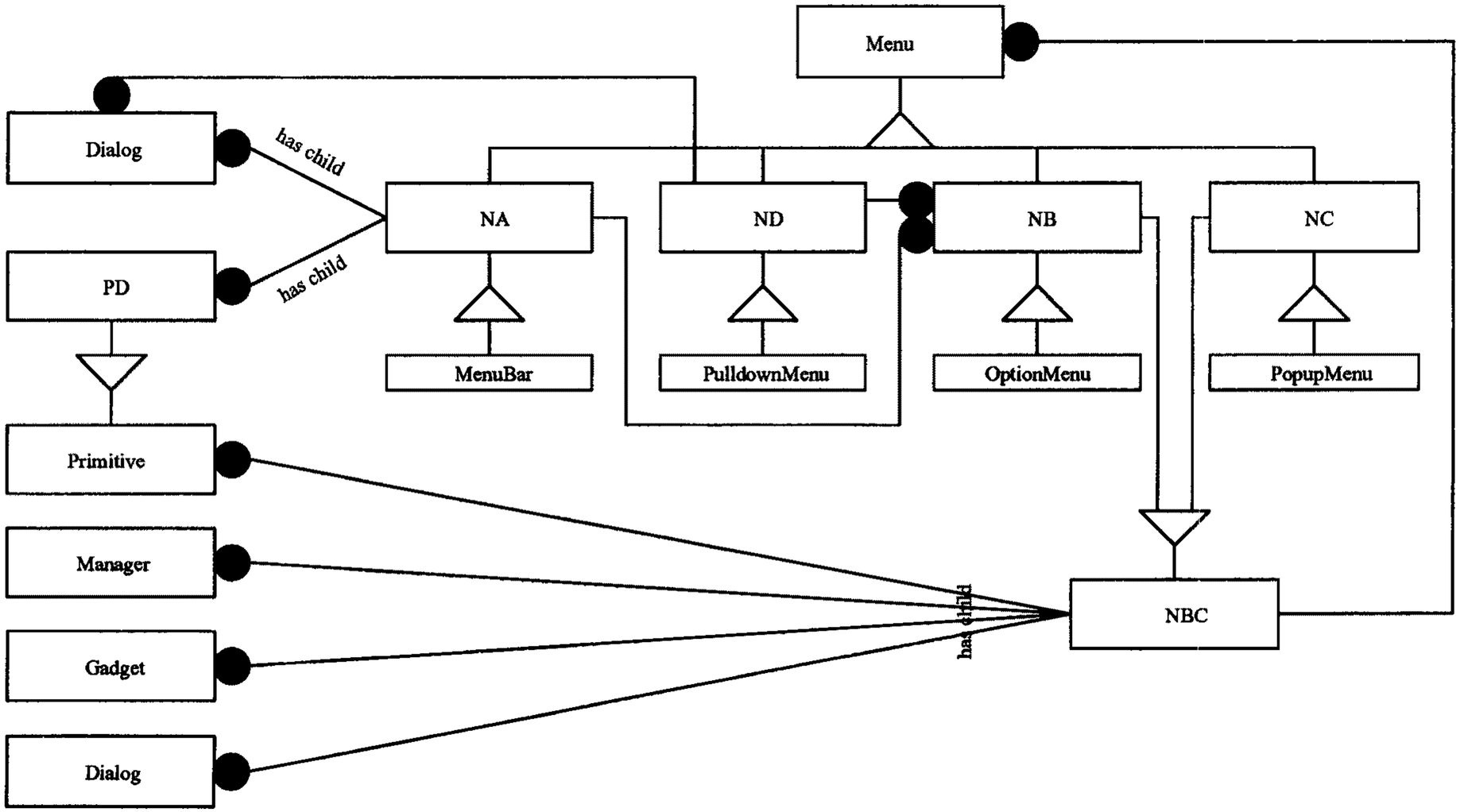
[그림 3-4-17] Manager 위짓의 종류와 합성 규칙



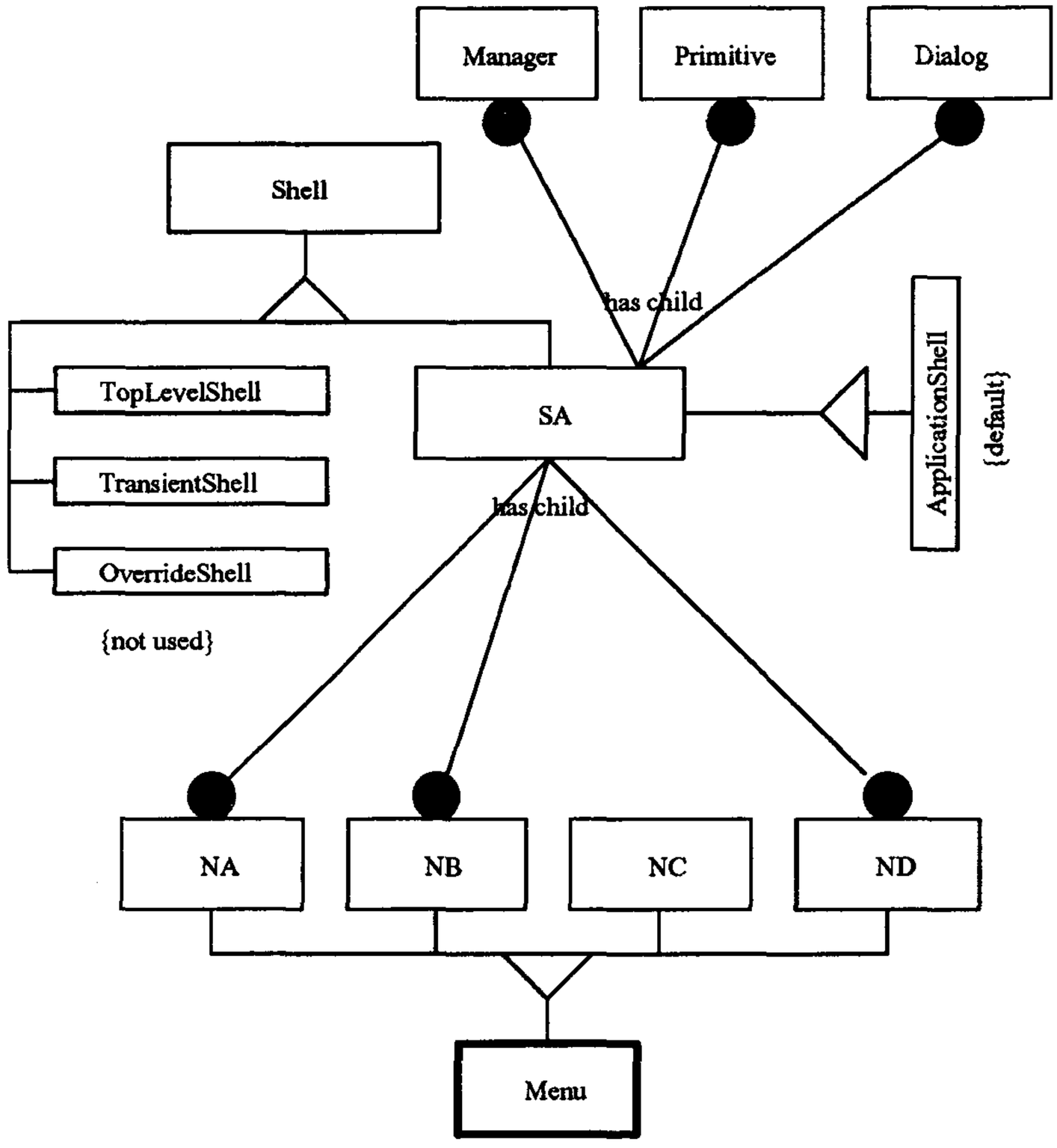
[그림 3-4-18] Dialog 위짓의 종류와 합성 규칙



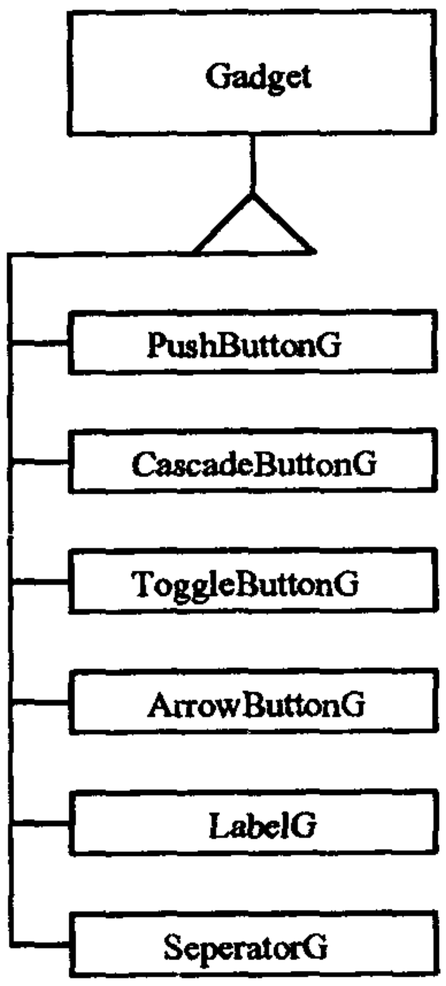
[그림 3-4-19] Primitive 위젯의 종류와 합성 규칙



[그림 3-4-20] Menu 위짓의 종류와 합성 규칙

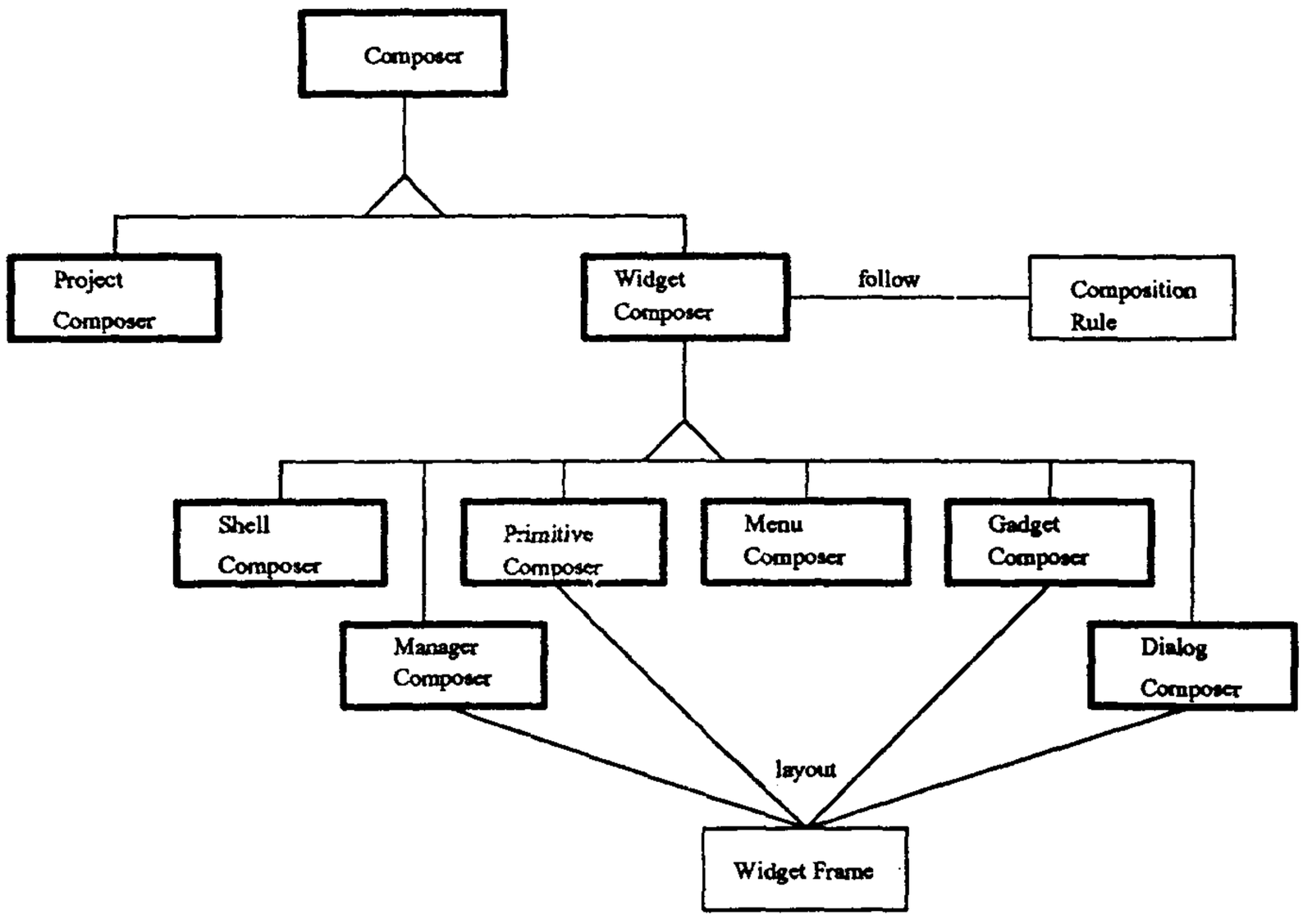


[그림 3-4-21] Shell 위짓의 종류와 합성 규칙



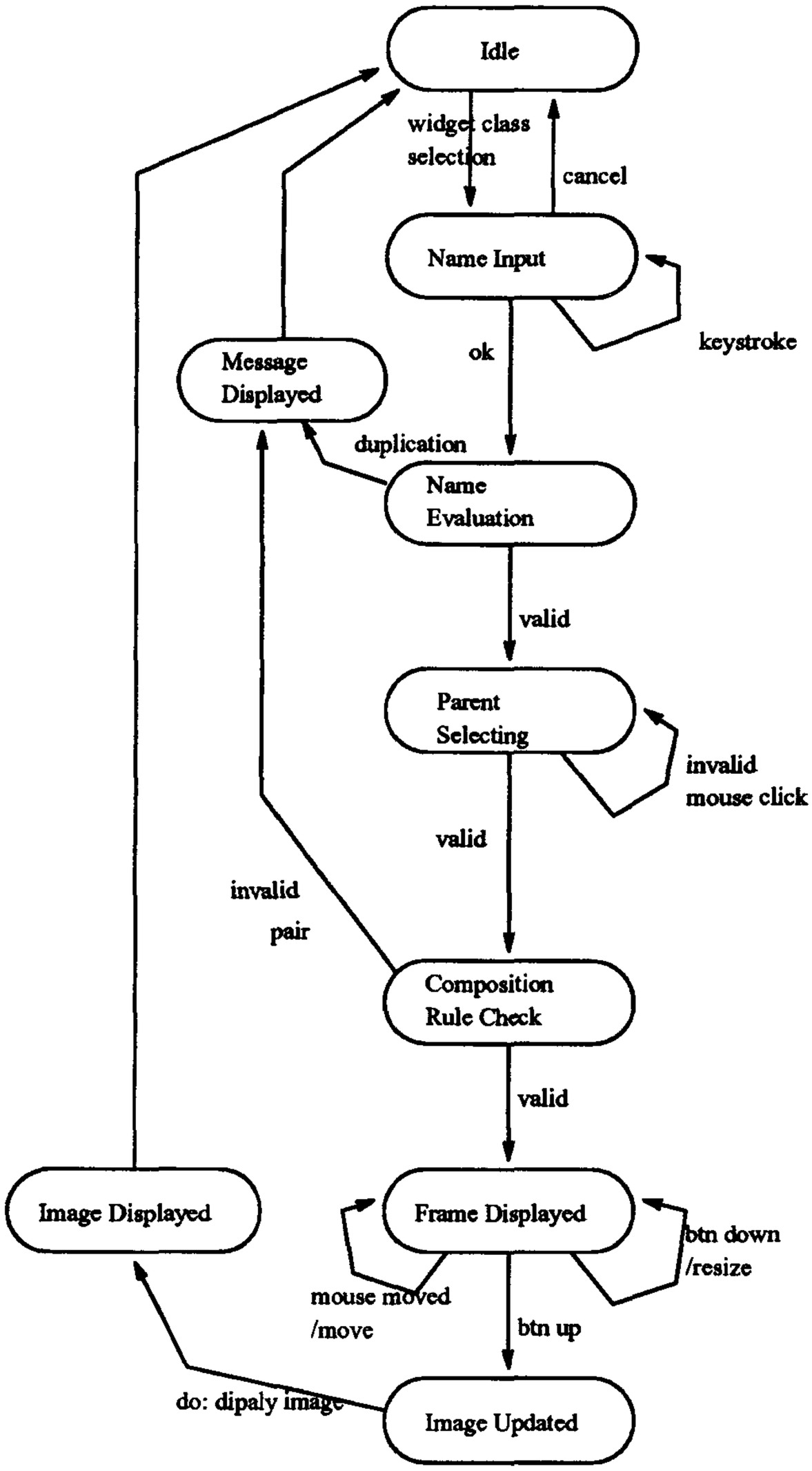
[그림 3-4-22] Gadget의 종류와 합성 규칙

기능 위주 객체인 위짓 합성기(Widget Composer)의 상속 체계와 위짓 윤곽 (Widget Frame) 및 합성 규칙 사이의 관계는 [그림 3-4-23]에서 보여준다.



[그림 3-4-23] 위짓 합성기의 종류와 위짓 윤곽 및 합성 규칙의 관계

위짓 합성은 작업절차에 따라 9개로 구성되며, 각 상태 사이의 천이는 각기 다른 상황에서 발생한다. [그림 3-4-24]는 위짓 합성 작업의 상태 천이를 보여준다.

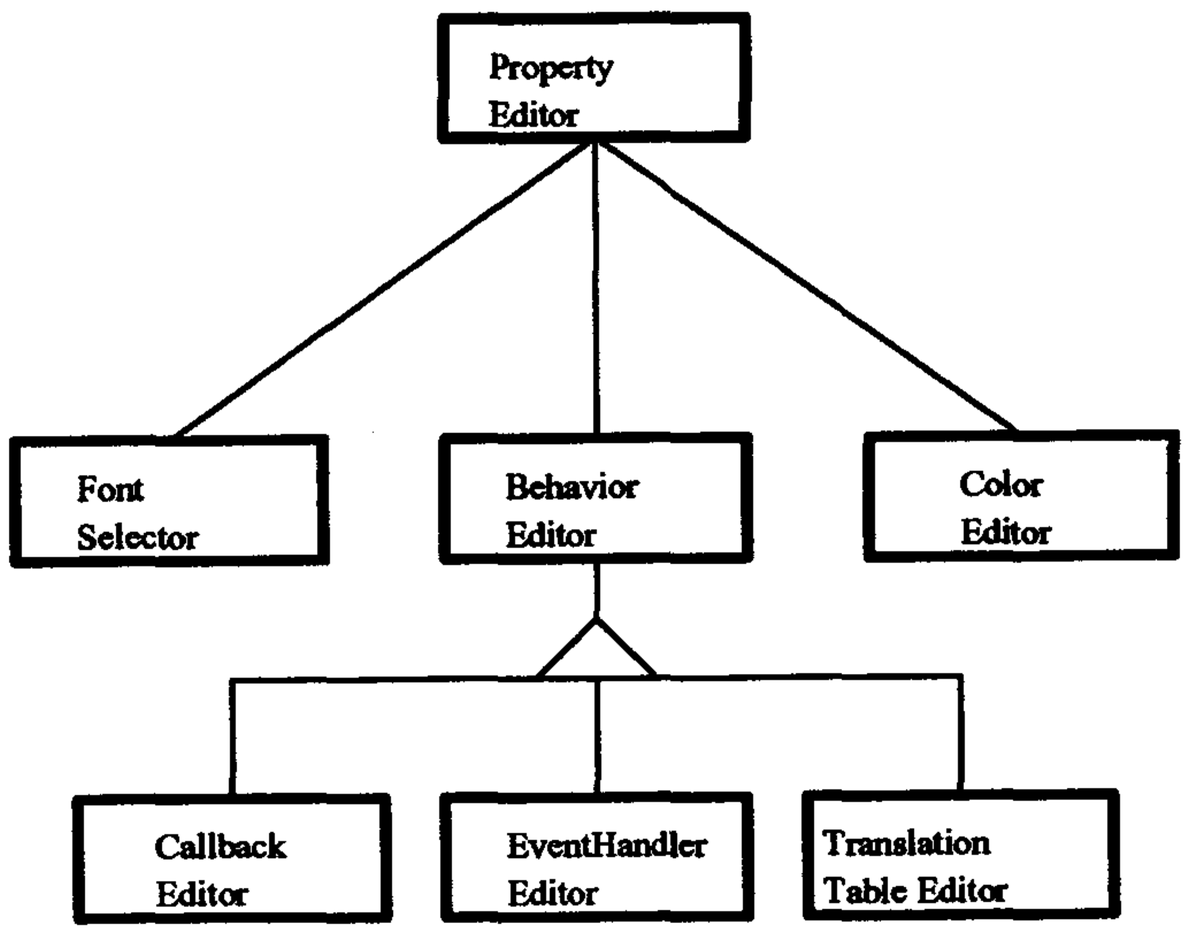


[그림 3-4-24] 위짓 합성 작업의 상태 천이

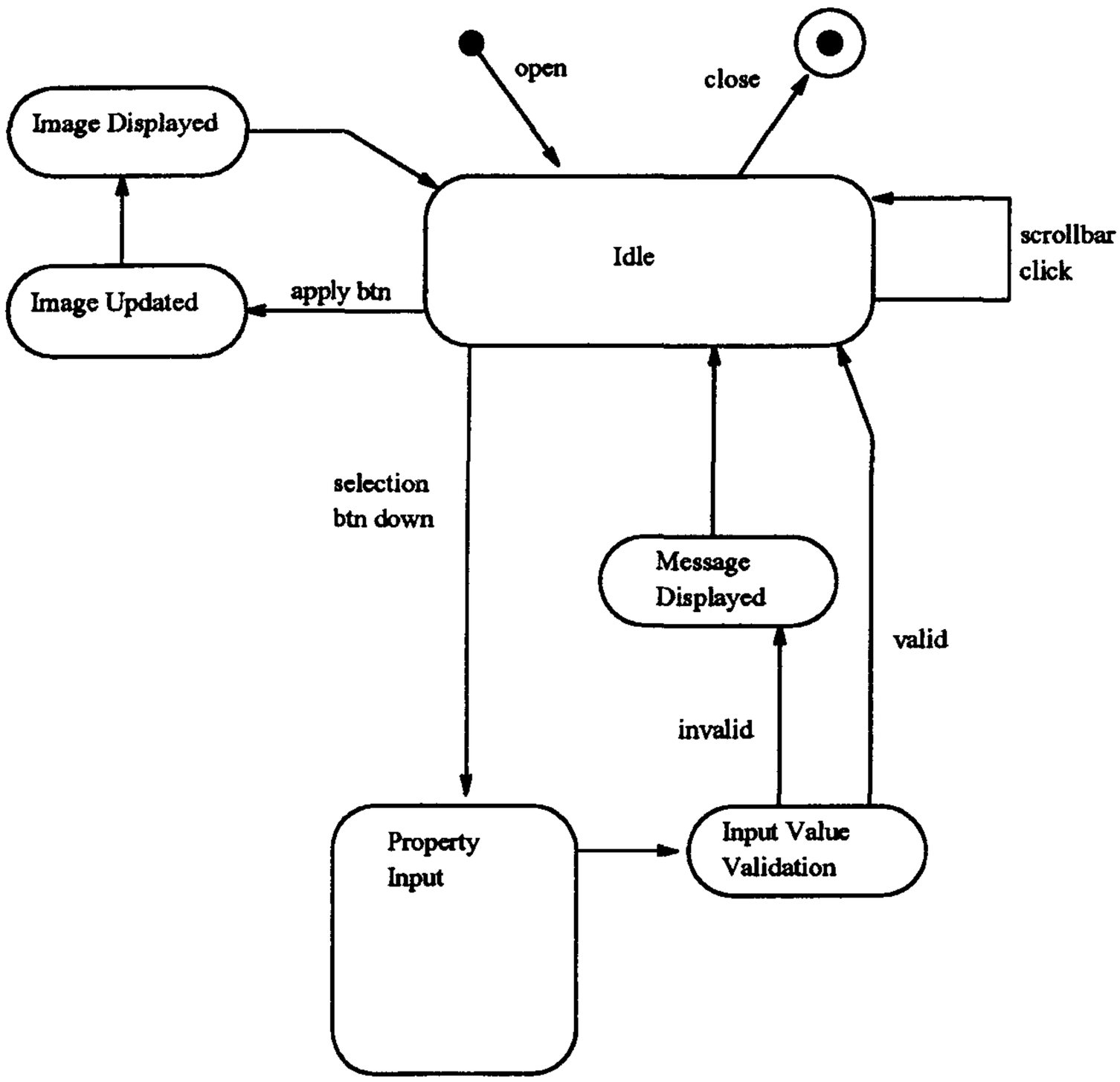
위짓 특성 편집은 합성된 위짓의 특성을 편집하는 기능으로 특별한 작업 절차는 존재치 않고 편집의 결과를 화면에 반영하는 기능이 포함된다. 편집해야할 위짓의 특성은 위짓 클래스 별로 차이가 있다.

특성들 중에서 미리 정의된 범위에서 선택할 경우에는 옵션 메뉴로 처리하고, 선택 대상의 수가 정해지지 않은 경우는 선택할 수 있는 다이알로그를 제공하여 결정하며, 위짓을 참조할 경우는 해당 위짓이 존재하는 가를 검증하며, 색상과 관련된 경우에는 색상 편집기를 통하여 색상을 결정하며, 행위와 관련된 특성을 편집할 경우에는 콜백 편집기, 사건처리기 편집기, 행위 번역표 편집기 등과 프로시듀어 편집기, 응용 편집기 등과 연계하여 편집하며, 그외의 경우는 특성 편집기 상의 값 정의 난을 이용한다. 한편 특성 편집기에서는 각 자원이 원시 코드에서 하드코딩될 것인지 또는 자원 화일 코드화 될 것인지에 관한 정보를 입력한다.

[그림 3-4-25]는 특성 편집기의 기능적 객체들의 관계를 보여주며, [그림 3-4-26]은 개략적인 상태의 천이를 보여준다.



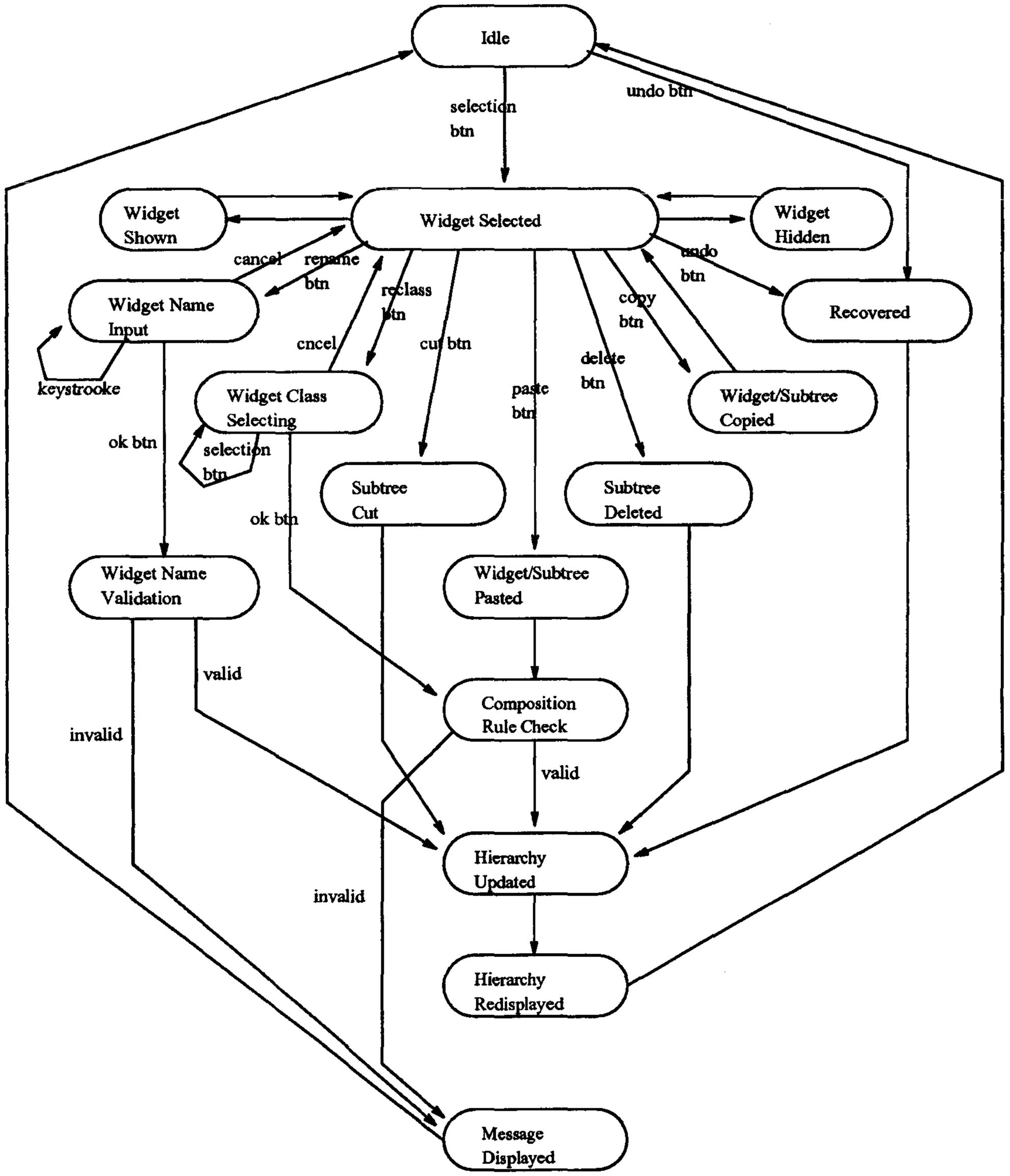
[그림 3-4-25] 위짓 특성 편집기의 기능 객체의 구성



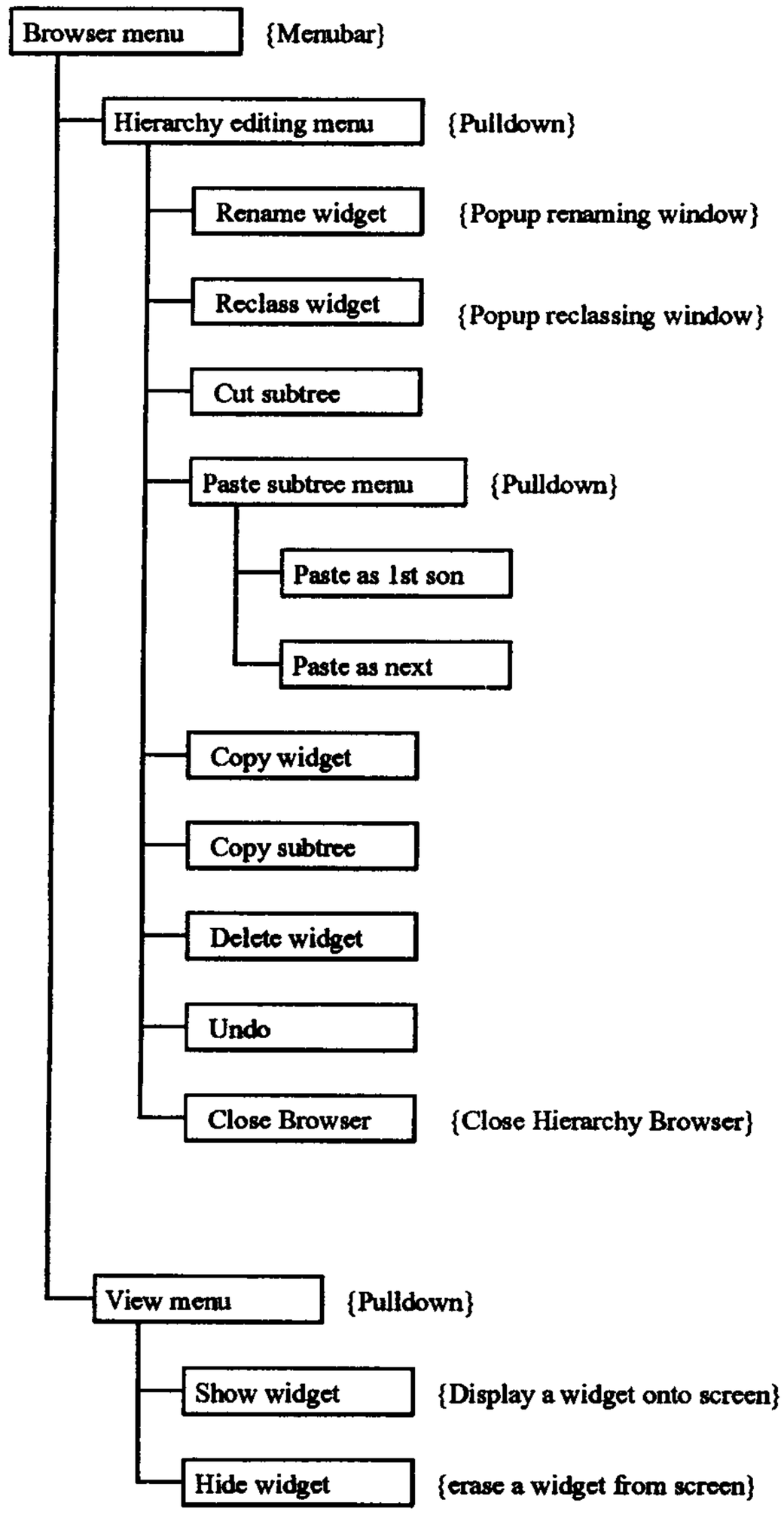
[그림 3-4-26] 위짓 특성 편집기의 상태 천이

인터페이스 구조의 편집은 위짓 합성의 결과로 만들어진 하나의 위짓 트리에 해당하는 인터페이스의 구조를 조정하는 작업으로 다양한 부속 작업들의 집합 작업이다. 다음과 같은 부속 기능들이 있다.

- 위짓의 이름을 바꾸는 기능(Rename Widget)
- 위짓의 클래스를 바꾸는 기능(Reclass Widget)
- 하나의 위짓을 복사하는 기능(Copy Widget)
- 부속 트리를 복사하는 기능(Copy Subtree)
- 부속 트리를 자르는 기능(Cut Subtree)



[그림 3-4-27] 구조 편집기의 상태 천이



[그림 3-4-28] 구조 편집기의 메뉴 상태 체계

- 잘린 부속 트리를 일정한 위치에 붙이는 기능

  (Paste as 1stson, Paste as next)

- 위짓 부속 트리를 삭제하는 기능(Delete Widget)

- 바로 앞의 변경 편집을 원상 회복 시키는 기능(Undo)

마지막으로 프로젝트 이미지의 위짓들을 화면에 구현하는 기능은 인터페이스 단위로 구현하며, 화면과 관련있는 특성들 만을 반영하며, 행동과 관련된 특성은 구현치 않는다.

5. 정형적 명세 언어

정형적 명세 언어는 프로젝트의 정보를 텍스트 언어의 형태로 기술하기 위한 것이며, 궁극적으로는 프로젝트 명세(Project Specification)를 표현하는 방법이다. 이 명세 언어의 모형은 [그림 3-4-9]의 구성을 따르며, 각 정보의 단위를 객체화 하여 X 자원을 기술하는 코드의 확장된 표현 방법을 사용하여, 비 순차적(non-procedural)으로 표현할 수 있다.

정보 객체의 종류는 [그림 3-4-30]에서 보여주는 것과 같이 다섯가지로 단순화 하며, 위짓은 모든 Motif 위짓 클래스와 사용자 위짓 클래스를 대표하는 클래스이다. 위짓 특성은 위짓 클래스를 속성화 하고, 응용과 프로시듀어의 종류는 각기 상위 클래스를 속성화 한다. 클래스 사이의 관계는 다(many) 쪽에서 하나(one) 쪽을 부모로 정의하여 속성화 한다.

각 정보 클래스의 속성은 [그림 3-4-31]과 같다. Project 클래스는 속성으로 file name을, Interface 클래스는 Project 객체를 가리키는 parent를, Widget 클래스 류는 Interface 객체를 가리키는 parent, 자신을 원시코드에서 global로 할 것인 지의 여부를 결정하는 global, 화면에 구현할 것인 지를 나타내는 visible, 그리고 위짓 특성들인 properties를, Application과 Procedure 객체는 Project 객체를 가르키는 parent, 선언부인지 정의부 인지를 나타내는 type 및 물리적으로 코드를 갖고있는 file을 속성으로 갖는다.

- 위짓을 화면에 보여주는 기능(Show widget)

- 위짓을 화면에서 감추는 기능(Hide Widget)

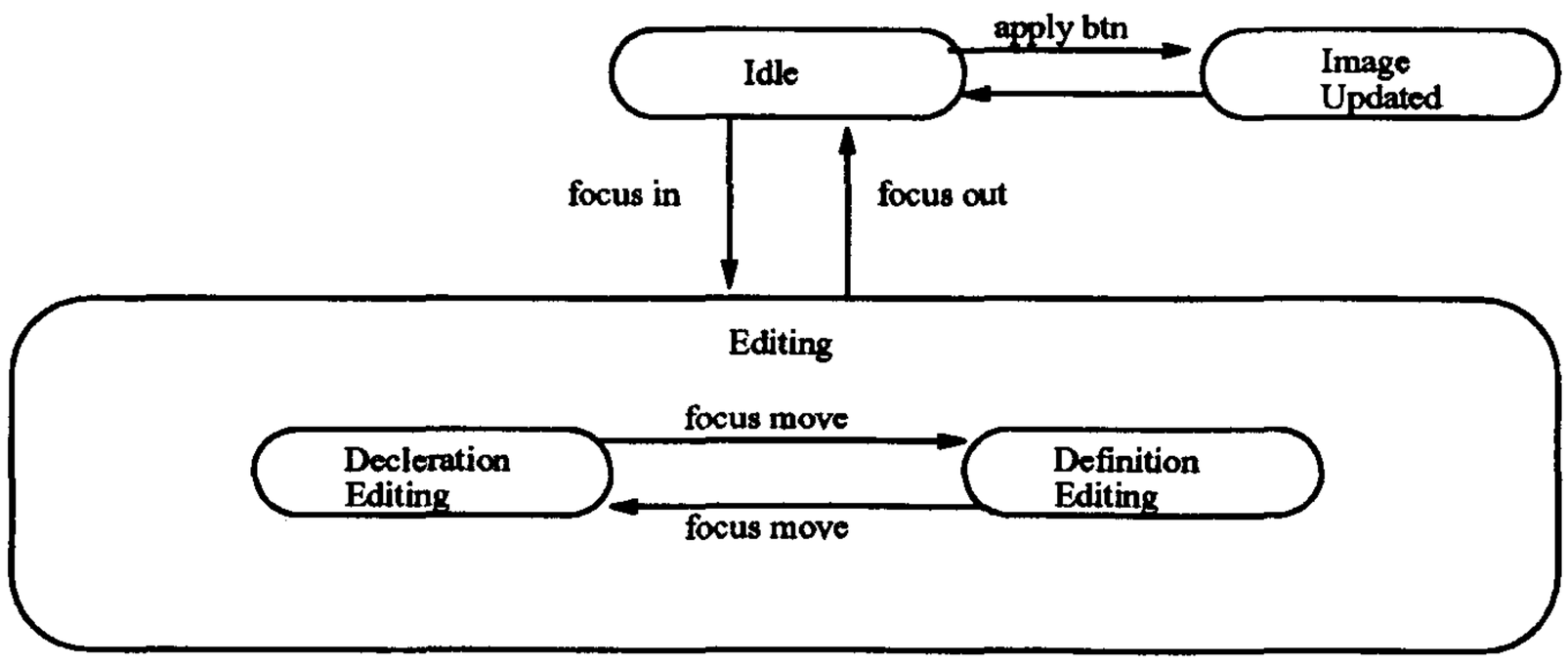
위짓의 이름을 바꾸는 경우에는 이름의 중복성 여부를 검증하며, 위짓 클래스를 바꾸는 경우에는 해당 위짓을 중심으로 부모와 자식과의 합성 규칙을 검증한다. 또한 부속 트리를 붙이는 경우에도 합성 규칙을 검증한다.

구조 편집기는 매우 대화식이며 다양한 기능으로 많은 상태가 존재한다. [그림 3-4-27]은 구조 편집기의 상태 천이를 보여주며, [그림 3-4-28]은 구조 편집기의 메뉴 상태 체계를 보여준다.

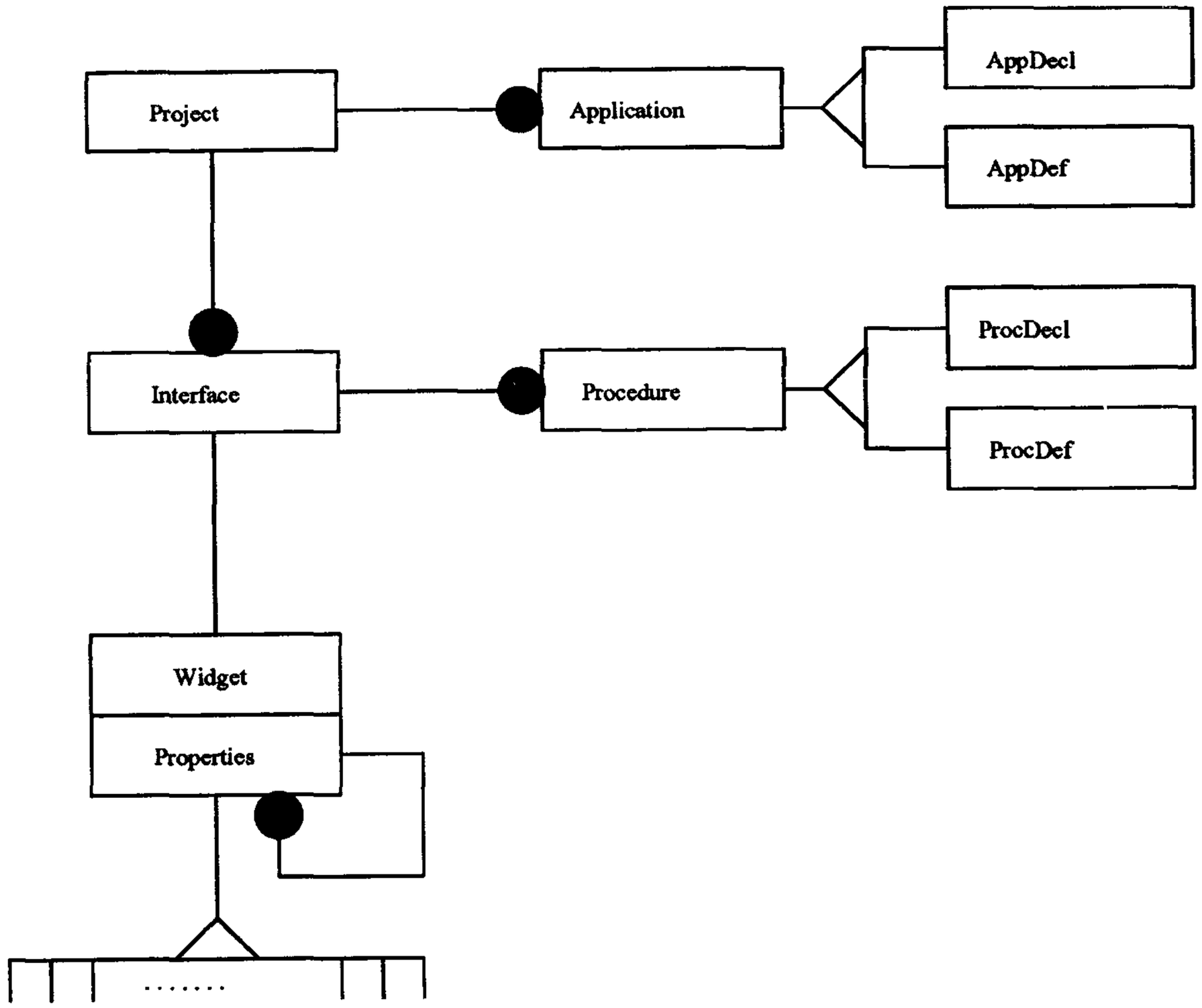
프로시듀어 편집에서 프로시듀어란 위짓 특성 중에서 콜백(Callback), 사건처리기(EventHandler), 행위 번역표(Action Translation Table)와 연계된 함수들을 말한다. 따라서 프로시듀어 편집은 프로그램 코드를 편집하는 작업이다. 프로시듀어 편집기에서는 선언부와 정의부를 각기 편집할 수 있다. 프로시듀어는 인터페이스 별로 그룹을 이룬다.

응용 편집에서 응용이란 응용 프로그램을 말하며 편집 과정은 프로시듀어의 편집과 유사하다. 그러나 응용의 그룹은 사용자가 마음대로 구성할 수 있다.

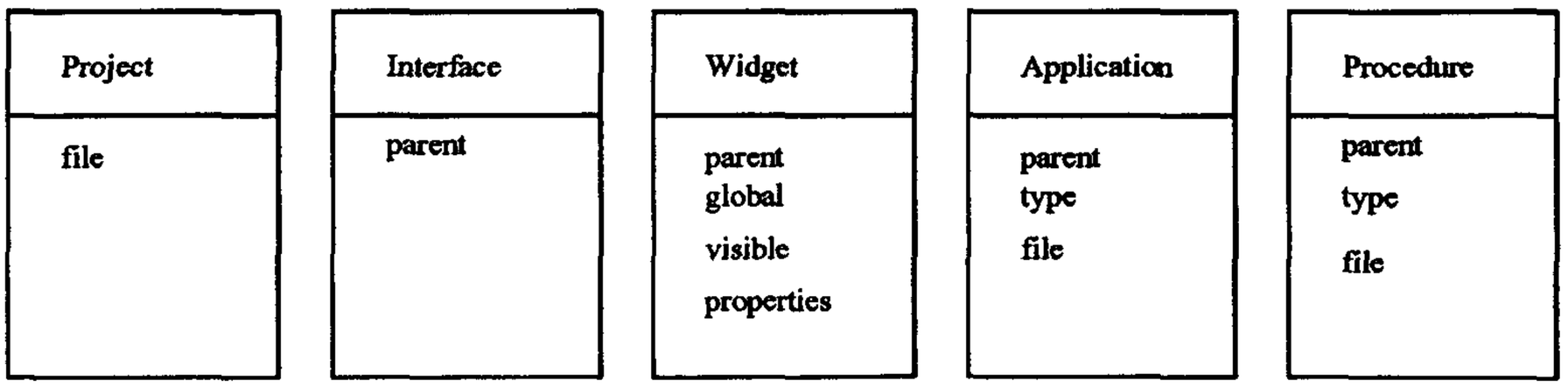
[그림 3-4-29]는 구조 편집기와 응용 편집기의 상태 천이를 보여준다.



[그림 3-4-29] 구조 편집기와 응용 편집기의 상태 천이



[그림 3-4-30] 명세 언어의 정보 객체 사이의 관계



[그림 3-4-31] 정보 클래스의 속성

정보 객체의 표현은 다음의 기본 양식을 근간으로 표현한다.

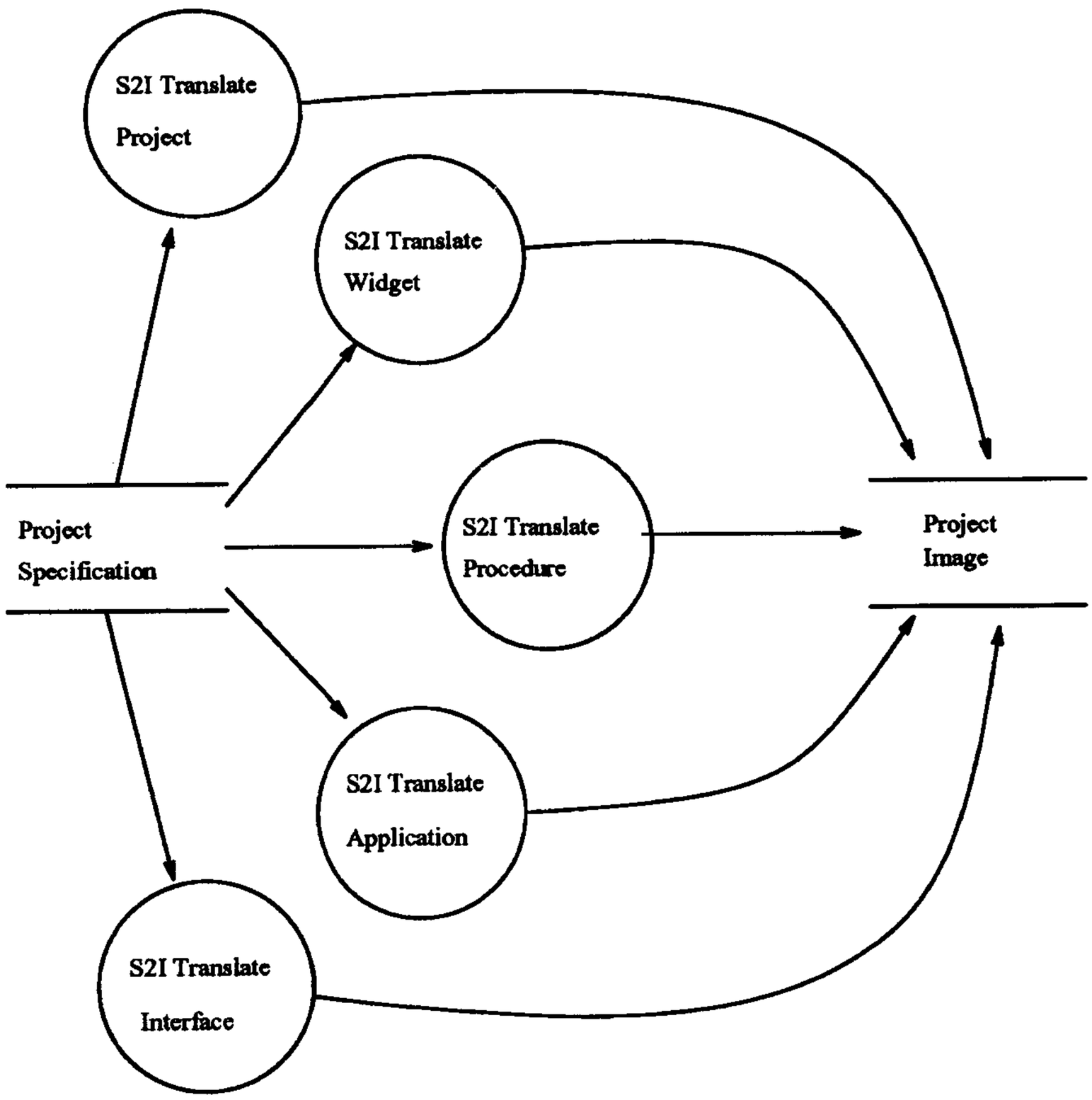
*[object name].class: [class name]

*[object name].[object attribute name]: [attribute value]

6. 명세코드 번역기

명세코드 번역기는 프로젝트 명세(Project Specification) 정보를 프로젝트 이미지(Project Image) 정보로 변환하는 기능으로 [그림 3-4-7]의 "Translate Graphical Spec into Textual Spec"에 해당한다. 번역기는 명세어의 다섯가지 종류의 정보를 명세로 부터 이미지로 변환한다.

[그림 3-4-32]은 명세코드 번역기의 기능적인 모형을 보여준다.

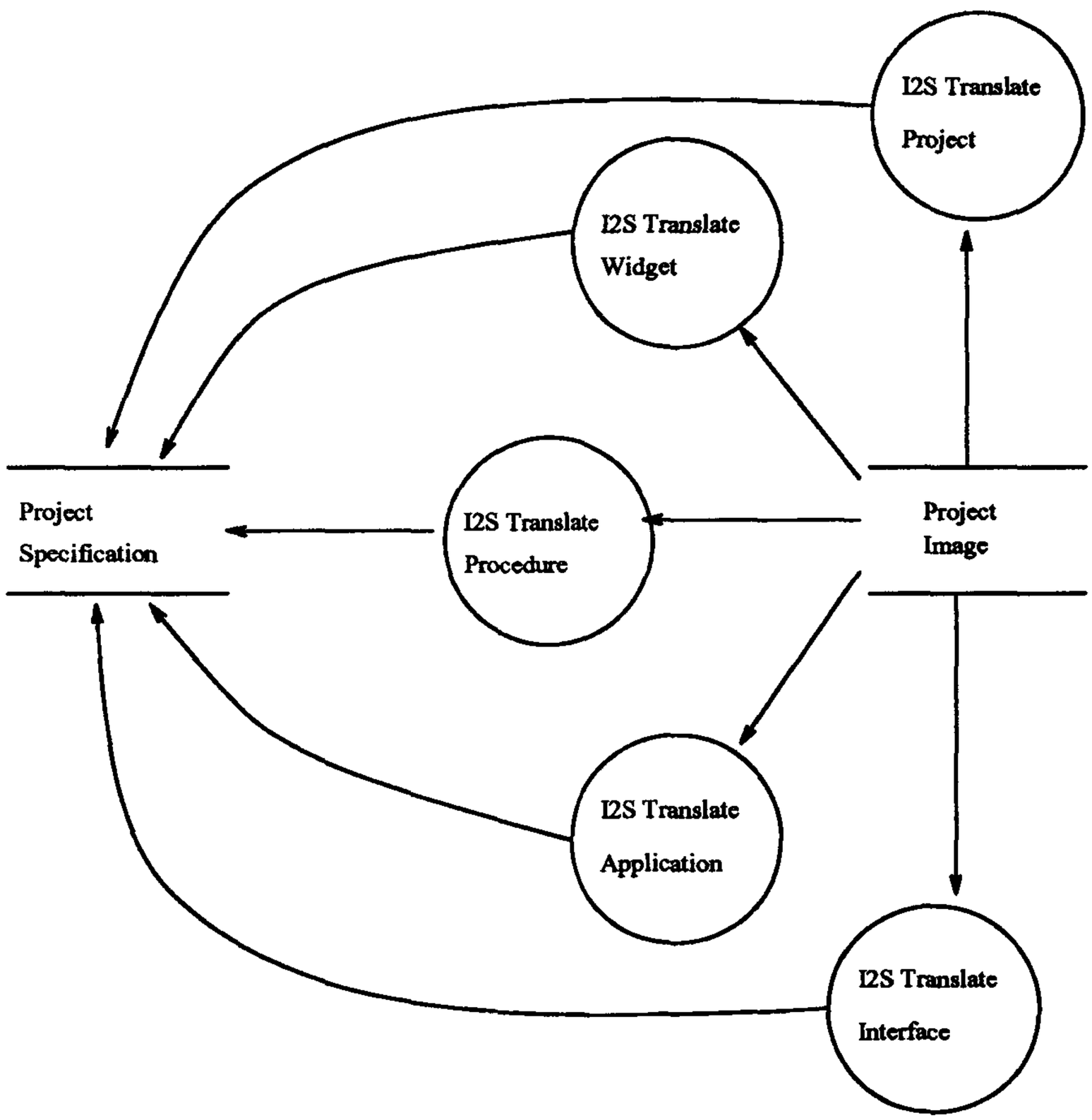


[그림 3-4-32] 명세 번역기의 기능적 모형

7. 명세코드 생성기

명세코드 생성기는 명세코드 번역의 역 작업을 수행하는 기능으로 [그림 3-4-7]의 "Translate Graphical Spec into Textual Spec"에 해당한다.

[그림 3-4-33]는 명세코드 생성기의 기능적 모형이다.

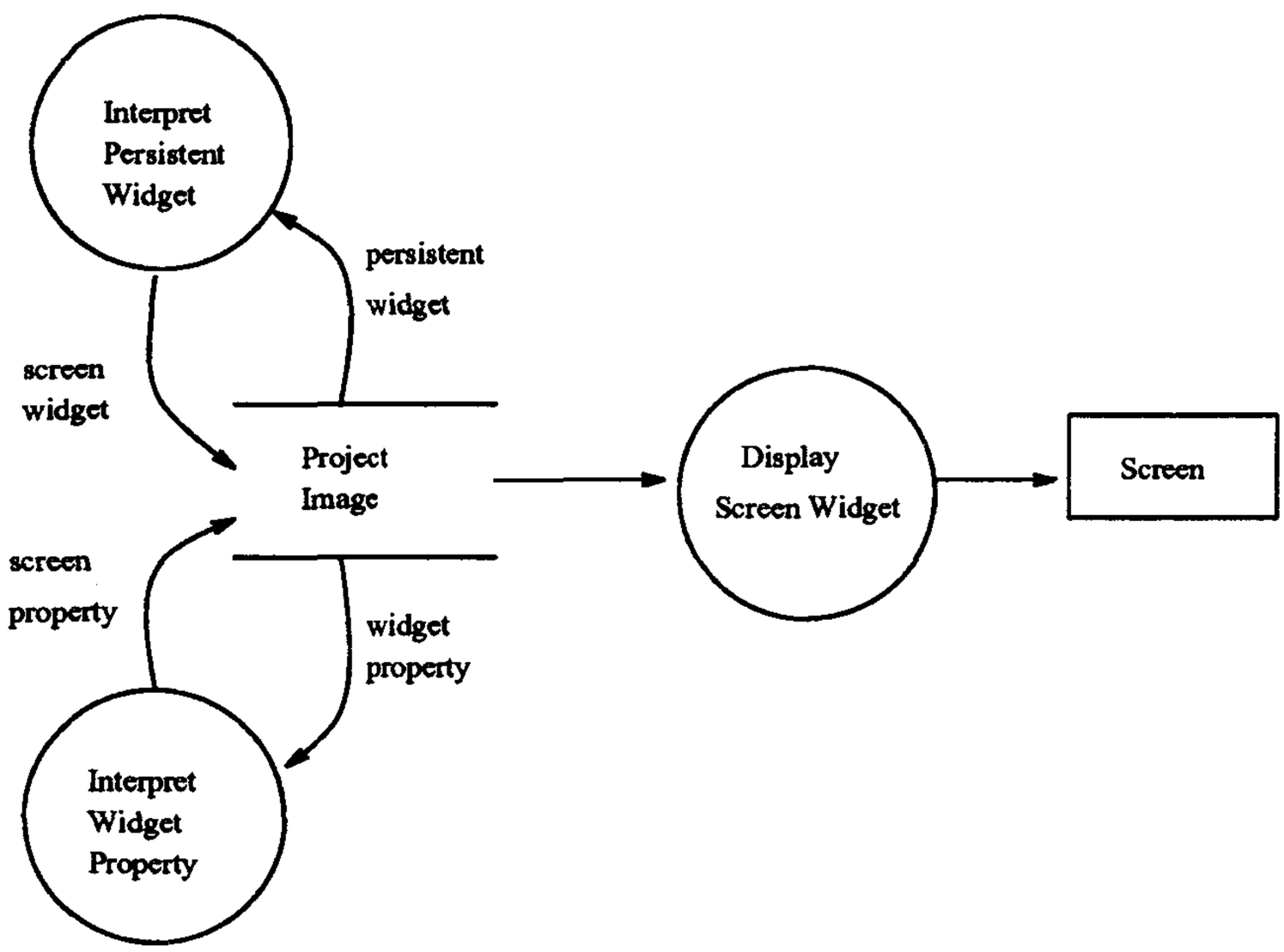


[그림 3-4-33] 명세코드 생성기의 기능적 모형

8. 인터페이스 구현기

인터페이스 구현기는 프로젝트 이미지의 단위 화면 요소를 화면에 구현하는 기능을 수행한다. [그림 3-4-7]의 "Interprete Graphical Spec"에 해당한다. 구현기는 위짓 특성을 반영한 위짓을 만들어 프로젝트 이미지의 Screen Widget에 배정하고, 그 Screen Widget을 화면에 구현한다.

[그림 3-4-34]는 구현기의 기능 모형을 보여준다.



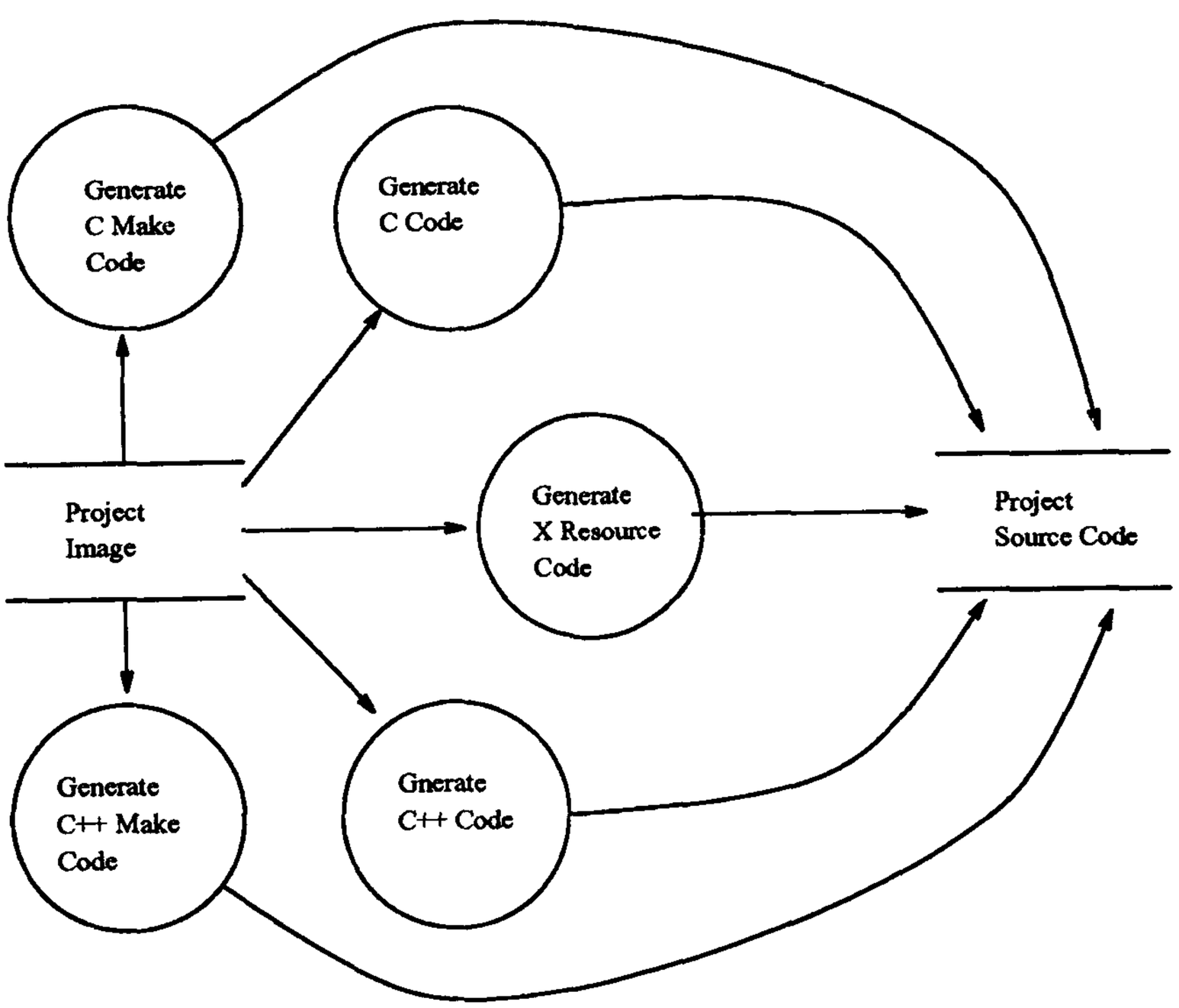
[그림 3-4-34] 인터페이스 구현기의 기능 모형

9.원시코드 생성기

원시코드 생성기는 GUI Mosaic 도구의 매우 중요한 기능을 수행하는 부분으로서, 프로젝트 이미지로 부터 원시코드를 만드는 기능을 수행한다. 원시코드는

[그림 3-4-11]에 나타나는 종류의 코드를 생성한다. C 또는 C++ 원시코드를 만들며 동시에 X 자원코드와 C make 코드와 C++ make 코드를 만들어 수정없이 컴파일이 가능한 환경을 제공한다.

[그림 3-4-35]는 원시코드 생성기의 기능 구성을 보여준다.



[그림 3-4-35] 원시코드 생성기의 기능 모형

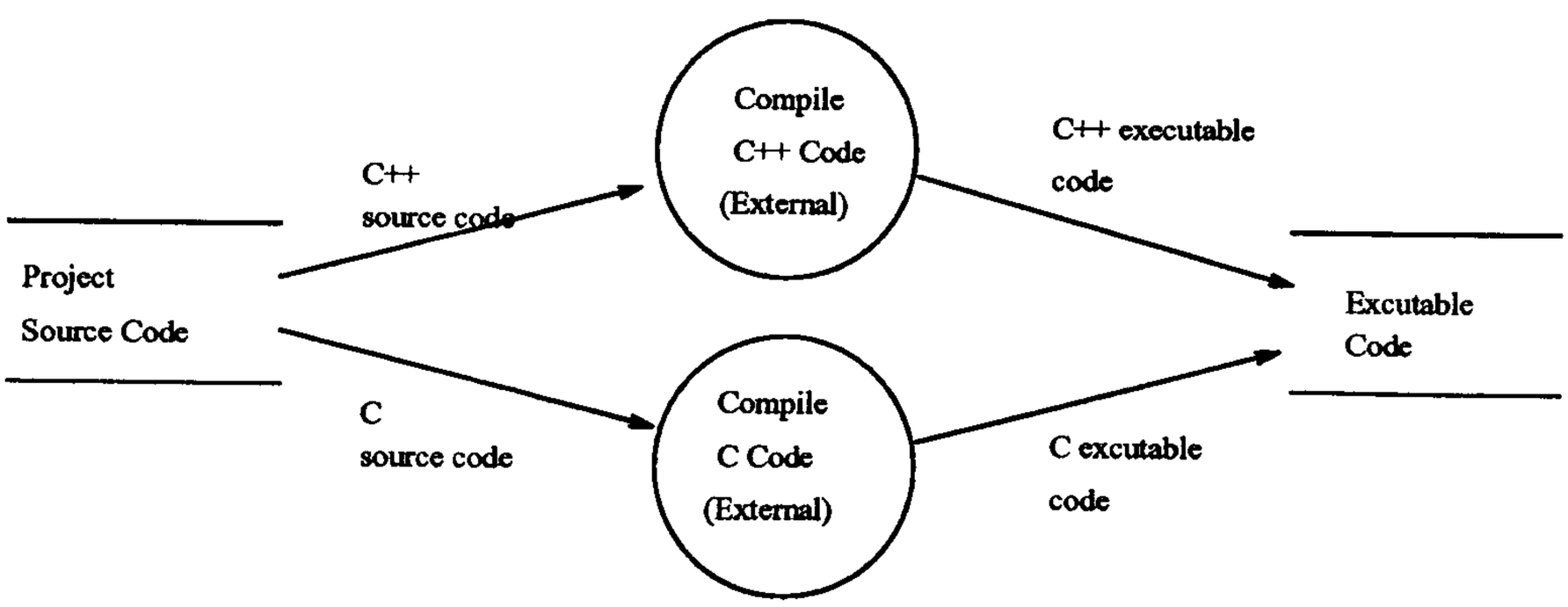
C++ 언어는 객체지향 언어인 점에서 C 언어와 차이가 있으므로 생성될 원시코드를 위한 기본모형의 설계가 필요하다. 기본모형은 생성되는 위짓 객체를 각기 하나의 C++ 클래스에 대응되도록 하고 위짓간의 부모 자식 관계를 잘 나타낼 수 있도록 한다. 또한 객체지향 프로그램의 다양한 특성들, 즉 캡슐화(Encapsulation), 추상클래스(Abstract class), 속성상속(Inheritance), 다형성(Polymorphism)등의 특성등을 반영하도록 구성한다. 따라서 각각의 위짓을 기본

적으로 독립된 객체로 설정하고, 이 위짓들의 공통점을 추출하여 추상클래스를 구축한다. 속성상속과 다형성은 각각 추상클래스와 서브클래스의 메시지(함수) 정의를 통해서 구현된다.

C 코드군의 생성은 C++ 코드군의 생성과 마찬가지로 명세코드를 입력받아 C 원시코드 및 메이크화일, 자원화일 등을 생성한다. 사용자 응용코드 부분은 수정없이 응용화일로 생성되므로 사용자의 임의성이 적용되는 부분이다. C 원시코드는 클래스의 생성 및 멤버의 선언 등이 없으므로 위짓의 구조 부분과 관계되는 코드은 C++의 경우와 다르다.

먼저 위짓을 생성(creation)하는 함수 정의와 행위(behavior) 의 정의부로 구성한다. 단위 위짓별로는 위짓구조 및 위짓간의 관계 설정 등을 고려하여 속성 설정, 생성, 관리(manage), 콜백(callback)을 하나의 쌍으로 구성한다. C++ 코드와 마찬가지로 인터페이스 별로 applicationShell을 생성하여 다중 toplevel 프로그램을 가능하도록 한다. 콜백 함수는 위짓생성모듈을 완료하고 콜백타입별로 모듈을 구성한다. 명세코드의 name 은 C 원시코드의 유일한 이름으로 정의한다.

10. 원시코드 컴파일



[그림 3-4-36] 외부 원시코드 컴파일러 이용 기능 모형

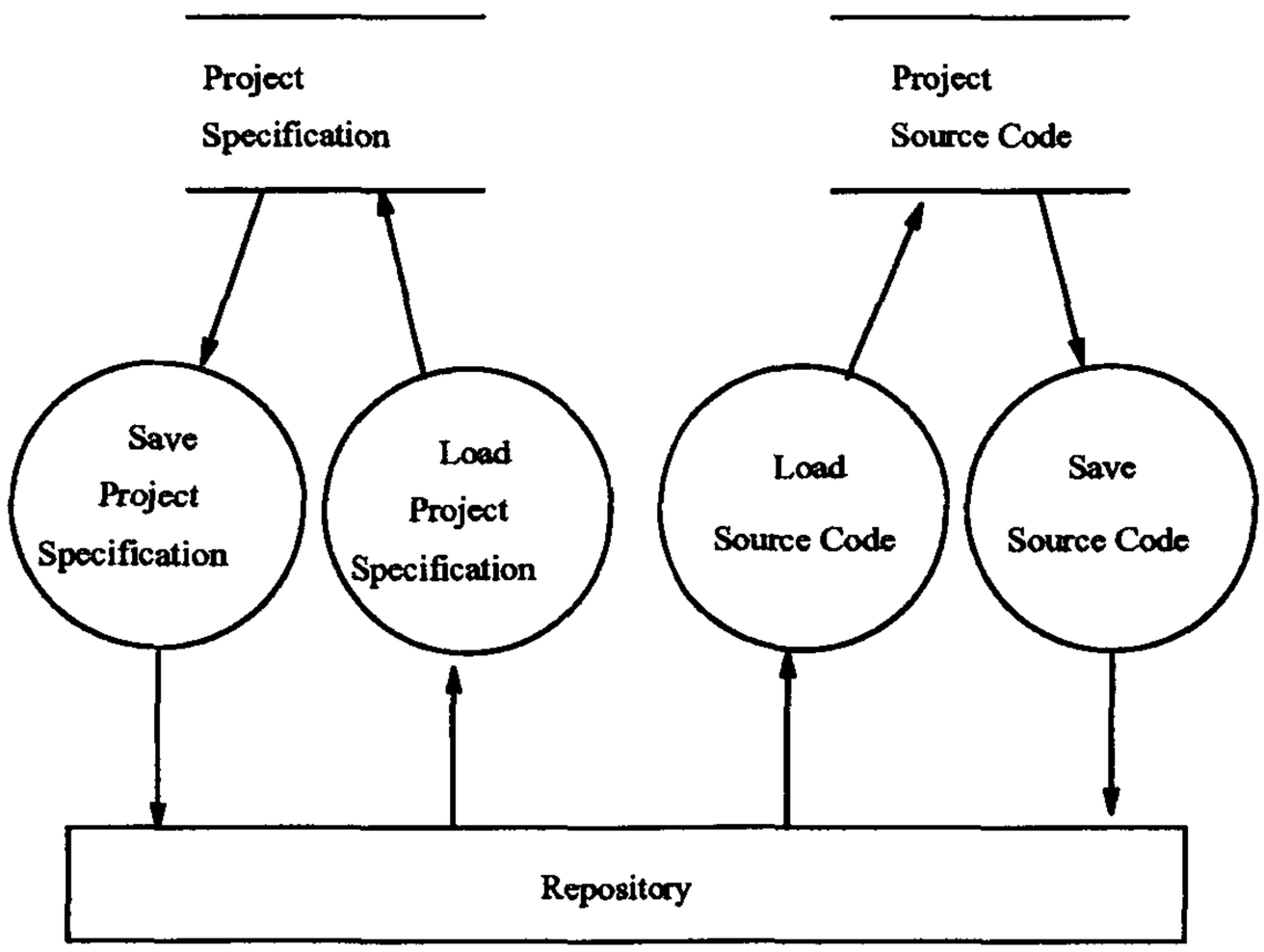
원시코드 컴파일은 원시코드 생성기에서 생성된 C 또는 C++ 코드를 컴파일하는 과정으로 외부의 컴파일러를 이용하여 컴파일을 수행한다.

[그림 3-4-36]은 C 또는 C++코드를 컴파일함 을 보여준다.

11. 정보 저장소 접속기

정보 저장소 접속기는 GUI Mosaic 단위 도구로서는 반드시 필요한 기능은 아니나, 들무새 시스템에서는 모든 정보를 저장소에 일괄적으로 관리함으로서 정보의 일관성과 도구간 상호 이용성 및 재사용성을 높이는 목적에 부합하기 위해서는 반드시 필요한 부분이다.

[그림 3-4-37]에서 보여주는 바와 같이 네가지의 기능을 수행한다. 이러한 기능들은 프로젝트를 도구에 올리거나 또는 변경된 프로젝트 정보를 저장할 때 연계되어 이용된다. 정보는 정보의 단위를 화일로 한다.



[그림 3-4-37] 저장소 접속기의 기능 모형

여 백

제 4 장 프로토타잎 개발

여 백

제 4 장 프로토타잎 개발

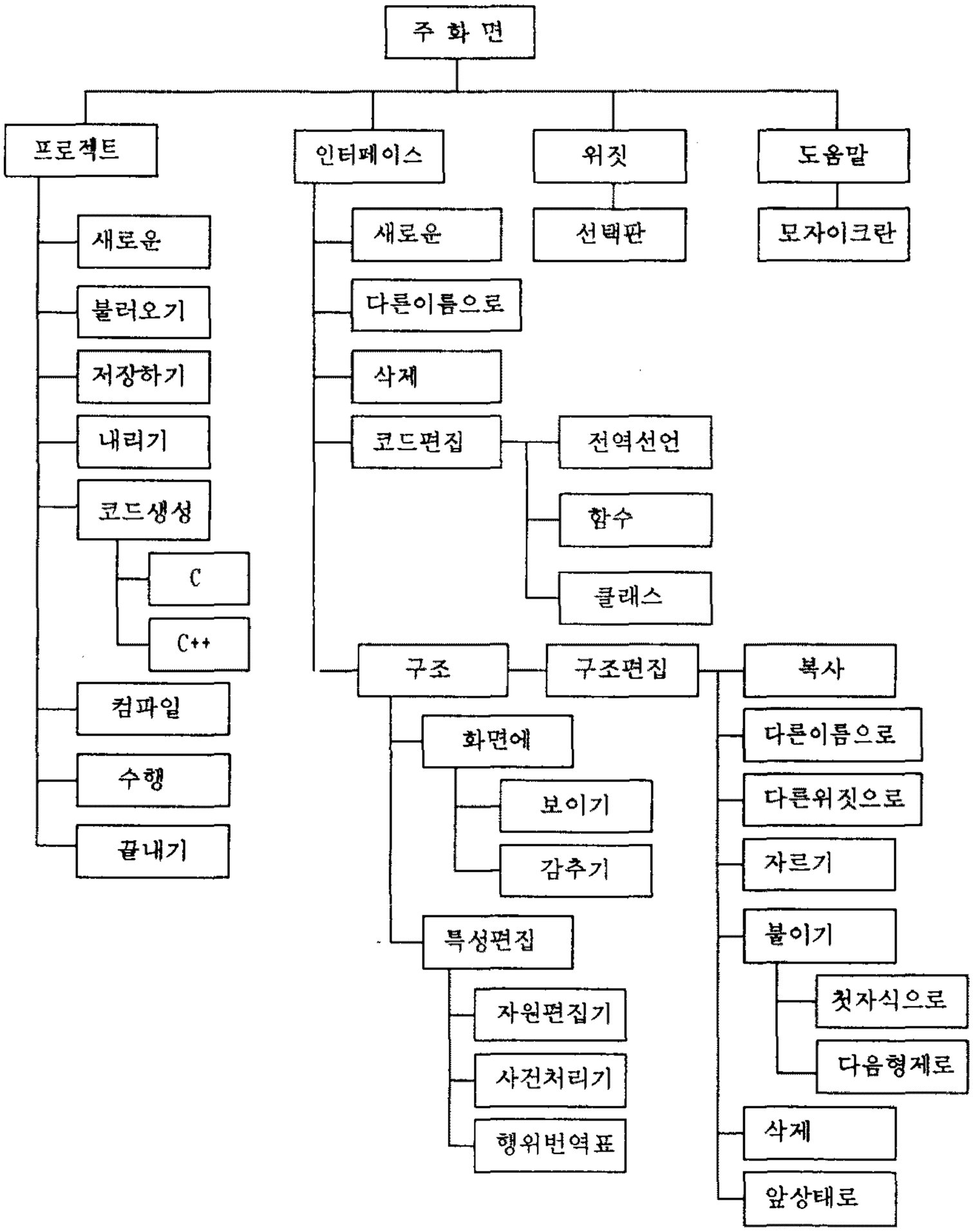
제 1 절 개요

Motif 위짓을 응용 프로그램의 사용자 인터페이스로 사용하고자 하는 사용자들을 위하여, Motif 환경에서 사용자 인터페이스를 개발할 수 있는 GUI 모자이크를 개발하였다. 대부분의 UIMS 도구들과 마찬가지로 모자이크 시스템도 인터페이스 합성작업과 편집 등을 시각적인 작업대를 통하여 수행할 수 있도록 하였으며, 사용자의 모든 작업을 스크린에 즉각 반영토록 하여 빠른 프로토타잎을 가능하도록 하였다. 또한 사용자의 작업영역을 스크린 전체로 확대하여 원하는 모양과 크기가 반영될 수 있도록 하였다. 사용자의 작업 결과는 명세언어 형태로 저장, 관리할 수 있고, C 또는 C++ 원시코드로 저장할 수 있다.

시스템의 구성은 초기 시스템을 구동하고 전체 프로젝트를 관리할 통합화면과, 인터페이스의 시각적인 합성을 지원할 위짓합성기, 합성된 위짓의 속성과 계층등을 즉각적으로 변경, 관리할 수 있도록 하는 속성편집기 및 구조편집기, 시각적인 작업의 결과를 텍스트로 변환하고, 텍스트로 된 명세를 시각화 해 주는 명세코드생성기 및 명세 번역기, 최종 작업결과를 컴파일 가능한 코드로 생성해 주는 C 또는 C++ 코드생성기로 구성된다. 이들은 [그림 4-1-1] 과 같은 메뉴체계를 가지고 사용자의 인터페이스 개발작업을 지원하게 된다.

시스템의 개발은 명세코드를 먼저 정의하고, C++ 코드생성기를 만들어 명세코드를 입력으로 하여 사용자 인터페이스를 개발하는 방법을 채택하였다. GUI모자이크 시스템의 모체가 되는 초기 화면의 명세코드를 에디터를 통하여 편집한 후

이를 코드생성기를 통하여 원시코드로 생성하고 컴파일 실행해 보고, 다시 요구되는 기능 및 인터페이스를 추가하는 작업을 반복하였다. 즉 rapid application design 및 prototyping 을 본 시스템의 설계와 구현에 적용한 사례라고 할 수 있겠다.



[그림 4-1-1] GUI모자이크의 메뉴체계도

GUI모자이크 시스템은 SunOS Release 4.1.3, X Window R5, Motif 1.2 에서 개발하였다.

제 2 절 프로토타입 구현

1. 사용자 인터페이스 정보의 구조

GUI모자이크 시스템은 사용자 인터페이스를 세가지 형태로 분류하였다. 첫째는 프로젝트 정보로써, 프로젝트가 저장될 물리적인 화일 명 또는 디렉토리 등에 관한 정보가 이에 속한다. 둘째는 인터페이스 정보로써, 사용자에 의하여 구분되는 화면단위로써 인터페이스 명과 그 인터페이스 안에서 공통으로 사용될 응용 프로그램에 관한 기술(description) 등이 이에 속한다. 세째는 인터페이스를 구성하는 각각의 위짓에 관한 정보로써, 위짓간의 관계 및 속성정의들로 이루어져 있다.

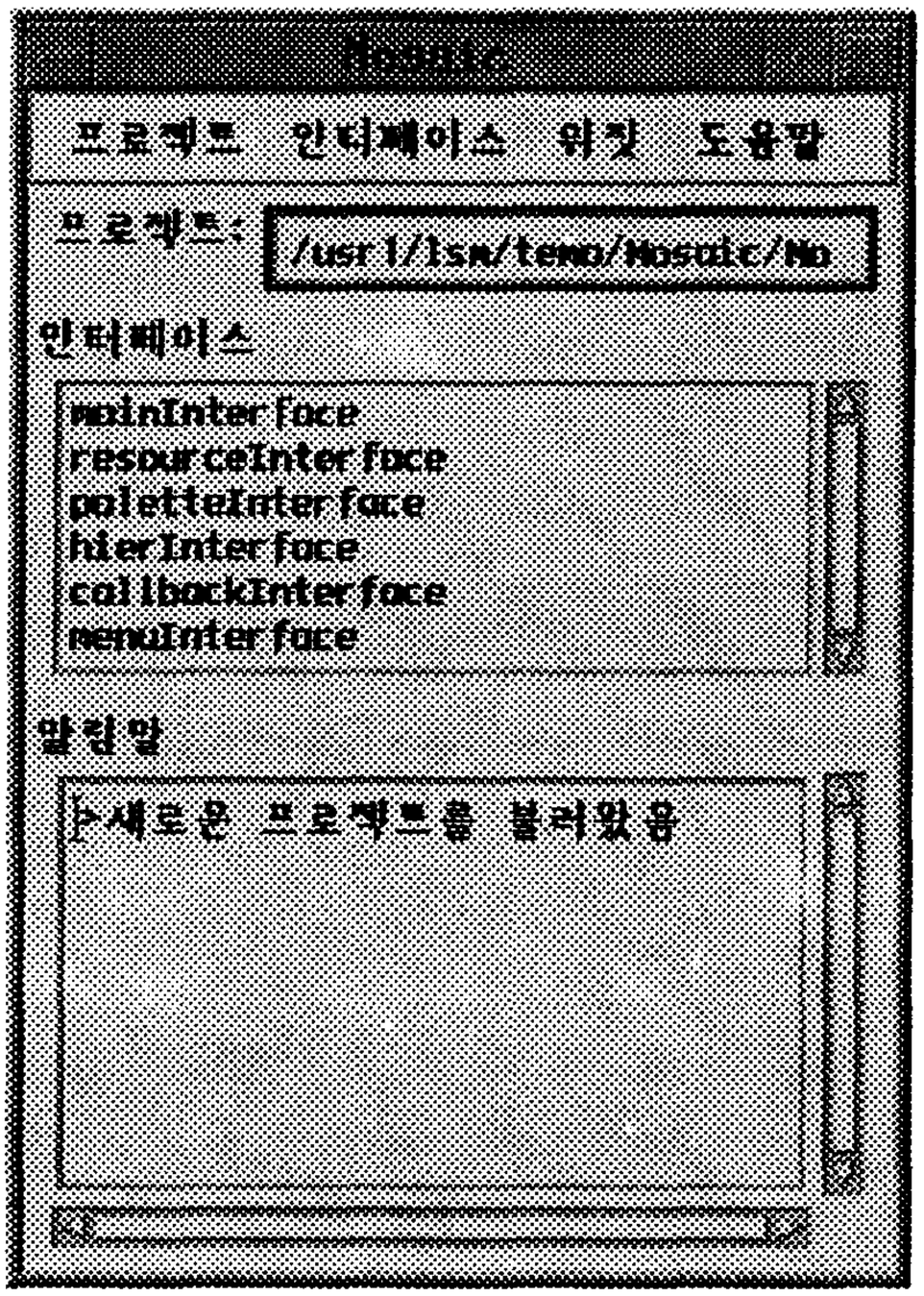
이 세가지 형태의 정보는 GUI모자이크 시스템안에서 트리 형태로 반영,관리된다. 시스템이 구동되고 있는 동안 사용자에 의하여 생성, 삭제, 변경되는 모든 화면 정보를 반영하여 트리의 생성, 변경, 삭제를 동적메모리 할당(dynamic memory allocation)으로 구현하였다.

즉 명세정보는 모자이크 시스템 안에서 트리정보로 변환되고, 트리정보는 명세정보로 변환되어 화일로 저장되기도 하고, 코드생성기의 입력자료로 들어가서 C 또는 C++ 원시코드로 변환되기도 한다.

시스템이 종료되면 트리정보는 소멸된다.

2. 통합화면

통합화면은 GUI모자이크 시스템을 구동하면 처음으로 대하게 되는 화면으로서, 시스템의 전반적인 작업을 관리할 수 있도록 구성하였다. 즉, 프로젝트 및 인터페이스의 생성, 로드(load), 저장, 코드생성 및 컴파일, 수행 등을 이 화면에서 관리하고, 인터페이스에 관한 정보를 한눈에 볼 수 있다. 또한 시스템에서 보내는 메시지를 통합화면의 알림말 창을 통하여 볼 수 있다.



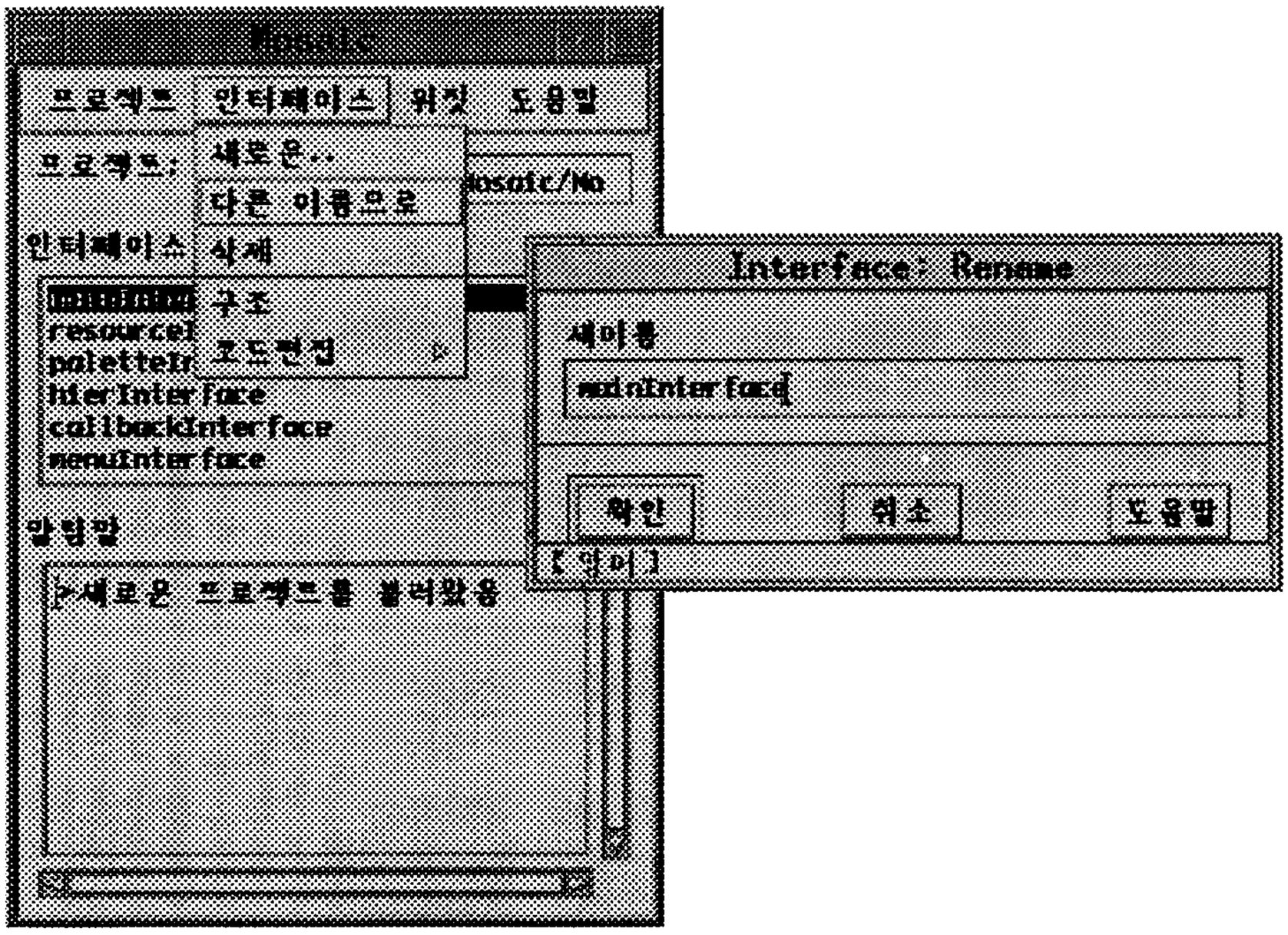
[그림 4-2-1] 통합화면

화면구성은 인터페이스 정보의 종류에 따라 프로젝트 정보, 인터페이스 정보,

위짓정보 별로 풀다운 메뉴를 가지고 있는 메뉴바와 사용자에 의하여 선택된 프로젝트의 경로(path)와 이름을 나타낼 프로젝트 창, 프로젝트를 구성하고 있는 모든 인터페이스를 리스팅하고 화면에 보여줄 인터페이스창, 사용자의 작업에 대한 피드백(feedback)과 에러메시지를 보내줄 알림말 창으로 구성하였다.

프로젝트 메뉴는 프로젝트 생성에서 부터 저장, 코드생성 및 수행, 시스템 끝내기에 이르기 까지 사용자의 작업절차 전반을 이곳에서 관리할 수 있도록 하였다. 이에 관해서는 작업절차관리기에서 상세히 기술한다.

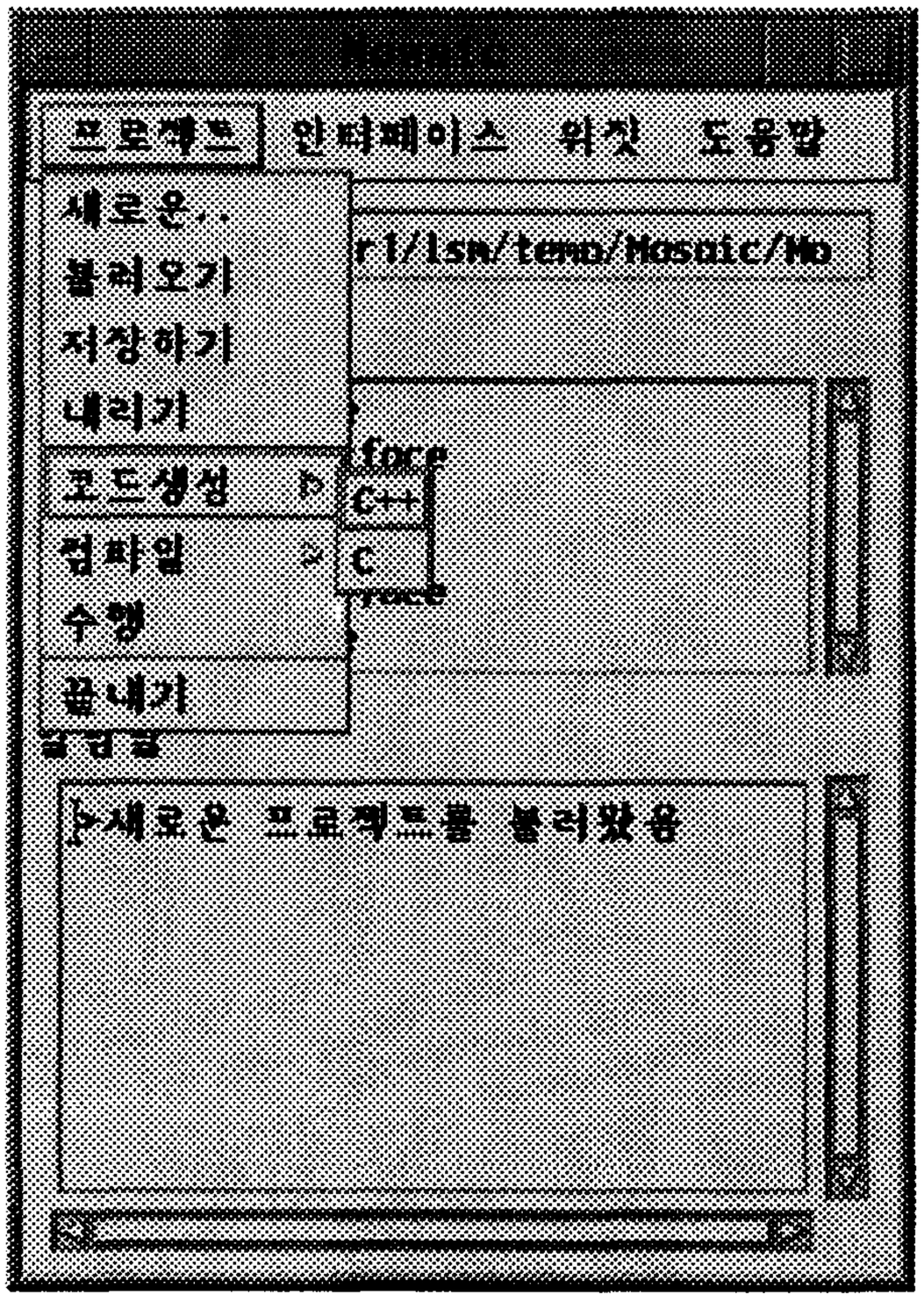
인터페이스 메뉴에서는 인터페이스의 생성, 이름 변경, 삭제 등 인터페이스 구성과 관련된 메뉴와 함께, 인터페이스를 구성하고 있는 위짓들의 구조를 조회할 구조조회기의 팝업(popup)과, 한 인터페이스 안에서 공통으로 사용될 응용 프로그램의 작성을 위한 응용프로그램 에디터를 팝업(popup)할 수 있도록 하였다.



[그림 4-2-2] 인터페이스 메뉴

3. 작업절차 관리기

통합화면의 프로젝트 메뉴는 사용자가 이미 작업한 프로젝트 혹은 새로 작업할 프로젝트를 로드, 또는 생성으로 부터 작업결과의 실제 수행에 이르기 까지 모든 사용자의 작업 절차를 지원하도록 구성하였다. 프로젝트의 생성 및 기존 프로젝트의 선택에 있어 사용자의 편의를 돕기 위하여 화일선택상자(file selection dialog)를 이용하여 선택시 팝업(popup)될 수 있도록 하였다.



[그림 4-2-4] 작업절차관리기

화일선택상자를 통하여 선택된 화일을 로드하면, 지정된 화일에 저장되어 있던 명세정보가 프로젝트 트리로 구성되어 사용자의 작업을 지원하며, 명세정보 중 인터페이스 정보는 인터페이스 브라우저(scrolledList) 에 리스팅 된다.

역시 화일선택상자를 통하여 새로운 프로젝트를 생성하면, 새로운 프로젝트를 위한 트리를 초기화 한 후 인터페이스 메뉴의 새로운 인터페이스를 생성할 때 마다 인터페이스에 관한 정보를 트리에 추가한다.

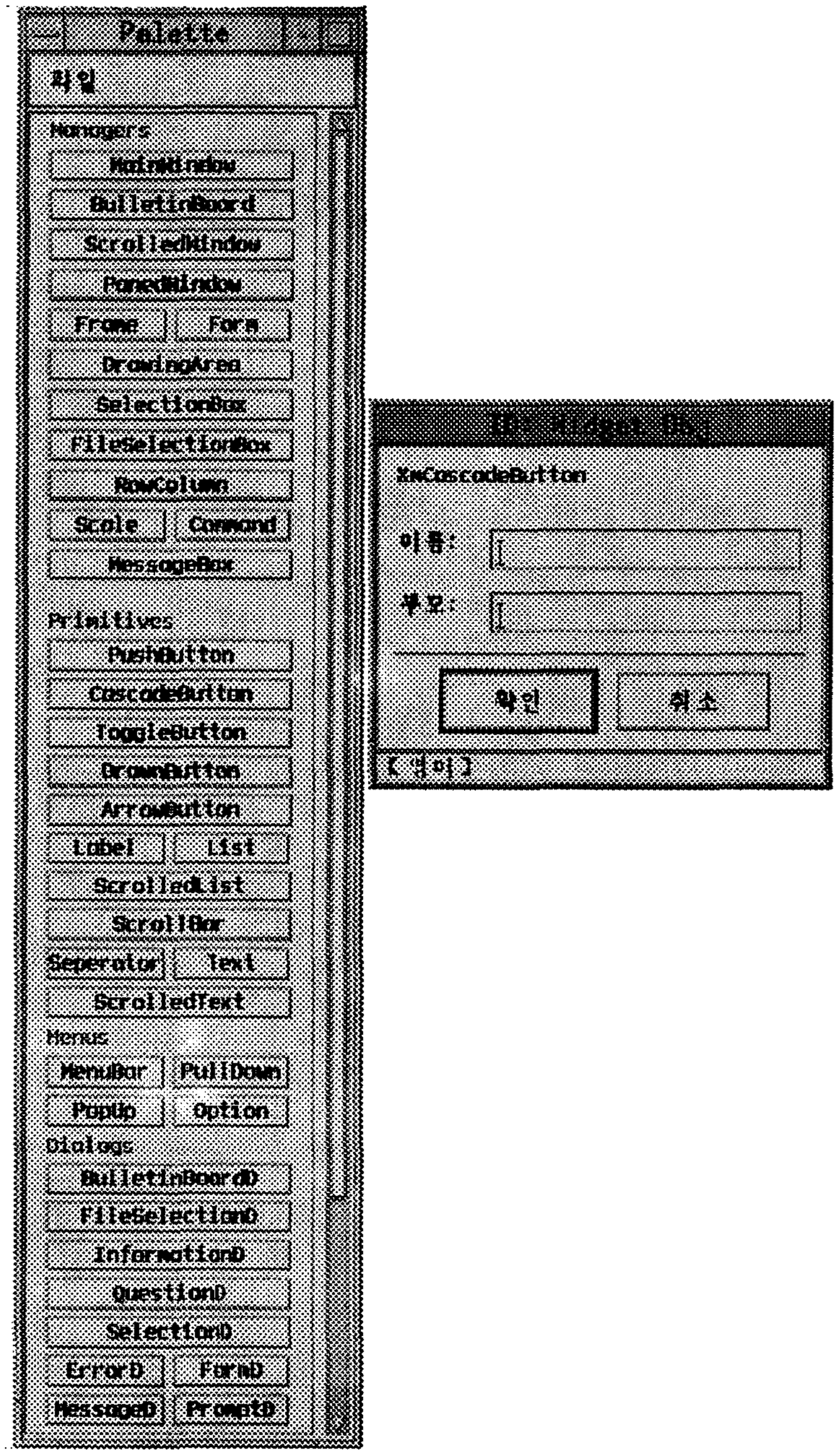
작업결과의 저장을 위한 저장하기 기능은 트리로 구성되어 있던 프로젝트 및 인터페이스 정보를 명세정보로 생성하여 프로젝트 선택시 지정된 화일에 저장한다.

코드생성, 컴파일, 수행은 시스템의 자식프로세스를 생성(fork)하여 컴파일 시에 소요되는 시간에 구애됨이 없이 모자이크 시스템의 다른 기능을 사용할 수 있도록 하였다.

4. 위짓 합성기

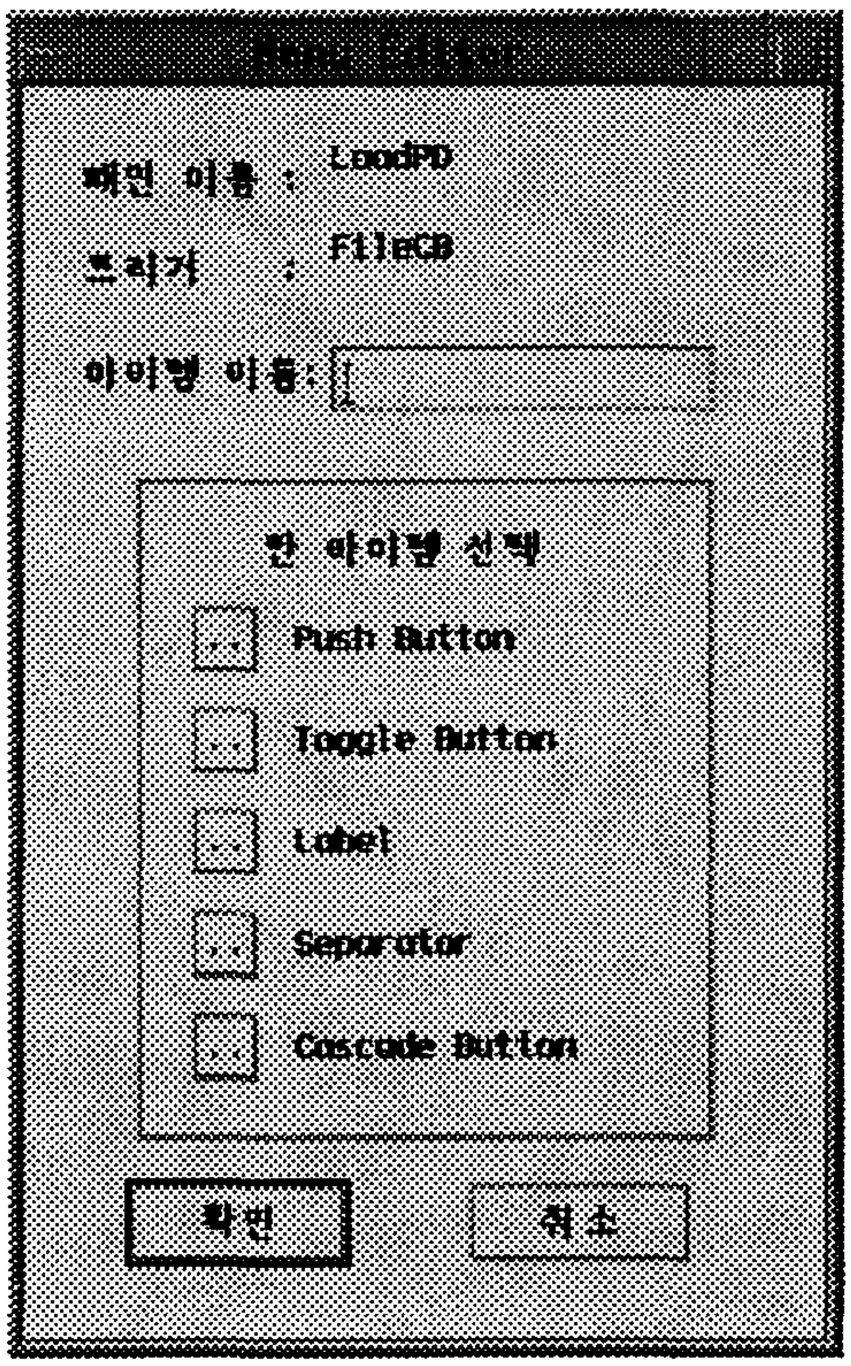
GUI모자이크 시스템은 위짓 합성기를 통하여 Motif 에서 제공하는 모든 위짓들을 합성을 고려한 특성별로 나누어 5개 그룹으로 분류하였다. 매니저 그룹에는 여러 위짓들을 포함할 수 있는 MainWindow, BulletinBoard, Form 위짓들과, 여러 위짓들로 구성되어 있는 SelectionBox, Scale, MessageBox 위짓들을 포함하였다. 프리미티브 그룹에는 다른 위짓(Dialog 위짓 제외) 들을 자식으로 가질 수 없는 위짓들, PushButton, CascadeButton, Label, List 위짓 들을 포함하였다. 메뉴 그룹은 Motif 의 위짓 분류에는 Dialog 위짓이나, 메뉴를 합성하는 방법 및 절차상의 특성을 고려하여 메뉴 그룹으로 추출하였다. 메뉴 그룹에는 Pulldown, Popup 및 OptionMenu 와 MenuBar 위짓들을 포함하였다. 다이알로그 그룹과 가젯

(Gadget) 그룹에는 Motif 에서 제공하는 모든 Dialog 위짓과 Gadget 위짓들을 모두 포함하고 있다.



[그림 4-2-5] 위짓 합성기

Motif 의 분류에는 Shell 위짓도 있으나, 코드생성기를 통하여 생성되는 원시 코드에서 채택된 위짓생성함수들은 위짓생성시 Shell 이 요구되는 위짓의 경우에는 Shell 도 동시에 생성해 주므로 중복되는 경향을 제거하기 위하여 Shell의 선택은 하지 않도록 합성기에서 제외하였다.



[그림 4-2-6] 메뉴편집기

위짓합성기를 구성하고 있는 위짓의 PushButton 마다 activateCallback을 부여하여 위짓의 이름과 부모를 입력할 위짓이름입력기를 팝업하도록 하였다. 위짓

이름입력기는 입력된 위짓의 이름과 부모 이름을 입력받아 같은 이름의 위짓이 있는지, 입력한 부모의 이름이 트리에 존재하고 있는지를 검사하고, 적합한 경우에는 사용자가 원하는 크기의 위짓을 그릴 수 있도록 준비한다. 위짓합성기는 사용자에 의하여 작성된 위짓의 스크린 정보와 위짓이름 입력기를 통하여 입력된 이름등을 트리에 저장한다.

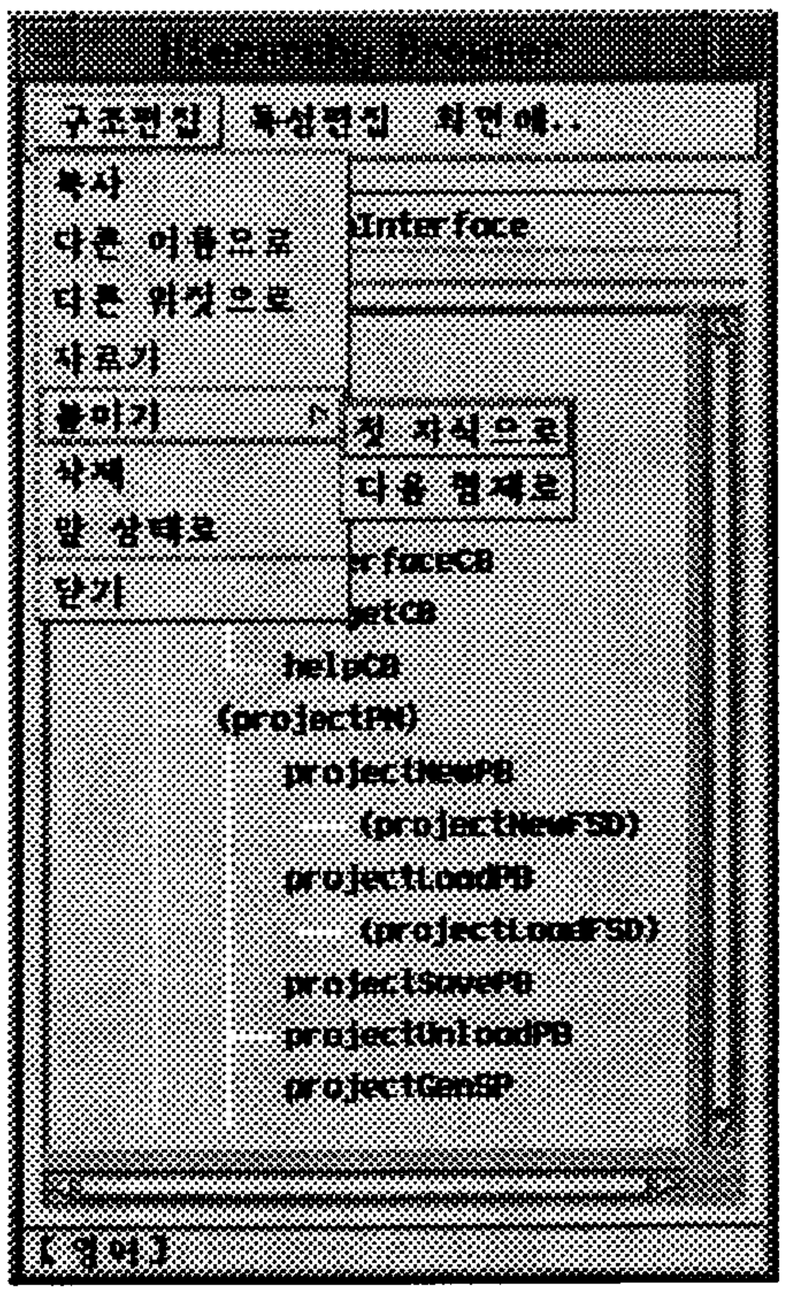
메뉴는 MenuPane에 자식위짓들을 배열하는 것으로써 그 메뉴를 팝업 시켜주는 트리거(Trigger), 즉 CascadeButton 위짓을 필요로 하며, 트리거와 MenuPane은 같은 부모를 가져야 한다. 자식위짓의 종류에는 PushButton, CascadeButton, ToggleButton, Label 위짓 등이 있으며, MenuPane을 자식위짓으로 구성할 수도 있다. 이와 같은 특성을 지닌 메뉴의 합성을 위하여 메뉴편집기를 별도로 구성하였다. 메뉴편집기는 이와 같은 메뉴편집의 특성을 고려하여 구성되었다. 메뉴편집기는 메뉴 그룹의 위짓을 선택하고, 위짓이름 입력기에서 이름과 부모를 입력한 후 확인하면 팝업되도록 하였다.

5. 구조편집기

프로젝트의 모든 정보를 담고 있는 트리에는 모든 위짓들간의 관계에 관한 정보가 들어 있다. 이 부모자식관계를 이용하여 인터페이스를 구성하고 있는 모든 위짓들의 구조를 보여줄 수 있다. 위짓구조는 위짓의 이름은 PushButton으로, 위짓의 관계는 선(White Line)으로 표현하여 트리모양을 취하였으며, 화면에 보여지는 위짓과 보여지지 않는 위짓을 ()를 이용하여 구별하였다. 각 PushButton 마다 activateCallback 을 부여하여 클릭시에는 PushButton의 그림자(shadowThickness)두께를 변경하여 그 위짓이 선택된 상태라는 것을 나타내도록 하였다. 구조편집기는 ScrolledWindow로 구성되어 있으므로 구조가 큰 인터페이

스도 ScrollBar 를 이용하여 조회가 가능하다.

구조편집기는 구조 조회와 더불어 구조와 위짓상태, 위짓속성 등을 변경할 수 있도록 구성되었다. 구조의 변경에는 위짓의 부모를 변경하는 것(붙이기)과, 복사, 삭제등이 있고, 위짓 상태 변경에는 다른 이름으로의 변경과 다른 위짓으로의 변경, 화면에 나타낼 것인가 화면에 나타내지 말 것인가 등을 들 수 있다.



[그림 4-2-7] 구조편집기

위짓의 속성에는 위짓의 모양, 색상, 크기, 라벨 등 눈에 보여지는 모습과 관련된 속성이 있고, 위짓의 행동과 관련된 속성이 있다. 위짓의 행동과 관련된 속성중 콜백은 모습과 관련된 속성을 편집할 수 있는 자원편집기에 포함하였고, 응용프로그램과 밀접한 관련이 있는 이벤트와 행위번역(action/translation)등은 각각을 위한 편집기를 별도로 구성하여 응용코드 편집을 지원할 예정이나 이번 프로토타잎에서는 반영하지 않았다.

사용자에 의하여 변경된 정보는 트리에 반영되며 그와 동시에 트리정보가 다시 해석(Interpreting)되어 사용자에게 보여질 수 있도록 하였다.

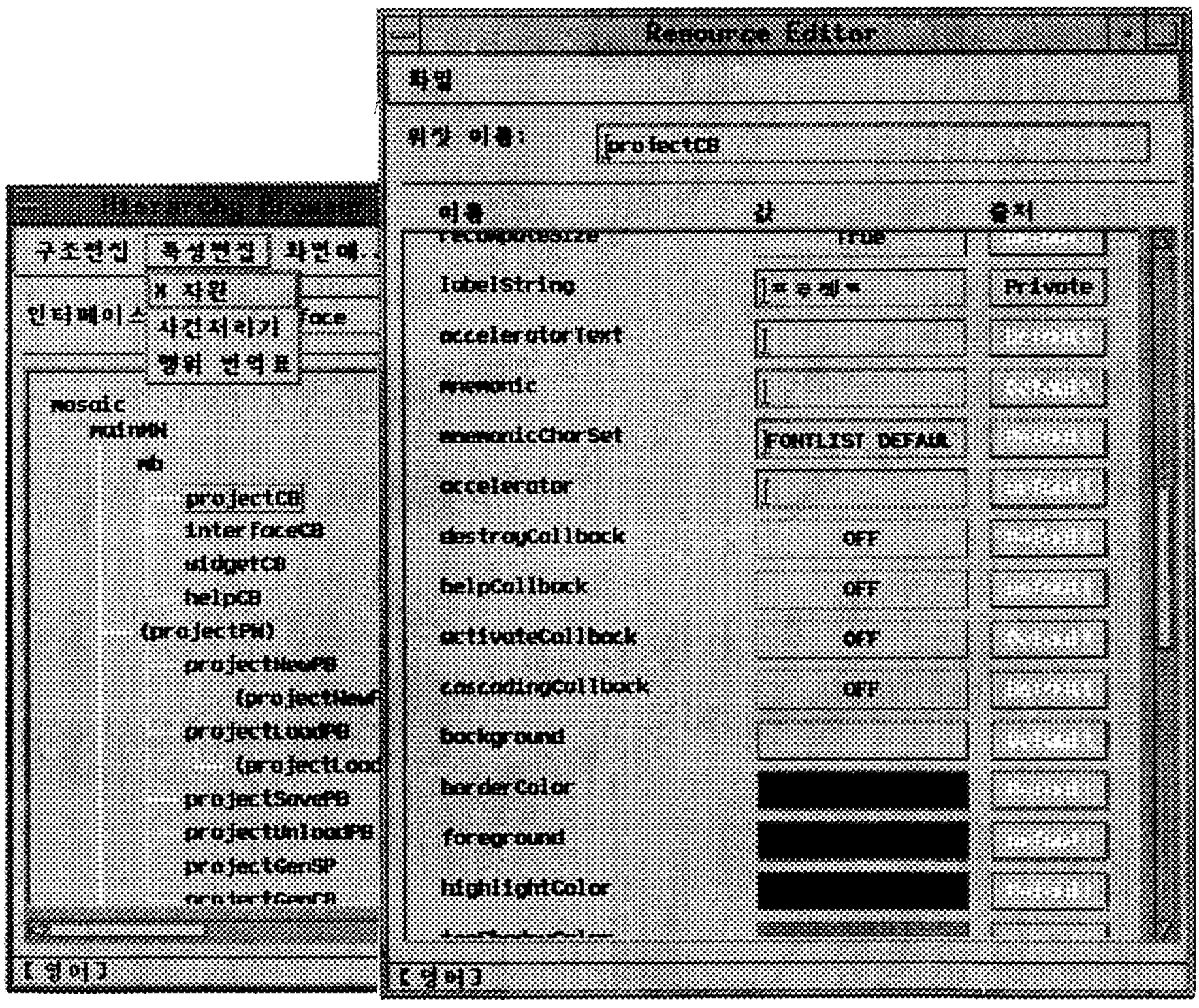
6. 자원편집기

모든 위짓은 각기 속성을 가지고 있고, 이 속성들은 변경이 가능하다. 자원편집기의 전체구성은 속성의 이름과 속성 값의 표현 및 입력, 변경여부의 표현으로 구성된다. 편집기 상에서 각 속성의 이름은 Label위짓을 이용하여 표현하였고 변경여부는 PushButton을 이용하여 나타내었다.

속성값의 표현 및 입력을 위하여 속성을 타입별로 분류하였다. x, y, width, height, shadowThickness, labelString 등 숫자나 문자는 직접 값을 입력해야 하는 속성들이며, 전경, 배경색 등 색상은 보여지는 X Window 에서 제공하는 색상표로 부터 색상을 선택할 수 있다. alignment, scrollBarDisplayPolicy, visualPolicy, rowColumnType 등은 미리 정의된 값들 중에서 선택하도록 되어 있다. 또한 recomputeSize, fillOnArm 등과 같은 속성들은 True 또는 False의 Boolean 값을 원한다. 콜백은 콜백을 불렀을 경우(invoke)에 해당되는 절차(procedure)를 요구하므로 응용 코드를 입력할 수 있는 편집기가 필요하다.

이와 같이 위짓을 속성별로 분류하여 값을 표현하거나 변경할 값을 입력받을

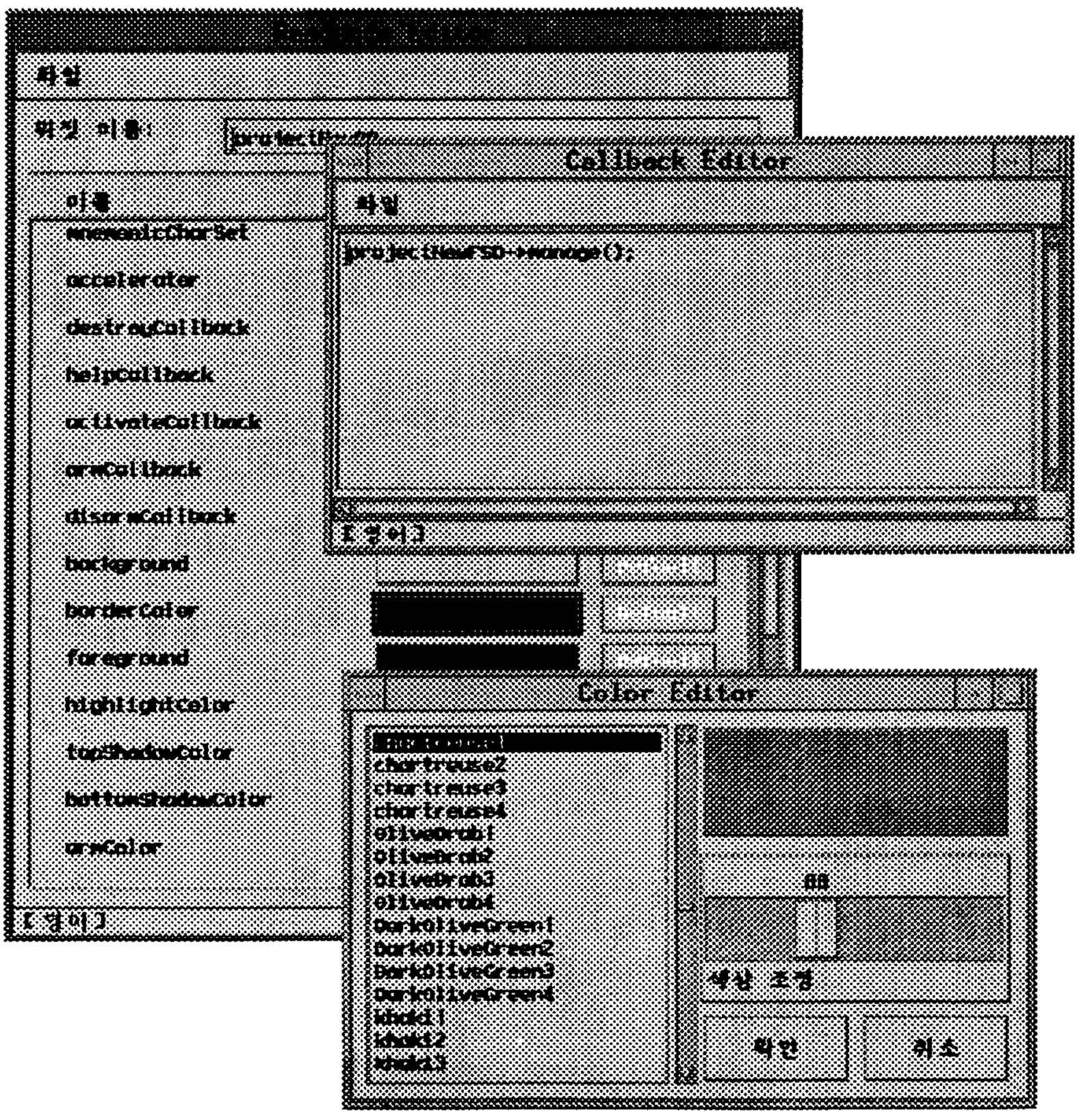
수 있다. 숫자나 문자는 Text 위짓을 사용하였고, Boolean 값의 표현을 위하여서는 PushButton을 이용하여 True와 False를 토글하도록 하였으며, 이미 정의된 (predefined) 값들을 위해서는 OptionMenu 위짓을 이용하여 그중에서 하나를 선택하도록 하였다. 한편 색상과 콜백은 색상편집기와 콜백편집기를 별도로 구성하여 색상은 PushButton 위짓의 배경색을 선택된 색상으로 표현하고 변경하고자 할 시에는 PushButton을 클릭하여 색상편집기를 팝업한다.



[그림 4-2-8] 자원편집기

콜백역시 PushButton을 이용하여 콜백정의 여부를 표현하였다. 콜백 절차(procedure)가 이미 정의되어 있는 경우에는 PushButton의 라벨값을 On 으로 지정하고, 정의되어 있지 않은 경우에는 Off 로 지정하여 나타내고, 콜백절차의 변경 또는 새로운 콜백의 입력을 위해서는 PushButton을 클릭하여 콜백편집기를 팝업시킨다.

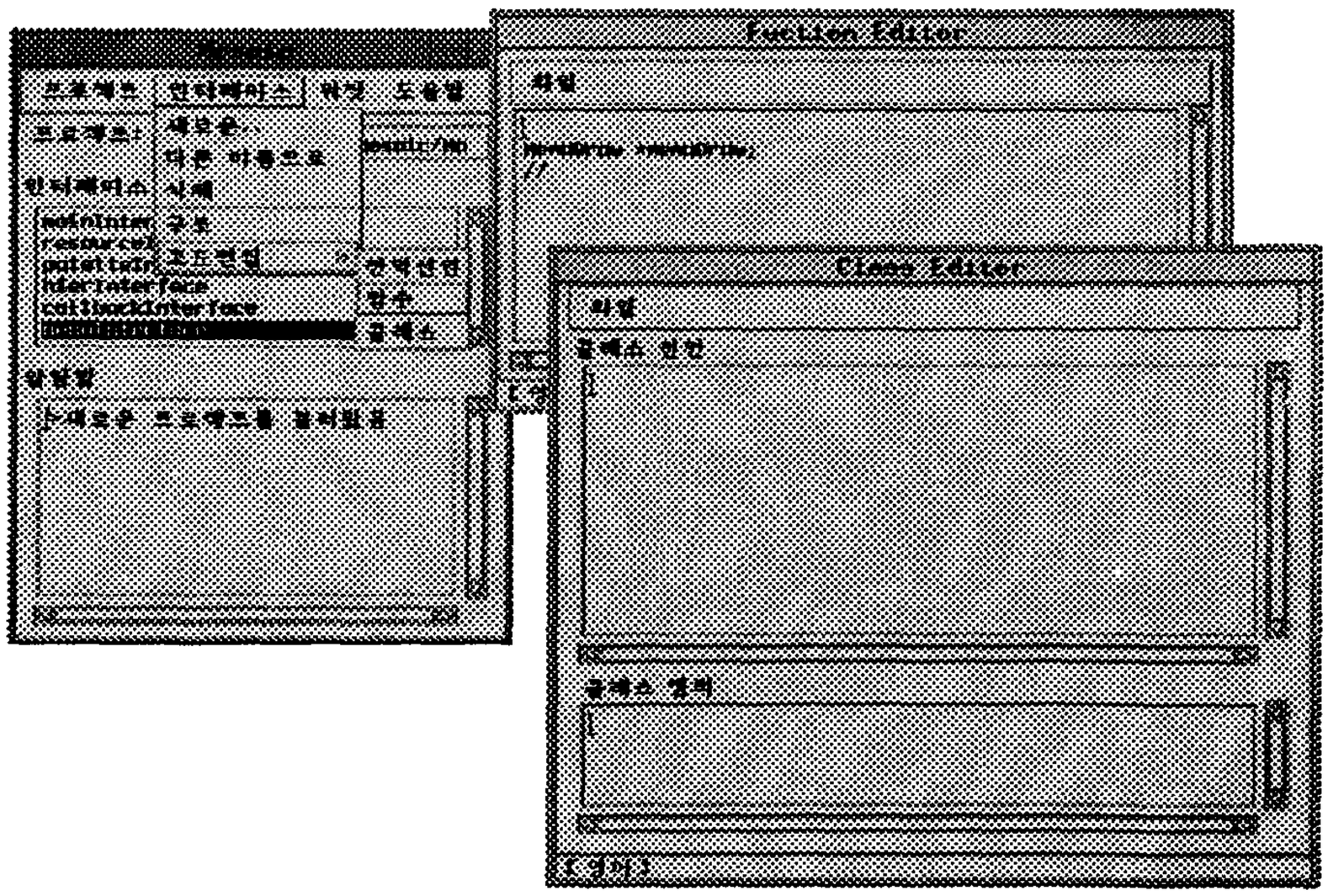
콜백편집기는 ScrolledText 를 이용하여 특별한 기능 없이 간단한 콜백절차를 기술할 수 있도록 만들어 졌는데, 트리에 콜백절차의 내용이 있는 경우에는 콜백편집기에 이 내용을 담아 사용자에게 보여준다.



[그림 4-2-9] 콜백편집기와 색상편집기

사용자 응용프로그램의 작성을 지원하기 위하여 전역선언(Global Declaration)부와 함수정의(Function Definition)부로 나누어 각기 에디터를 만들고, C++ 언어 사용자를 위하여 클래스 편집기를 만들어 사용자 클래스 선언과 정의를 따로 따로 편집할 수 있도록 하였다. 위짓 메뉴에서는 종류별로 나열된 위짓 선택판(palette)을 팝업(popup)할 수 있도록 하였다.

또한 프로젝트를 구성하는 인터페이스를 표현할 인터페이스 창은 ScrolledList 위짓을 이용하여 작성되었다. 리스팅되는 인터페이스는 프로젝트를 로드하거나 인터페이스 메뉴를 이용하여 인터페이스를 새로이 생성했을 경우 만들어지는 이름들로서, 원하는 인터페이스를 선택하면 트리에 저장되어 있는 인터페이스 명세 정보를 그래픽 정보로 변환하여 화면에 보여준다. 인터페이스의 선택은 scrolledList의 single selection callback을 이용하여 하나의 인터페이스 만을 선택하도록 하고 있다.



[그림 4-2-3] 함수 및 클래스 편집기

색상편집기는 색상을 색상이름, 색상값, 실제 색상의 세가지 형태로 반영한다. 색상이름은 ScrolledList 위짓을 이용하여 하나의 색상만을 선택하도록 하였으며, 색상값은 Scale 위짓을 이용하여 숫자로 나타내었다. 색상은 PushButton을 이용하여 색상이름 또는 색상값에서 선택된 색상을 배경색으로 지정하여 원하는 색상인지의 여부를 눈으로 직접 확인할 수 있도록 하였다.

자원편집기를 팝업하면 해당 위짓이 가지고 있는 속성을 먼저 나타내고, 사용자가 속성변경을 마치고 적용을 하면 이 값들은 트리에 반영이 되며, 다시 번역작업을 통하여 화면에 반영되어, 원하는 형태로 변경되었는지를 바로 확인할 수 있다.

7. 명세코드 생성기

시각적인 명세작업대, 즉 위짓합성기, 구조편집기, 자원편집기 등을 이용하여 생성된 사용자 인터페이스의 정보는 트리에 저장되어 있다가 명세코드로 변환되어 디스크에 화일의 형태로 저장된다. 트리가 가지고 있는 정보의 종류에 따라 명세코드도 프로젝트 화일, 인터페이스 화일, 위짓 화일로 나뉘어 저장된다.

명세코드생성기는 프로젝트 정보와 인터페이스 정보의 종류를 나타내는 클래스와 이름등으로 양식화(Formatting)하여 생성한다. 프로젝트 마다 하나씩의 프로젝트화일과 인터페이스 화일을 생성한다. 각 인터페이스 마다 여러 위짓들로 구성되어 있으므로 위짓정보는 인터페이스의 수 만큼의 화일로 생성할 수 있다. 각 위짓정보화일은 위짓의 종류를 나타내는 클래스와 위짓의 이름, 위짓의 부모, 위짓의 상태(보이기), 위짓의 속성 등 트리의 위짓정보를 인터페이스 명을 화일명으로 하여 정해진 양식에 의거하여 화일을 생성한다.

8. 명세 번역기

사용자의 작업을 즉각 반영할 수 있도록 지원해주는 것이 명세번역기(Spec Interpreter)로써 트리의 위짓정보를 이용하여 위짓들을 화면에 직접 크리에이션하고 그 주소(Widget id)를 별도의 트리(Swidget)를 생성하여 관리한다. 인터페이스의 내리기(unload) 시에는 생성된 위짓들을 모두 파괴(unmanage & destroy)하고, Swidget의 트리도 모두 삭제한다.

작업절차관리기에서 인터페이스를 선택하는 경우와, 선택판을 이용하여 위짓을 합성하는 경우와, 구조조회기에서 구조를 변경하는 경우 및 자원편집기를 통하여 위짓의 속성을 변경. 적용할 경우에 트리가 변경되는데 트리가 변경될 때마다 Swidget의 트리도 변경이 되고 명세번역기도 번역을 실행한다.

명세번역기는 위짓의 생성과 함께 속성도 함께 반영하는데 위짓의 실제 주소 즉, subMenuId나 위짓의 주소가 요구되는 attachment값을 부여해야하는 경우에는 위짓이 먼저 생성되어야 하므로 XtSetValues()를 통하여 생성 후에 속성을 부여하고, 그 이외의 속성들은 모두 위짓생성시에 속성 부여를 한다. 단, 명세번역기는 콜백과 같이 컴파일 후 실행시에 반영되는 행동(behavior) 관련 속성들은 번역하지 않으므로 Swidget 트리에 반영하지 않는다.

9. 코드 생성기

코드생성기는 사용자 인터페이스의 명세정보를 입력으로 하여 최종 컴파일이 가능한 원시코드를 생성한다. 프로젝트명과 패스(path)는 향후 생성될 메인화일, X 자원화일, 메이크화일 등의 물리적인 화일명으로 사용된다.

인터페이스 객체가 가지고 있는 정보를 이용하여 다수의 원시코드를 생성한다.

인터페이스 객체는 사용자가 기술한 응용코드들을 가지고 있다.이 응용코드들은 각기 선언부와 정의부로 구별되어 선언부는 헤더화일로, 정의부는 원시코드화일로 생성한다.

C++ 코드 생성의 경우에는 코드생성기에 의해 자동으로 생성되는 클래스 이외에 사용자가 자신의 응용프로그램을 위해서 만든 클래스의 선언 및 정의도 이 헤더화일과 원시코드화일로 각각 생성한다.

위짓 객체는 원시코드 생성에서 가장 중요한 핵심부분으로서 각 인터페이스를 구성하고 있는 위짓들의 구조와 관련된 정보들을 가지고 있으며, 각각의 위짓 객체가 가지고 있는 특성과 행동에 관한 정보들을 가지고 있다. 위짓객체의 특성은 다시 원시코드에 반영되는 특성과, X 자원화일로 들어가는 특성으로 구별된다.

생성되는 원시코드의 골격은 이 위짓객체가 가지고 있는 구조와 관련된 정보 즉 각 위짓간의 관계에 의해 구성된다.

생성되는 원시코드의 기본 모형은 다음과 같다.

```
class [object_name]_Class {
        Widget _w;
        char *_name;
        [chld_object_name]_Class *chld_object_name;
        ....
public:
        [object_name]_Class(Widget);
        void createWidget(Widget, char *);
        void manage()       { XtManageChild(_w); }
        void unmanage()     { XtUnmanageChild(_w); }
```

```
        void callback(Widget, XtPointe7r, XtPointer);
};
void [object_name]_Class::createWidget(Widget parent, char *name)
{
        n = 0;
        XtSetArg(args[n], XmN[property_name], [property_value]); n++;
        _w = XmCreate[class_name](parent, name, args, n);
        [child_object_name] = new [child_object_name]_Class(_w,
                "[child_object_name]");
        [child_object_name]->manage();
}
```

C 코드생성기는 C++ 코드생성기와 마찬가지로 명세코드를 입력받아 C 원시코드 및 메이크화일, 자원화일 등을 생성한다. C 원시코드는 클래스의 생성 및 멤버선언 등이 없으므로 위짓의 구조 부분과 관계되는 코드의 설계가 비교적 간단하다.

먼저 위짓을 생성(creation)하는 함수 정의와 행위(behavior) 의 정의부로 구성한다. 단위 위짓별로는 위짓구조 및 위짓간의 관계 설정 등을 고려하여 속성설정, 생성, 관리(manage), 콜백(callback)을 하나의 쌍으로 구성한다. C++ 코드와 마찬가지로 인터페이스 별로 applicationShell을 생성하여 다중 toplevel 프로그램을 가능하도록 한다.

콜백 함수는 위짓생성모듈을 완료하고 콜백타입별로 모듈을 구성한다. 명세코드의 이름(name)은 C 원시코드의 유일한 이름으로 정의한다.

생성되는 C 원시코드의 모형은 다음과 같다.

```
void [interface_name]Create()
{
...
Widget [obj_name];
XtSetArg(args[n], XmN[property_name], [property_value]); n++;
[obj_name] = XmCreate[class_name]([parent_name], "[obj_name]", args, n);
XtManageChild([obj_name]);
XtAddCallback([obj_name], XmN[callbackType], [callback_func], NULL);
...
}
int [callback_func](Widget,XtPointer,XtPointer)
{
...
}
```

main 프로그램은 사용자 프로그램을 가동하기 위해 요구되는 그래픽 사용자 인터페이스 툴킷을 초기화하는 부분과 모자이크 시스템을 통해서 합성한 인터페이스를 구동시키는 부분으로 구성된다. 즉, 응용프로그램과 오퍼 레이팅 시스템과의 인터페이스를 위해 툴킷을 초기화 하고, 참조할 X 자원 화일을 정의한다.

사용자가 작성한 모든 인터페이스를 구동하는 부분은 C++ 코드는 인터페이스 객체를 new로 생성하고, C 코드는 개별 인터페이스를 각각 크리에이션하는 함수를 호출한다.

인터페이스 단위로 각각 화일이 생성되며 인터페이스 명이 화일명으로 사용된다. C++코드는 C 코드와의 구별을 위하여 .C 를 확장자로 사용한다. 인터페이스

명이 menu 인 경우 C 코드화일은 menu.c, menu.h 이며, C++ 코드화일의 경우에는 menu.C.C, menu.C.H 이다. 이는 C++ 코드의 경우 컴파일 과정 중 중간화일로 .c 화일이 생성되므로 혼동을 피하기 위함이다.

각 인터페이스 별로 프로그램의 구조는 같으며, 위짓의 생성(creation), 관리(manage), 속성부여(set value), 행위처리(callback)로 구성된다. 각 인터페이스 별로 applicationShell이 생성되므로 다중 toplevel 윈도우를 가능하게 한다.

C++ 원시코드의 경우에는 하나의 위짓이 하나의 객체로 1 대 1 대응이 되고 객체 내에는 그 위짓을 부모로 하는 모든 자식 위짓들의 객체가 멤버변수로 정의된다.

X 또는 Motif에서 제공되는 위짓들은 모양이나 색상 등의 속성을 위해서 미리 정해진 값들이 있다. 사용자 임의의 속성값을 지정하고자 할 경우에는 응용자원 화일을 이용하는 방법이 있다. 명세코드에서 원시코드로 변환되는 속성을 제외한 모든 속성들은 이 X 자원 코드화일에 저장된다.

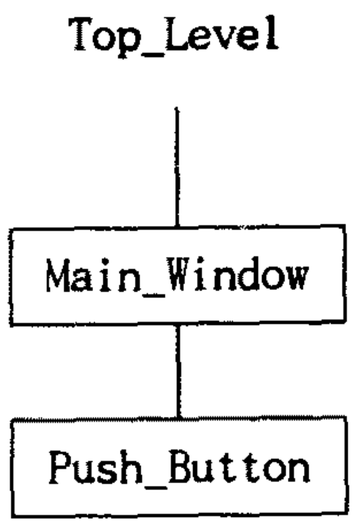
메이크화일은 생성되는 모든 화일의 이름들로 구성되며, 컴파일 시에 요구되는 플래그(flag) 및 링크시에 요구되는 라이브러리들을 자동으로 생성한다. 자동생성되는 원시코드들이 Motif와 X 툴킷 함수들을 사용하고 있으므로 Motif, Xt, X11 등의 라이브러리 등을 포함하며, 사용자 라이브러리는 메이크화일 생성후 에디터를 이용하여 추가할 수 있다.

제 3 절 적용 사례

1. Hello, GUI Mosaic

모자이크 시스템에서 적용하는 사례로 메인윈도우에 'Hello, GUI Mosaic' 이

라는 라벨을 갖는 버튼(Push_Button)을 생성하는 예로써 [그림 4-3-1]은 위짓의 구조를 나타낸 그림이며 [그림 4-3-2]는 모자이크로 작성한 인터페이스의 실행 화면이고 [그림 4-3-3]은 프로젝트에 대한 명세 코드 리스트이며 [그림 4-3-4]는 응용 객체에 대한 명세 코드를 나타내주며 [그림 4-3-5]에서는 계면 위짓 객체군을 표현한 명세 코드이다.



[그림 4-3-1] 위짓 구조도

[그림 4-3-2] Hello, GUI Mosaic 실행화면

```
#
*Hello_Mo.class:          Project
*Hello_Mo.name: Hello_Mo
*Hello_Mo.path: /usr1/phm/temp/report94
*Hello_Mo.ifFileName:    Hello_Mo.if
```

[그림 4-3-3] 프로젝트 Hello_Mo 명세코드

```
#
*Hello_Mo_IF.class:       Interface
*Hello_Mo_IF.name:        Hello_Mo_IF
*Hello_Mo_IF.parent:      Hello_Mo
*Hello_Mo_IF.bodyFileName:        Hello_Mo_IF.bd
```

[그림 4-3-4] 응용객체 Hello_Mo_IF 명세코드

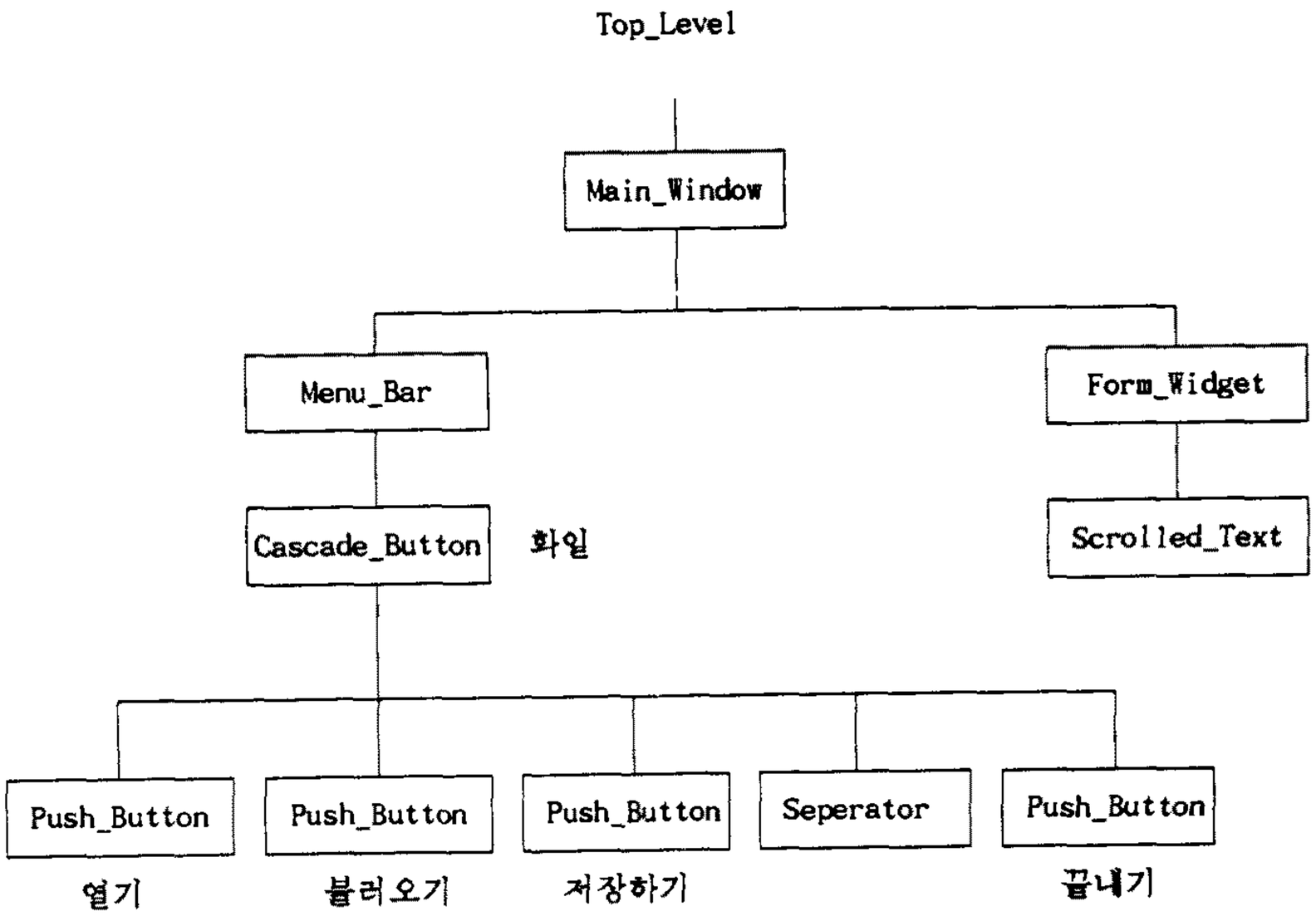
```
#
*Hello_Mo_IFTOP.class:   applicationShell
*Hello_Mo_IFTOP.name:    Hello_Mo_IFTOP
*Hello_Mo_IFTOP.parent:  Hello_Mo_IF
*Hello_Mo_IFTOP.static:  1
*Hello_Mo_IFTOP.x:       638
*Hello_Mo_IFTOP.y:       303
*Hello_Mo_IFTOP.y:       303

#
*Hello_If_Main.class:    XmMainWindow
*Hello_If_Main.name:     Hello_If_Main
*Hello_If_Main.parent:   Hello_Mo_IFTOP
*Hello_If_Main.static:   1
*Hello_If_Main.x:        638
*Hello_If_Main.y:        303
*Hello_If_Main.width:    295
*Hello_If_Main.height:   152
```

```
#
*Hello_If_PB1.class:     XmPushButton
*Hello_If_PB1.name:      Hello_If_PB1
*Hello_If_PB1.parent:    Hello_If_Main
*Hello_If_PB1.static:    1
*Hello_If_PB1.x:         6
*Hello_If_PB1.y:         7
*Hello_If_PB1.width:     283
*Hello_If_PB1.height:    136
*Hello_If_PB1.background:        AntiqueWhite2
*Hello_If_PB1.borderColor:       black
*Hello_If_PB1.foreground:        black
*Hello_If_PB1.highlightColor:    black
*Hello_If_PB1.topShadowColor:    #bfbfb3b3a3a3
*Hello_If_PB1.bottomShadowColor:         #83837b7b7070
*Hello_If_PB1.labelString:       Hello, GUI Mosaic
*Hello_If_PB1.armColor: #cbcbbebeaeae
```

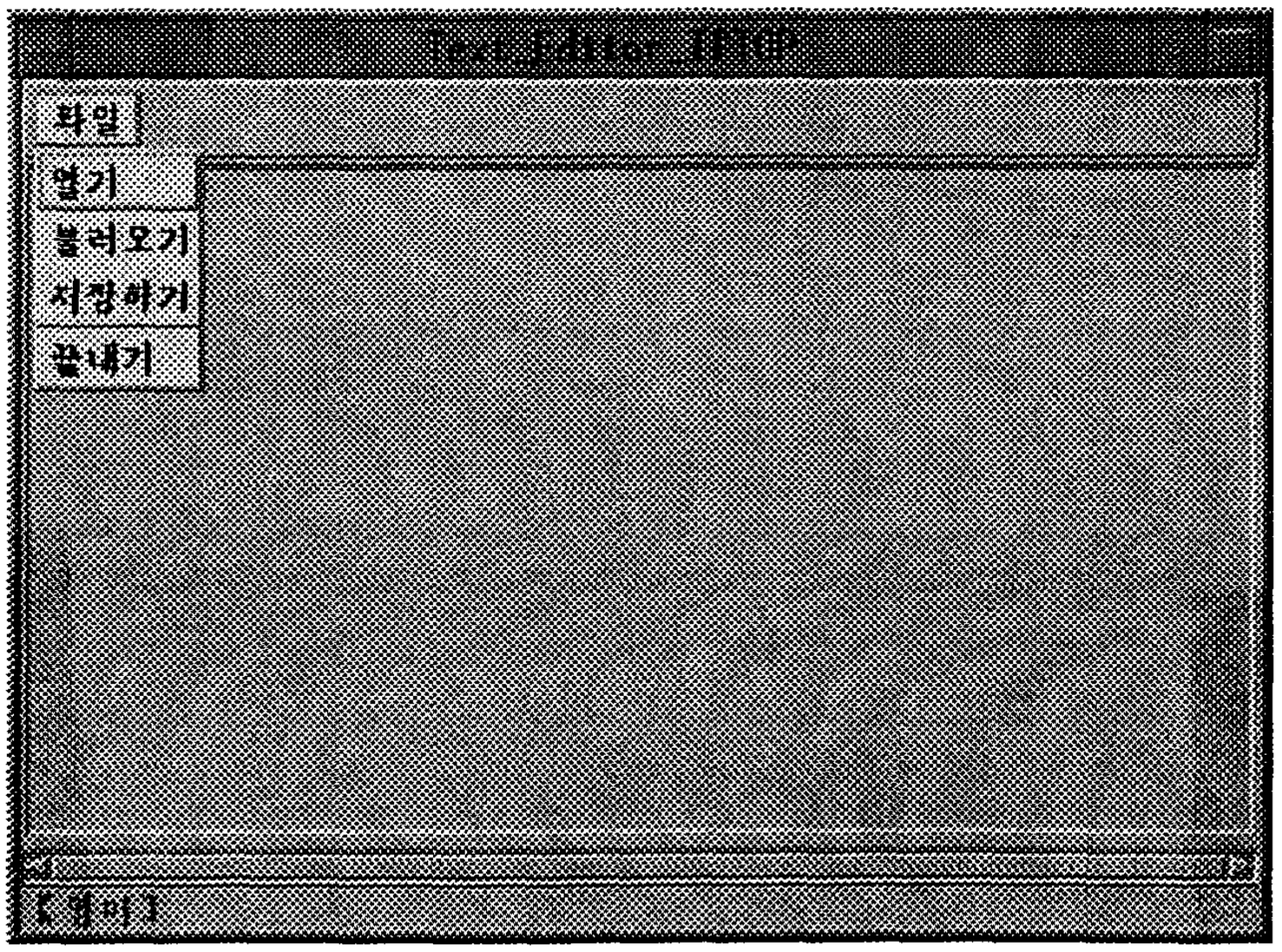
[그림 4-3-5] 응용객체 Hello_Mo_IF의 계면요소 객체군의 명세코드

2. 텍스트 에디터(Text Editor)

두번째 사례로는 Motif 형식의 기본적인 텍스트 에디터에 대한 인터페이스를 작성하는 예로써 메인 윈도우에 메뉴바(menu bar) 및 Scrolled Text 위짓을 생성하며 메뉴바에는 '화일' 메뉴를 생성하고 자식으로 '열기', '불러오기', '저장', '끝내기' 버튼을 생성하며, 스크롤 텍스트의 부모 위짓은 Form 위짓을 사용하여 텍스트 위짓은 이 Form 위짓에 붙이는(attachment) 것으로 가정한다. [그림 4-3-6]은 위짓의 구조를 나타내는 그림이며, [그림 4-3-7]은 생성된 인터페이스의 화면을 나타낸 그림이다. [그림 4-3-8]은 프로젝트에 대한 명세코드 리스트이며 [그림 4-3-9]는 응용 객체에 대한 명세코드를 나타내며 [그림 4-3-10]는 계면 위짓 객체군의 리스트이다.



[그림 4-3-6] 위짓 구조도



[그림 4-3-7] 텍스트 에디터 실행화면

```
#
*Text_Editor.class:       Project
*Text_Editor.name:        Text_Editor
*Text_Editor.path:        /usr1/phm/temp/report94
*Text_Editor.ifFileName:          Text_Editor.if
```

[그림 4-3-8] 프로젝트 Text_Editor 명세코드

```
#
*Text_Editor_If.class:  Interface
*Text_Editor_If.name:   Text_Editor_If
*Text_Editor_If.parent: Text_Editor
*Text_Editor_If.bodyFileName:   Text_Editor_If.bd
```

[그림 4-3-9] 응용객체 Text_Editor_If 명세코드

```
#
*Text_Editor_IfTOP.class:         applicationShell
*Text_Editor_IfTOP.name:          Text_Editor_IfTOP
*Text_Editor_IfTOP.parent:        Text_Editor_If
*Text_Editor_IfTOP.static:        1
*Text_Editor_IfTOP.x:    624
*Text_Editor_IfTOP.y:    288
*Text_Editor_IfTOP.y:    288
#

*Text_Editor_Main.class:          XmMainWindow
*Text_Editor_Main.name: Text_Editor_Main
*Text_Editor_Main.parent:         Text_Editor_IfTOP
*Text_Editor_Main.static:         1
*Text_Editor_Main.x:     624
*Text_Editor_Main.y:     288
*Text_Editor_Main.width:          605
*Text_Editor_Main.height:         449

#
*Text_Editor_MB.class:   MoMenuBar
*Text_Editor_MB.name:    Text_Editor_MB
*Text_Editor_MB.parent: Text_Editor_Main
```

```
*Text_Editor_MB.static: 1

#
*Editor_CB.class:          XmCascadeButton
*Editor_CB.name:           Editor_CB
*Editor_CB.parent:         Text_Editor_MB
*Editor_CB.static:         1
*Editor_CB.x:      19
*Editor_CB.y:      1
*Editor_CB.width:          20
*Editor_CB.height:         20
*Editor_CB.background:     AntiqueWhite2
*Editor_CB.borderColor:    black
*Editor_CB.foreground:     black
*Editor_CB.highlightColor:         black
*Editor_CB.topShadowColor:         #bfbfb3b3a3a3
*Editor_CB.bottomShadowColor:      #83837b7b7070
*Editor_CB.labelString:    화일
*Editor_CB.subMenuId:      MB_FilePD

#
*MB_FilePD.class:          MoPulldownMenu
*MB_FilePD.name:           MB_FilePD
*MB_FilePD.parent:         Text_Editor_MB
*MB_FilePD.static:         0

#
*File_OpenPB.class:        XmPushButton
*File_OpenPB.name:         File_OpenPB
*File_OpenPB.parent:       MB_FilePD
*File_OpenPB.static:       1
*File_OpenPB.background:           AntiqueWhite2
*File_OpenPB.borderColor:          black
*File_OpenPB.foreground:           black
*File_OpenPB.highlightColor:       black
*File_OpenPB.topShadowColor:       #bfbfb3b3a3a3
*File_OpenPB.bottomShadowColor:    #83837b7b7070
*File_OpenPB.labelString:          열기
*File_OpenPB.mnemonicCharSet:      FONTLIST_DEFAULT_TAG_STRING
*File_OpenPB.armColor:     #cbcbbebeaeae

#
*File_LoadPB.class:        XmPushButton
```

```
*File_LoadPB.name:          File_LoadPB
*File_LoadPB.parent:        MB_FilePD
*File_LoadPB.static:        1
*File_LoadPB.background:            AntiqueWhite2
*File_LoadPB.borderColor:           black
*File_LoadPB.foreground:            black
*File_LoadPB.highlightColor:        black
*File_LoadPB.topShadowColor:        #bfbfb3b3a3a3
*File_LoadPB.bottomShadowColor:     #83837b7b7070
*File_LoadPB.labelString:           불러오기
*File_LoadPB.mnemonicCharSet:       FONTLIST_DEFAULT_TAG_STRING
*File_LoadPB.armColor:      #cbcbbebeaeae

#
*File_SavePB.class:         XmPushButton
*File_SavePB.name:          File_SavePB
*File_SavePB.parent:        MB_FilePD
*File_SavePB.static:        1
*File_SavePB.background:            AntiqueWhite2
*File_SavePB.borderColor:           black
*File_SavePB.foreground:            black
*File_SavePB.highlightColor:        black
*File_SavePB.topShadowColor:        #bfbfb3b3a3a3
*File_SavePB.bottomShadowColor:     #83837b7b7070
*File_SavePB.labelString:           저장하기
*File_SavePB.mnemonicCharSet:       FONTLIST_DEFAULT_TAG_STRING
*File_SavePB.armColor:      #cbcbbebeaeae

#
*File_SaveSP.class:         XmSeparator
*File_SaveSP.name:          File_SaveSP
*File_SaveSP.parent:        MB_FilePD
*File_SaveSP.static:        1

#
*File_QuitPB.class:         XmPushButton
*File_QuitPB.name:          File_QuitPB
*File_QuitPB.parent:        MB_FilePD
*File_QuitPB.static:        1
*File_QuitPB.background:            AntiqueWhite2
*File_QuitPB.borderColor:           black
*File_QuitPB.foreground:            black
*File_QuitPB.highlightColor:        black
```

```
*File_QuitPB.topShadowColor:     #bfbfb3b3a3a3
*File_QuitPB.bottomShadowColor: #83837b7b7070
*File_QuitPB.labelString:        끝내기
*File_QuitPB.mnemonicCharSet:    Callback
*File_QuitPB.armColor:   #cbcbbebeaeae
*File_QuitPB.activateCallback:   exit(0);

#
*Editor_ST.class:        MoScrolledText
*Editor_ST.name:         Editor_ST
*Editor_ST.parent:       Text_Editor_Main
*Editor_ST.static:       1
*Editor_ST.x:    6
*Editor_ST.y:    36
*Editor_ST.width:        593
*Editor_ST.height:       404
```

[그림 4-3-10] 응용객체 Text_Editor_If의 계면요소 객체군의 명세코드

여 백

제 5 장 결 론

여 백

제 5 장 결 론

제 1 절 연구의 결과

본 연구는 국산 주전산기용 소프트웨어 개발 지원도구인 들무새 시스템 개발 과제의 세부과제로 총 2년중에서 당해 년도는 1차년도이다.

들무새 시스템은 통합 사용자 계층, 도구 계층, 저장소 서비스 계층, 그리고 물적 저장 계층 등의 네개의 계층으로 구성되었으며, GUI Mosaic은 도구 계층에 속한다. GUI Mosaic은 ODF/Motif 스타일의 GUI 개발을 지원하며, 확장된 개념으로는 응용 소프트웨어의 개발을 지원하는 도구로 볼 수 있다. 그러나 GUI의 개발은 그래픽 명세화를 기반으로 지원하여 C/C++ 원시코드를 자동으로 만들어 주나 응용 프로그램의 경우는 사용자가 직접 코딩 작업을 수행하는 작업을 거쳐야 한다.

1차년도인 당해년도에는 도구의 프로토타잎 개발에 주력하였으며, 이 결과 프로토타잎 개발에 성공하였으며, 도구가 갖추어야할 기술적인 요소들을 파악하게 되었으며, 또한 2차년도의 실용 버젼 개발의 기반을 구축하게 되었다.

GUI Mosaic은 분류상 시각 프로그래밍 도구에 속할 수 있으며, 또한 자동화 도구로도 분류가 가능하다. 이러한 분류는 도구의 구성 요소의 성격으로 부터 기인한다. GUI Mosaic의 주요 구성 요소는 사용자 작업 절차 관리기, 시각 사용자 명세 작업대, 정형적 명세어, 명세 변환기, 명세 구현기, 그리고 원시코드 생성기 등이다.

사용자 작업 절차 관리기는 사용자의 작업이 체계적으로 이루어 질 수 있도록

안내하는 기능을 수행한다. 사용자는 이 기능을 통하여 작업 순서의 오류에 의한 소프트웨어 개발 시간의 연장을 예방할 수 있는 효과를 얻는다. 시각 사용자 명세 작업대는 이 도구가 시각 프로그래밍 도구로 분류될 수 있는 특징을 제공하는 지원기로서, 합성의 방법을 중심으로 다양한 시각적인 기능들을 제공한다. 정형적 명세어는 시각 작업의 결과를 텍스트로 표현할 수 있게 정의된 언어로 간편하게 정보를 관리할 수 있는 방법을 제공한다. 이 언어를 이용할 수 있도록 지원기로 명세 변환기, 명세 구현기가 있다. 명세 변환기는 일종의 파서와 코드 생성기로 구성되어 있으며, 구현기는 명세의 내용을 화면에 구현하여 사용자에 의한 명세의 검증이 가능하도록 하는 지원기이다. 마지막으로 원시코드 생성기는 명세로부터 C 또는 C++ 원시코드를 만들어 내는 지원기로 명세를 완전한 프로그램으로 바꾸는 기능을 수행한다. 사용자는 이러한 다양한 도구들의 도움을 받아 C 또는 C++ 언어 및 Motif에 대한 큰 지식 없이도 원하는 GUI를 개발 할 수 있다.

GUI Mosaic 프로토타잎의 개발 외에 당해년도에 수행한 중요한 연구는 객체지향 개발 방법인 Rumbaugh의 OMT 방법으로 실용 버젼의 분석을 수행하였다는 것을 들 수 있다. 이 시도는 향후의 차세대 개발 방법론으로 각광을 받고 있는 객체지향 개발 방법론의 적용 기술력을 확보하기 위한 중요한 사례로 기대하고 있다. 이러한 개발 방법론은 GUI Mosaic 개발에 지속적으로 적용할 계획을 가지고 있다.

제 2 절 향후의 연구개발

당해 년도의 연구개발은 프로토타잎의 개발과 실용버젼의 분석을 중심으로 이루어 졌고, 다음 년도에는 실용 버젼의 설계 및 구현에 촛점이 맞추어 질 것이

다.

실용 버젼의 개발은 OMT 방법을 적용하며 구현은 C++를 이용할 것이다. 따라서 설계는 객체의 설계에 세심한 배려가 주어질 것이며, 또한 사용자의 이용 편의성 향상을 위한 사용자 접속부의 개발이 중요한 부분을 차지할 것이다. 연구 개발은 들무새 시스템의 GUI 개발에 당해 년도에 개발한 도구를 적용하여 보다 구체적인 사용자 요구 사항과 필요 기술을 세련화하는 효과를 얻어, 그 결과를 실용 버젼 개발의 근거 자료로 이용할 계획이다.

다음년도의 결과물은 GUI Mosaic 버젼 1.0, 사용자 지침서가 될 것이며, 기업체의 개발 인력을 보다 참여시켜 상품화를 위한 기술 이전도 함께 수행할 계획이다. 이와함께 상품화의 기반을 다지기 위한 시장 홍보 및 확보를 위한 기업의 활동이 활발히 진행될 것이다.

여 백

참고문헌

여 백

참고문헌

[1] 이경환 외, “소프트웨어 자동생산 기술에 관한 연구,” 연구보고서, 과학기술처, 1993.

[2] Sheldon, K.M., Barron J.J., and Smith B., "Window Wars," Byte, pp. 124-134, June 1991.

[3] Winblad A.L., Edwards S.D., and King D.R., "Object-Oriented Software," Addison-Wesley, 1990.

[4] Hess, D and Eibisch, J., "Graphical User Interfaces: An Overview," DATAPRO, pp. 101-110, Nov 1993.

[5] 이진우, “COSE/CDE,” 한국정보과학회, HCI, pp 43, 제3권 제1호, 1994.

[6] Hayes, F. and Baran, N., "A guide to GUIs," BYTE, pp. 250-257, July 1989.

[7] 이선미, 노영주, 박현민, 신규상, “GUI 도구 모자이크에서 시각적 명세화 방법,” 한국정보처리응용학회 논문집, pp. 195-198, 제1권 제1호, 1994

[8] Myers, B.A. "User Interface Tools: Introduction and Survey," IEEE Software, pp. 15-23, Jan 1989.

[9] 노영주, 이선미, 이형, “GUI Mosaic에서 위짓의 합성제어기법,” 한국정보처리응용학회 논문집, pp. 219-222, 제1권 제2호, 1994.

[10] Rumbaugh, J., et al, "Object-Oriented Modeling and Design," Prentice-Hall, 1991.

[11] 노영주, 이선미, 박현민, “Motif 표준을 지원하는 사용자계면 합성기 개발:

모자이크 모형," 한국정보과학회 논문집, pp. 825-828.제20권 제2호, 1993.

[12] 이선미, 노영주, "모자이크 시스템에서 컴파일 가능한 C++원시코드 생성기법," 한국정보과학회 논문집, pp. 829-832, 제20권 제2호, 1993

[13] OSF, "OSF/Motif Programmer's Guide," Prentice-Hall, 1990.

[14] OSF, "OSF/Motif Programming Manual," Prentice-Hall, 1990.

부록 A: 적용사례 코드

여 백

부록 A:적용사례 코드

1. Hello, GUI Mosaic

가. C++ 소스코드

```
//
//Hello_Mo.C.H
//

#ifndef Hello_Mo.C.H
#define Hello_Mo.C.H

#include <stdio.h>
#include <sys/signal.h>
#include <limits.h>
#include <assert.h>
#include <ctype.h>
#include <X11/Intrinsic.h>
#include <X11/IntrinsicP.h>
#include <X11/Shell.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/CutPaste.h>
#include "Mo_BaseWidget.h"
#include "Hello_Mo_IF_IF.C.H"
#include "Hello_Mo_IF.C.H"

#endif

//
//Hello_Mo.C.C
//

#include "Hello_Mo.C.H"

void main(int argc, char **argv)
```

```
{
        Widget toplevel;
        toplevel = XtInitialize(argv[0], "Hello_Mo.rc", NULL, 0, &argc, arg
v);
        Hello_Mo_IFTOP_Class *local_Hello_Mo_IFTOP = new Hello_Mo_IFTOP_Cla
ss(toplevel, "Hello_Mo_IFTOP");
        XtMainLoop();
}


//
//Hello_Mo_IF_IF.C.H
//

#ifndef Hello_Mo_IF_IF.C.H
#define Hello_Mo_IF_IF.C.H

#endif


//
//Hello_Mo_IF_IF.C.C
//

#include "Hello_Mo.C.H"


//
//Hello_Mo_IF.C.H
//

#ifndef Hello_Mo_IF_H
#define Hello_Mo_IF_H

#include <Xm/MainW.h>
#include <Xm/PushB.h>

class Hello_Mo_IFTOP_Class;
extern Hello_Mo_IFTOP_Class *Hello_Mo_IFTOP;
class Hello_If_Main_Class;
extern Hello_If_Main_Class *Hello_If_Main;
class Hello_If_PB1_Class;
```

```
extern Hello_If_PB1_Class *Hello_If_PB1;
class Hello_Mo_IFTOP_Class : public BaseWidget {
        Arg args[10];
        int n;
        Hello_If_Main_Class *Hello_If_Main_widget;
public:
        Hello_Mo_IFTOP_Class(Widget, char *);
        void createWidget(Widget);
        void popup();
        void popdown();
};

class Hello_If_Main_Class : public BaseWidget {
        Arg args[10];
        int n;
        Hello_If_PB1_Class *Hello_If_PB1_widget;
public:
        Hello_If_Main_Class(Widget, char *);
        void createWidget(Widget);
};

class Hello_If_PB1_Class : public BaseWidget {
        Arg args[10];
        int n;
public:
        Hello_If_PB1_Class(Widget, char *);
        void createWidget(Widget);
};

#endif


//
//Hello_Mo_IF.C.C
//

#include "Hello_Mo.C.H"

Hello_Mo_IFTOP_Class *Hello_Mo_IFTOP = NULL;
Hello_Mo_IFTOP_Class::Hello_Mo_IFTOP_Class(Widget parent, char *name) : Bas
eWidget(name)
{
        Hello_Mo_IFTOP = this;
```

```
        Hello_Mo_IFTOP->createWidget(parent);
        Hello_Mo_IFTOP->popup();
}

void Hello_Mo_IFTOP_Class::createWidget(Widget parent)
{
        n = 0;
        Display *display;
        display = XtDisplay(parent);
        XtSetArg(args[n], XmNdeleteResponse, XmDO_NOTHING); n++;
        _w = XtAppCreateShell(_name, "TopLevelShell",
                topLevelShellWidgetClass, display, args, n);
        Hello_If_Main_widget = new Hello_If_Main_Class(_w, "Hello_If_Main
");
        Hello_If_Main_widget->manage();
}

void Hello_Mo_IFTOP_Class::popup()
{
        XtRealizeWidget(_w);
}

void Hello_Mo_IFTOP_Class::popdown()
{
        XtUnrealizeWidget(_w);
}

Hello_If_Main_Class *Hello_If_Main = NULL;
Hello_If_Main_Class::Hello_If_Main_Class(Widget parent, char *name) : BaseW
idget(name)
{
        Hello_If_Main = this;
        Hello_If_Main->createWidget(parent);
}

void Hello_If_Main_Class::createWidget(Widget parent)
{
        n = 0;
        XtSetArg(args[n], XmNx, 638); n++;
        XtSetArg(args[n], XmNy, 303); n++;
        _w = XmCreateMainWindow(parent, _name, args, n);
        Hello_If_PB1_widget = new Hello_If_PB1_Class(_w, "Hello_If_PB1");
        Hello_If_PB1_widget->manage();
```

```
}
Hello_If_PB1_Class *Hello_If_PB1 = NULL;
Hello_If_PB1_Class::Hello_If_PB1_Class(Widget parent, char *name) : BaseWid
get(name)
{
        Hello_If_PB1 = this;
        Hello_If_PB1->createWidget(parent);
}

void Hello_If_PB1_Class::createWidget(Widget parent)
{
        n = 0;
        XtSetArg(args[n], XmNx, 6); n++;
        XtSetArg(args[n], XmNy, 7); n++;
        _w = XmCreatePushButton(parent, _name, args, n);
}


##############################
##    Hello_Mo.C.mk
##############################

C++ = CC
INC = /usr1/lsm/owc/Gen/include
MO_LIB = /usr1/lsm/owc/Gen/lib
C++FLAGS = -g -I$(INC) -DFUNCPROTO -DXTFUNCPROTO
LDFLAGS =
CFLAGS = -g -I$(INC)

XLIB = -lXm -lXt -lX11
MOLIB = $(MO_LIB)/libGen.a
LIBS = $(XLIB) $(MOLIB)

.SUFFIXES: .C .c .o

.C:
        $(C++) $(C++FLAGS) $(LDFLAGS) -o $* $*.C $(LIBS)

.C.o:
        $(C++) -c $(C++FLAGS) $*.C
OBJS = Hello_Mo.C.o  Hello_Mo_IF.C.o Hello_Mo_IF_IF.C.o

Hello_Mo:       $(OBJS)
```

```
                $(C++) $(LDFLAGS) $(OBJS) $(LIBS) -o $@
Hello_Mo.C.o:   Hello_Mo.C.C
Hello_Mo_IF.C.o:        Hello_Mo_IF.C.C
Hello_Mo_IF_IF.C.o:     Hello_Mo_IF_IF.C.C

clean:
        rm Hello_Mo $(OBJS)
```

나. C 소스코드

```
/*****************************
 Hello_Mo.h
*****************************/

/* main header Hello_Mo.h file is generated */

#include <stdio.h>
#include <sys/signal.h>
#include <limits.h>
#include <assert.h>
#include <ctype.h>
#include <X11/Intrinsic.h>
#include <X11/IntrinsicP.h>
#include <X11/Shell.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/CutPaste.h>

#include "Hello_Mo_IF_IF.h"
#include "Hello_Mo_IF.h"

/* end */
```

```
/*************************************
Hello_Mo.c
*************************************/

#include "Hello_Mo.h"

void main(argc, argv)
```

```
int argc;
char **argv;
{
        Widget toplevel;
        toplevel = XtInitialize(argv[0], "Hello_Mo.rc", NULL, 0, &argc,
argv);
        Hello_Mo_IFTOPCreate(toplevel);
        XtMainLoop();
}


/*******************************
Hello_Mo_IF_IF.h
*******************************/

/* application  Hello_Mo_IF_IF.h file is generated */

/*end*/


/*******************************
Hello_Mo_IF_IF.c
*******************************/

#include "Hello_Mo.h"


/*******************************
Hello_Mo_IF.h
*******************************/

/* Hello_Mo_IF.h file is generated */
/* global values and functions declaration */

#include <Xm/MainW.h>
#include <Xm/PushB.h>

extern Widget Hello_Mo_IFTOP;
extern Widget Hello_If_Main;
extern Widget Hello_If_PB1;
extern void Hello_Mo_IFTOPCreate();
```

```
/*****************************
Hello_Mo_IF.c
*****************************/

#include "Hello_Mo.h"

Widget Hello_Mo_IFTOP;
Widget Hello_If_Main;
Widget Hello_If_PB1;

void Hello_Mo_IFTOPCreate(parent)
Widget parent;
{
        Arg args[10];
        int n = 0;
        Display *display;
        display = XtDisplay(parent);
        XtSetArg(args[n], XmNdeleteResponse, XmDO_NOTHING); n++;
        Hello_Mo_IFTOP = XtAppCreateShell("Hello_Mo_IFTOP", "TopLevelShell
",
                        topLevelShellWidgetClass, display, args, n);
        n = 0;
        XtSetArg(args[n], XmNx, 638); n++;
        XtSetArg(args[n], XmNy, 303); n++;
        Hello_If_Main = XmCreateMainWindow(Hello_Mo_IFTOP, "Hello_If_Main",
args, n);
        XtManageChild(Hello_If_Main);
        n = 0;
        XtSetArg(args[n], XmNx, 6); n++;
        XtSetArg(args[n], XmNy, 7); n++;
        Hello_If_PB1 = XmCreatePushButton(Hello_If_Main, "Hello_If_PB1", ar
gs, n);
        XtManageChild(Hello_If_PB1);
        XtRealizeWidget(Hello_Mo_IFTOP);
}

##############################
##  Hello_Mo.mk
##############################

C = cc
```

```
INC =
LDFLAGS =
CFLAGS = -g -I$(INC) -D_NO_PROTO

XLIB = -lXm -lXt -lX11
LIBS = $(XLIB)

.SUFFIXES: .c .o

.c:
        $(C) $(CFLAGS) $(LDFLAGS) -o $* $*.c $(LIBS)


.c.o:
        $(C) -c $(CFLAGS) $*.c

OBJS = Hello_Mo.o  Hello_Mo_IF.o Hello_Mo_IF_IF.o

Hello_Mo:       $(OBJS)
                $(C) $(LDFLAGS) $(OBJS) $(LIBS) -o $@

Hello_Mo.o:     Hello_Mo.c
Hello_Mo_IF.o:  Hello_Mo_IF.c
Hello_Mo_IF_IF.o:       Hello_Mo_IF_IF.c

clean:
        rm Hello_Mo $(OBJS)
```

다. 리소스 화일

```
###################################
##    Hello_Mo.rc
###################################

*background:      LightSteelBlue
*foreground:      black
*fontList:        clb8x13:tgM16ks:
*Hello_If_Main.width:     295
*Hello_If_Main.height:    152
*Hello_If_PB1.width:      283
*Hello_If_PB1.height:     136
```

```
*Hello_If_PB1.background:        AntiqueWhite2
*Hello_If_PB1.borderColor:       black
*Hello_If_PB1.foreground:        black
*Hello_If_PB1.highlightColor:    black
*Hello_If_PB1.topShadowColor:    #bfbfb3b3a3a3
*Hello_If_PB1.bottomShadowColor:         #83837b7b7070
*Hello_If_PB1.labelString:       Hello, GUI Mosaic
*Hello_If_PB1.armColor: #cbcbbebeaeae
```

2. 텍스트 에디터(Text Editor)

가. C++ 소스코드

```
//
//Text_Editor.C.H
//

#ifndef Text_Editor.C.H
#define Text_Editor.C.H

#include <stdio.h>
#include <sys/signal.h>
#include <limits.h>
#include <assert.h>
#include <ctype.h>
#include <X11/Intrinsic.h>
#include <X11/IntrinsicP.h>
#include <X11/Shell.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/CutPaste.h>
#include "Mo_BaseWidget.h"
#include "Text_Editor_If_IF.C.H"
#include "Text_Editor_If.C.H"

#endif
```

```
//
//Text_Editor.C.C
//


#include "Text_Editor.C.H"
void main(int argc, char **argv)
{
        Widget toplevel;
        toplevel = XtInitialize(argv[0], "Text_Editor.rc", NULL, 0, &argc,
argv);

        Text_Editor_IfTOP_Class *local_Text_Editor_IfTOP = new Text_Editor_
IfTOP_Class(toplevel, "Text_Editor_IfTOP");

        XtMainLoop();
}


//
//Text_Editor_If_IF.C.H
//


#ifndef Text_Editor_If_IF.C.H
#define Text_Editor_If_IF.C.H

#endif


//
//Text_Editor_If_IF.C.C
//


#include "Text_Editor.C.H"

//
//Text_Editor_If.C.H
//
```

```
#ifndef Text_Editor_If_H
#define Text_Editor_If_H

#include <Xm/MainW.h>
#include <Xm/RowColumn.h>
#include <Xm/CascadeB.h>
#include <Xm/PushB.h>
#include <Xm/Separator.h>
#include <Xm/Text.h>
class Text_Editor_IfTOP_Class;
extern Text_Editor_IfTOP_Class *Text_Editor_IfTOP;
class Text_Editor_Main_Class;
extern Text_Editor_Main_Class *Text_Editor_Main;
class Text_Editor_MB_Class;
extern Text_Editor_MB_Class *Text_Editor_MB;
class Editor_CB_Class;
extern Editor_CB_Class *Editor_CB;
class MB_FilePD_Class;
extern MB_FilePD_Class *MB_FilePD;
class File_OpenPB_Class;
extern File_OpenPB_Class *File_OpenPB;
class File_LoadPB_Class;
extern File_LoadPB_Class *File_LoadPB;
class File_SavePB_Class;
extern File_SavePB_Class *File_SavePB;
class File_SaveSP_Class;
extern File_SaveSP_Class *File_SaveSP;
class File_QuitPB_Class;
extern File_QuitPB_Class *File_QuitPB;
class Editor_ST_Class;
extern Editor_ST_Class *Editor_ST;

class Text_Editor_IfTOP_Class : public BaseWidget {
        Arg args[10];
        int n;
        Text_Editor_Main_Class *Text_Editor_Main_widget;

public:
        Text_Editor_IfTOP_Class(Widget, char *);
        void createWidget(Widget);
        void popup();
        void popdown();
};
```

```
class Text_Editor_Main_Class : public BaseWidget {
        Arg args[10];
        int n;
        Text_Editor_MB_Class *Text_Editor_MB_widget;
        Editor_ST_Class *Editor_ST_widget;

public:
        Text_Editor_Main_Class(Widget, char *);
        void createWidget(Widget);
};

class Text_Editor_MB_Class : public BaseWidget {
        Arg args[10];
        int n;
        Editor_CB_Class *Editor_CB_widget;
        MB_FilePD_Class *MB_FilePD_widget;
public:
        Text_Editor_MB_Class(Widget, char *);
        void createWidget(Widget);
        void Editor_CB_subMenu(Widget);
};

class Editor_CB_Class : public BaseWidget {
        Arg args[10];
        int n;
public:
        Editor_CB_Class(Widget, char *);
        void createWidget(Widget);
        void subMenu(Widget);
};

class MB_FilePD_Class : public BaseWidget {
        Arg args[10];
        int n;
        File_OpenPB_Class *File_OpenPB_widget;
        File_LoadPB_Class *File_LoadPB_widget;
        File_SavePB_Class *File_SavePB_widget;
        File_SaveSP_Class *File_SaveSP_widget;
        File_QuitPB_Class *File_QuitPB_widget;
public:
        MB_FilePD_Class(Widget, char *);
        void createWidget(Widget);
};
```

```
class File_OpenPB_Class : public BaseWidget {
        Arg args[10];
        int n;
public:
        File_OpenPB_Class(Widget, char *);
        void createWidget(Widget);
};

class File_LoadPB_Class : public BaseWidget {
        Arg args[10];
        int n;
public:
        File_LoadPB_Class(Widget, char *);
        void createWidget(Widget);
};

class File_SavePB_Class : public BaseWidget {
        Arg args[10];
        int n;
public:
        File_SavePB_Class(Widget, char *);
        void createWidget(Widget);
};

class File_SaveSP_Class : public BaseWidget {
        Arg args[10];
        int n;

public:
        File_SaveSP_Class(Widget, char *);
        void createWidget(Widget);
};

class File_QuitPB_Class : public BaseWidget {
        Arg args[10];
        int n;
public:
        File_QuitPB_Class(Widget, char *);
        void createWidget(Widget);
        void callback(XtPointer);
};

class Editor_ST_Class : public BaseWidget {
```

```
        Arg args[10];
        int n;
public:
        Editor_ST_Class(Widget, char *);
        void createWidget(Widget);
};

#endif


//
//Text_Editor_If.C.C
//


#include "Text_Editor.C.H"

Text_Editor_IfTOP_Class *Text_Editor_IfTOP = NULL;
Text_Editor_IfTOP_Class::Text_Editor_IfTOP_Class(Widget parent, char *name)
: BaseWidget(name)
{
        Text_Editor_IfTOP = this;
        Text_Editor_IfTOP->createWidget(parent);
        Text_Editor_IfTOP->popup();
}

void Text_Editor_IfTOP_Class::createWidget(Widget parent)
{
        n = 0;
        Display *display;
        display = XtDisplay(parent);
        XtSetArg(args[n], XmNdeleteResponse, XmDO_NOTHING); n++;
        _w = XtAppCreateShell(_name, "TopLevelShell",
                topLevelShellWidgetClass, display, args, n);
        Text_Editor_Main_widget = new Text_Editor_Main_Class(_w, "Text_Edit
or_Main");
        Text_Editor_Main_widget->manage();
}

void Text_Editor_IfTOP_Class::popup()
{
        XtRealizeWidget(_w);
}
```

```
void Text_Editor_IfTOP_Class::popdown()
{
        XtUnrealizeWidget(_w);
}

Text_Editor_Main_Class *Text_Editor_Main = NULL;
Text_Editor_Main_Class::Text_Editor_Main_Class(Widget parent, char *name) :
BaseWidget(name)
{
        Text_Editor_Main = this;
        Text_Editor_Main->createWidget(parent);
}

void Text_Editor_Main_Class::createWidget(Widget parent)
{
        n = 0;
        XtSetArg(args[n], XmNx, 624); n++;
        XtSetArg(args[n], XmNy, 288); n++;
        _w = XmCreateMainWindow(parent, _name, args, n);
        Text_Editor_MB_widget = new Text_Editor_MB_Class(_w, "Text_Editor_M
B");
        Text_Editor_MB_widget->manage();
        Editor_ST_widget = new Editor_ST_Class(_w, "Editor_ST");
        Editor_ST_widget->manage();
        Text_Editor_MB_widget->Editor_CB_subMenu(MB_FilePD->baseWidget());
}

Text_Editor_MB_Class *Text_Editor_MB = NULL;
Text_Editor_MB_Class::Text_Editor_MB_Class(Widget parent, char *name) : Bas
eWidget(name)
{
        Text_Editor_MB = this;
        Text_Editor_MB->createWidget(parent);
}

void Text_Editor_MB_Class::createWidget(Widget parent)
{
        _w = XmCreateMenuBar(parent, _name, NULL, 0);
        Editor_CB_widget = new Editor_CB_Class(_w, "Editor_CB");
        Editor_CB_widget->manage();
        MB_FilePD_widget = new MB_FilePD_Class(_w, "MB_FilePD");
}
```

```
void Text_Editor_MB_Class::Editor_CB_subMenu(Widget sub)
{
        Editor_CB->subMenu(sub);
}

Editor_CB_Class *Editor_CB = NULL;
Editor_CB_Class::Editor_CB_Class(Widget parent, char *name) : BaseWidget(na
me)
{
        Editor_CB = this;
        Editor_CB->createWidget(parent);
}

void Editor_CB_Class::createWidget(Widget parent)
{
        n = 0;
        XtSetArg(args[n], XmNx, 19); n++;
        XtSetArg(args[n], XmNy, 1); n++;
        _w = XmCreateCascadeButton(parent, _name, args, n);
}

void Editor_CB_Class::subMenu(Widget sub)
{
        Arg myargs[10];
        int myn = 0;
        XtSetArg(myargs[myn], XmNsubMenuId, sub); myn++;
        XtSetValues(_w, myargs, myn);
}

MB_FilePD_Class *MB_FilePD = NULL;
MB_FilePD_Class::MB_FilePD_Class(Widget parent, char *name) : BaseWidget(na
me)
{
        MB_FilePD = this;
        MB_FilePD->createWidget(parent);
}

void MB_FilePD_Class::createWidget(Widget parent)
{
        _w = XmCreatePulldownMenu(parent, _name, NULL, 0);
        File_OpenPB_widget = new File_OpenPB_Class(_w, "File_OpenPB");
        File_OpenPB_widget->manage();
        File_LoadPB_widget = new File_LoadPB_Class(_w, "File_LoadPB");
```

```
File_LoadPB_widget->manage();
File_SavePB_widget = new File_SavePB_Class(_w, "File_SavePB");
File_SavePB_widget->manage();
File_SaveSP_widget = new File_SaveSP_Class(_w, "File_SaveSP");
File_SaveSP_widget->manage();
File_QuitPB_widget = new File_QuitPB_Class(_w, "File_QuitPB");
File_QuitPB_widget->manage();
        File_QuitPB_widget->create_callback();
}

File_OpenPB_Class *File_OpenPB = NULL;
File_OpenPB_Class::File_OpenPB_Class(Widget parent, char *name) : BaseWidge
t(name)
{
        File_OpenPB = this;
        File_OpenPB->createWidget(parent);
}

void File_OpenPB_Class::createWidget(Widget parent)
{
        _w = XmCreatePushButton(parent, _name, NULL, 0);
}

File_LoadPB_Class *File_LoadPB = NULL;
File_LoadPB_Class::File_LoadPB_Class(Widget parent, char *name) : BaseWidge
t(name)
{
        File_LoadPB = this;
        File_LoadPB->createWidget(parent);
}

void File_LoadPB_Class::createWidget(Widget parent)
{
        _w = XmCreatePushButton(parent, _name, NULL, 0);
}

File_SavePB_Class *File_SavePB = NULL;
File_SavePB_Class::File_SavePB_Class(Widget parent, char *name) : BaseWidge
t(name)
{
        File_SavePB = this;
        File_SavePB->createWidget(parent);
}
```

```
void File_SavePB_Class::createWidget(Widget parent)
{
        _w = XmCreatePushButton(parent, _name, NULL, 0);
}
File_SaveSP_Class *File_SaveSP = NULL;
File_SaveSP_Class::File_SaveSP_Class(Widget parent, char *name) : BaseWidge
t(name)
{
        File_SaveSP = this;
        File_SaveSP->createWidget(parent);
}

void File_SaveSP_Class::createWidget(Widget parent)
{
        _w = XmCreateSeparator(parent, _name, NULL, 0);
}

File_QuitPB_Class *File_QuitPB = NULL;
File_QuitPB_Class::File_QuitPB_Class(Widget parent, char *name) : BaseWidge
t(name)
{
        File_QuitPB = this;
        File_QuitPB->createWidget(parent);
}

void File_QuitPB_Class::createWidget(Widget parent)
{
        _w = XmCreatePushButton(parent, _name, NULL, 0);
}

void File_QuitPB_Class::callback(XtPointer calldata)
{
        exit(0);
}


Editor_ST_Class *Editor_ST = NULL;
Editor_ST_Class::Editor_ST_Class(Widget parent, char *name) : BaseWidget(na
me)
{
        Editor_ST = this;
        Editor_ST->createWidget(parent);
}
```

```
void Editor_ST_Class::createWidget(Widget parent)
{
        n = 0;
        XtSetArg(args[n], XmNx, 6); n++;

        XtSetArg(args[n], XmNy, 36); n++;
        _w = XmCreateScrolledText(parent, _name, args, n);
}
```

```
##################################
##    Text_Editor.C.mk
##################################

C++ = CC
INC = /usr1/lsm/owc/Gen/include
MO_LIB = /usr1/lsm/owc/Gen/lib
C++FLAGS = -g -I$(INC) -DFUNCPROTO -DXTFUNCPROTO
LDFLAGS =
CFLAGS = -g -I$(INC)

XLIB = -lXm -lXt -lX11
MOLIB = $(MO_LIB)/libGen.a
LIBS = $(XLIB) $(MOLIB)

.SUFFIXES: .C .c .o

.C:
        $(C++) $(C++FLAGS) $(LDFLAGS) -o $* $*.C $(LIBS)

.C.o:
        $(C++) -c $(C++FLAGS) $*.C

OBJS = Text_Editor.C.o  Text_Editor_If.C.o Text_Editor_If_IF.C.o

Text_Editor:    $(OBJS)
        $(C++) $(LDFLAGS) $(OBJS) $(LIBS) -o $@

Text_Editor.C.o:        Text_Editor.C.C
Text_Editor_If.C.o:     Text_Editor_If.C.C
Text_Editor_If_IF.C.o:  Text_Editor_If_IF.C.C

clean:
```

```
	rm Text_Editor $(OBJS)
```

나. C 소스코드

```
/*****************************
Text_Editor.h
*****************************/

/* main header Text_Editor.h file is generated */

#include <stdio.h>
#include <sys/signal.h>
#include <limits.h>
#include <assert.h>
#include <ctype.h>
#include <X11/Intrinsic.h>
#include <X11/IntrinsicP.h>
#include <X11/Shell.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/CutPaste.h>

#include "Text_Editor_If_IF.h"
#include "Text_Editor_If.h"

/* end */


/*****************************
Text_Editor.c
*****************************/

#include "Text_Editor.h"


void main(argc, argv)
int argc;
char **argv;
{
```

```
        Widget toplevel;
        toplevel = XtInitialize(argv[0], "Text_Editor.rc", NULL, 0, &argc,
argv);
        Text_Editor_IfTOPCreate(toplevel);

        XtMainLoop();
}

/*******************************
Text_Editor_If_IF.h
*******************************/

/* application  Text_Editor_If_IF.h file is generated */

/*end*/

/*******************************
Text_Editor_If_IF.c
*******************************/

#include "Text_Editor.h"

/*******************************
Text_Editor_If.h
*******************************/

/* Text_Editor_If.h file is generated */
/* global values and functions declaration */

#include <Xm/MainW.h>
#include <Xm/RowColumn.h>
#include <Xm/CascadeB.h>
#include <Xm/PushB.h>
#include <Xm/Separator.h>
#include <Xm/Text.h>

extern Widget Text_Editor_IfTOP;
```

```
extern Widget Text_Editor_Main;
extern Widget Text_Editor_MB;
extern Widget Editor_CB;
extern Widget MB_FilePD;
extern Widget File_OpenPB;
extern Widget File_LoadPB;
extern Widget File_SavePB;
extern Widget File_SaveSP;
extern Widget File_QuitPB;
extern Widget Editor_ST;

extern void Text_Editor_IfTOPCreate();
extern void File_QuitPB_activateCallback();


/*****************************
Text_Editor_If.c
*****************************/


#include "Text_Editor.h"

Widget Text_Editor_IfTOP;
Widget Text_Editor_Main;
Widget Text_Editor_MB;
Widget Editor_CB;
Widget MB_FilePD;
Widget File_OpenPB;
Widget File_LoadPB;
Widget File_SavePB;
Widget File_SaveSP;
Widget File_QuitPB;
Widget Editor_ST;

void Text_Editor_IfTOPCreate(parent)
Widget parent;
{
        Arg args[10];
        int n = 0;
        Display *display;
        display = XtDisplay(parent);
        XtSetArg(args[n], XmNdeleteResponse, XmDO_NOTHING); n++;
        Text_Editor_IfTOP = XtAppCreateShell("Text_Editor_IfTOP", "TopLevel
```

```
Shell",
                topLevelShellWidgetClass, display, args, n);
        n = 0;
        XtSetArg(args[n], XmNx, 624); n++;
        XtSetArg(args[n], XmNy, 288); n++;
        Text_Editor_Main = XmCreateMainWindow(Text_Editor_IfTOP, "Text_Edit
or_Main", args, n);
        XtManageChild(Text_Editor_Main);
        Text_Editor_MB = XmCreateMenuBar(Text_Editor_Main, "Text_Editor_MB
", NULL, 0);
        XtManageChild(Text_Editor_MB);
        n = 0;
        XtSetArg(args[n], XmNx, 19); n++;
        XtSetArg(args[n], XmNy, 1); n++;
        Editor_CB = XmCreateCascadeButton(Text_Editor_MB, "Editor_CB", arg
s, n);
        XtManageChild(Editor_CB);
        MB_FilePD = XmCreatePulldownMenu(Text_Editor_MB, "MB_FilePD", NULL,
0);
        File_OpenPB = XmCreatePushButton(MB_FilePD, "File_OpenPB", NULL,
0);
        XtManageChild(File_OpenPB);
        File_LoadPB = XmCreatePushButton(MB_FilePD, "File_LoadPB", NULL,
0);
        XtManageChild(File_LoadPB);
        File_SavePB = XmCreatePushButton(MB_FilePD, "File_SavePB", NULL,
0);
        XtManageChild(File_SavePB);
        File_SaveSP = XmCreateSeparator(MB_FilePD, "File_SaveSP", NULL, 0);
        XtManageChild(File_SaveSP);
        File_QuitPB = XmCreatePushButton(MB_FilePD, "File_QuitPB", NULL,
0);
        XtManageChild(File_QuitPB);
        XtAddCallback(File_QuitPB, XmNactivateCallback, File_QuitPB_activat
eCallback, NULL);
        n = 0;
        XtSetArg(args[n], XmNx, 6); n++;
        XtSetArg(args[n], XmNy, 36); n++;
        Editor_ST = XmCreateScrolledText(Text_Editor_Main, "Editor_ST", arg
s, n);
        XtManageChild(Editor_ST);
        XtVaSetValues(Editor_CB, XmNsubMenuId, MB_FilePD, NULL);
```

```
        XtRealizeWidget(Text_Editor_IfTOP);
}

void File_QuitPB_activateCallback(w, client_data, call_data)
Widget w;
caddr_t client_data;
caddr_t call_data;
{
        exit(0);
}


################################
##  Text_Editor.mk
################################


C = cc
INC =
LDFLAGS =
CFLAGS = -g -I$(INC) -D_NO_PROTO

XLIB = -lXm -lXt -lX11
LIBS = $(XLIB)

.SUFFIXES: .c .o

.c:
        $(C) $(CFLAGS) $(LDFLAGS) -o $* $*.c $(LIBS)


.c.o:
        $(C) -c $(CFLAGS) $*.c

OBJS = Text_Editor.o  Text_Editor_If.o Text_Editor_If_IF.o

Text_Editor:    $(OBJS)
                $(C) $(LDFLAGS) $(OBJS) $(LIBS) -o $@

Text_Editor.o:  Text_Editor.c
Text_Editor_If.o:       Text_Editor_If.c
Text_Editor_If_IF.o:    Text_Editor_If_IF.c
```

```
clean:
        rm Text_Editor $(OBJS)
```

다. 리소스 화일

```
###################################
##    Text_Editor.rc
###################################


*background:     LightSteelBlue
*foreground:     black
*fontList:       clb8x13:tgM16ks:
*Text_Editor_Main.width:          605
*Text_Editor_Main.height:         449
*Editor_CB.width:          20
*Editor_CB.height:         20
*Editor_CB.background:     AntiqueWhite2
*Editor_CB.borderColor:    black
*Editor_CB.foreground:     black
*Editor_CB.highlightColor:         black
*Editor_CB.topShadowColor:         #bfbfb3b3a3a3
*Editor_CB.bottomShadowColor:      #83837b7b7070
*Editor_CB.labelString:  화일
*File_OpenPB.background:           AntiqueWhite2
*File_OpenPB.borderColor:          black
*File_OpenPB.foreground:           black
*File_OpenPB.highlightColor:       black
*File_OpenPB.topShadowColor:       #bfbfb3b3a3a3
*File_OpenPB.bottomShadowColor:    #83837b7b7070
*File_OpenPB.labelString:          열기
*File_OpenPB.mnemonicCharSet:      FONTLIST_DEFAULT_TAG_STRING
*File_OpenPB.armColor:    #cbcbbebeaeae
*File_LoadPB.background:           AntiqueWhite2
*File_LoadPB.borderColor:          black
*File_LoadPB.foreground:           black
*File_LoadPB.highlightColor:       black
*File_LoadPB.topShadowColor:       #bfbfb3b3a3a3
*File_LoadPB.bottomShadowColor:    #83837b7b7070
*File_LoadPB.labelString:          불러오기
*File_LoadPB.mnemonicCharSet:      FONTLIST_DEFAULT_TAG_STRING
```

```
*File_LoadPB.armColor:   #cbcbbebeaeae
*File_SavePB.background:          AntiqueWhite2
*File_SavePB.borderColor:         black
*File_SavePB.foreground:          black
*File_SavePB.highlightColor:      black
*File_SavePB.topShadowColor:      #bfbfb3b3a3a3
*File_SavePB.bottomShadowColor:  #83837b7b7070
*File_SavePB.labelString:         저장하기
*File_SavePB.mnemonicCharSet:     FONTLIST_DEFAULT_TAG_STRING
*File_SavePB.armColor:   #cbcbbebeaeae
*File_QuitPB.background:          AntiqueWhite2
*File_QuitPB.borderColor:         black
*File_QuitPB.foreground:          black
*File_QuitPB.highlightColor:      black
*File_QuitPB.topShadowColor:      #bfbfb3b3a3a3
*File_QuitPB.bottomShadowColor:  #83837b7b7070
*File_QuitPB.labelString:         끝내기
*File_QuitPB.mnemonicCharSet:     Callback
*File_QuitPB.armColor:   #cbcbbebeaeae
*Editor_ST.width:         593
*Editor_ST.height:        404
```

여 백

부록 B: GUI Mosaic 사용 안내서

여 백

부록 B: GUI Mosaic 사용 안내서

1. 모자이크(Mosaic)의 개요

가. 모자이크(Mosaic)란?

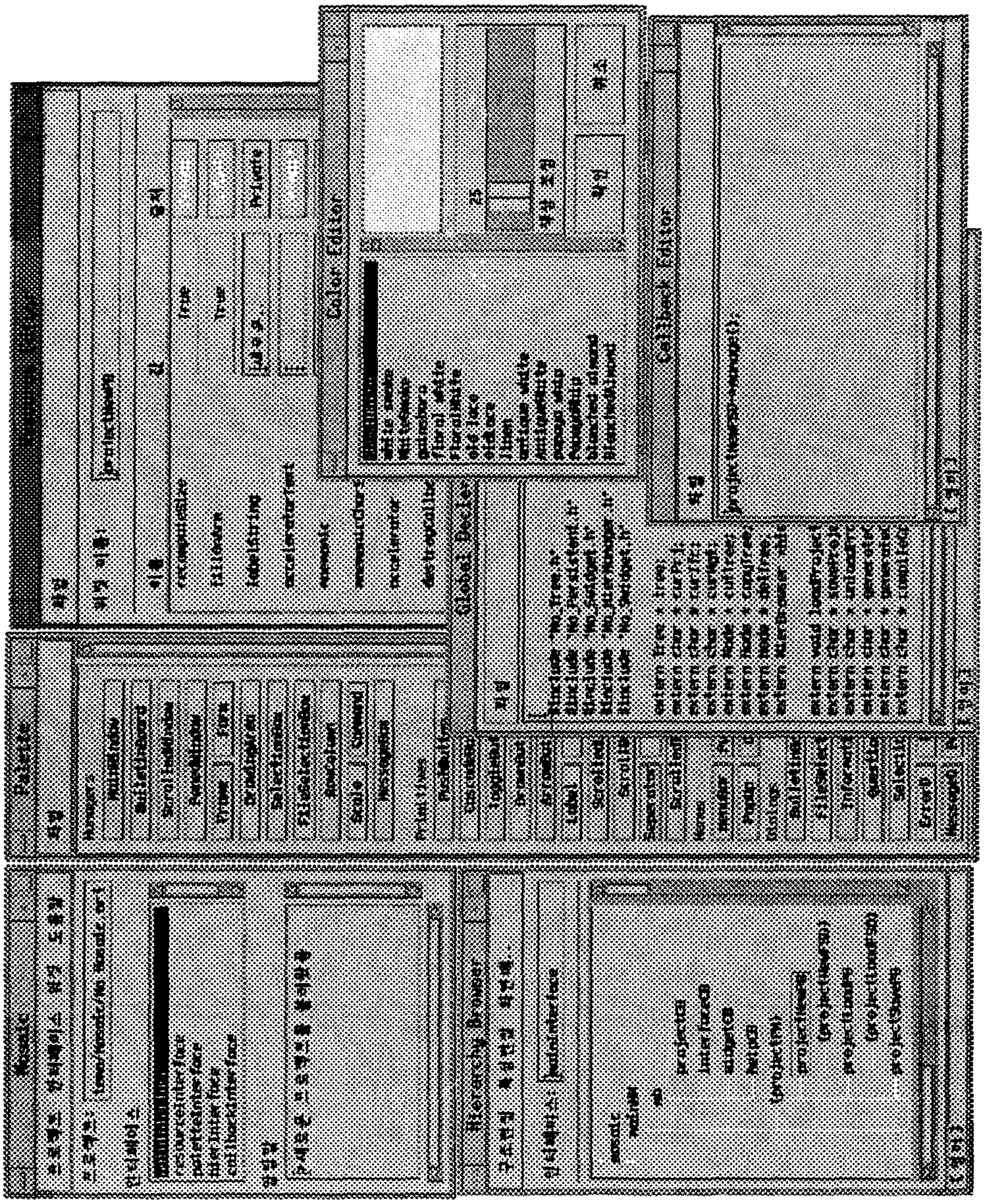
모자이크는 Motif 표준을 따르는 사용자 계면 개발을 지원하는 GUI 도구로써 UNIX, X Window, Motif 환경에서 운영되며, 최종적으로는 Motif 라이브러리를 이용하는 C, C++ 원시코드 및 Makefile을 만들어 준다. 모자이크는 대부분의 작업을 시각적으로 진행할 수 있도록 구성하였으며, 객체지향 개념을 시스템 전반에 걸쳐 도입하여 사용자의 작업, 최종 생성코드 및 작업결과의 관리등 모든 작업이 사용자 입장의 객체를 중심으로하여 이루어 진다. 모자이크의 사용자는 응용시스템의 GUI를 분석하는 분석가, 효과적인 배열 및 모양을 조정 설계하는 설계자 또는 응용시스템을 구현하는 프로그래머등 그 누구도 가능하다. 즉, 모자이크 시스템은 프로토타이핑을 지원하며, 설계 및 코딩을 지원해주는 GUI 도구이다.

현재 모자이크 시스템은 SunOS Release 4.1.3-KLE1.1.3(Solaris 1.1), X Window Release 5(X11R5)/ Motif 1.2 에서 구동된다.

나. 모자이크의 구성

모자이크의 구성은 모자이크 시스템의 전반적인 기동 및 소스코드 생성, 컴파일등을 수행하는 메인 메뉴, 레이아웃상의 각 위짓의 관계를 트리형식으로 표시하며 계층구조를 설계할 수 있는 구조 편집기, 위짓의 속성을 정의 및 편집하는 전용 에디터인 속성 편집기, 속성 편집기상의 색상에 대한 편집을 해주는 색상 편집기, 각 인터페이스의 콜백 및 클래스등에 대한 편집 작업을 수행할 수 있도록 도와주는 각각의 편집기, OSF/Motif에서 제공하는 다양한 위짓을 표시하며

필요한 위짓을 마우스로 선택하여 사용자 인터페이스의 레이아웃을 작성할 수 있도록 도와주는 선택판(palette)등으로 구성된다. [그림 B-1-1]은 모자이크 시스템의 전반적인 구성을 나타낸 화면이다.

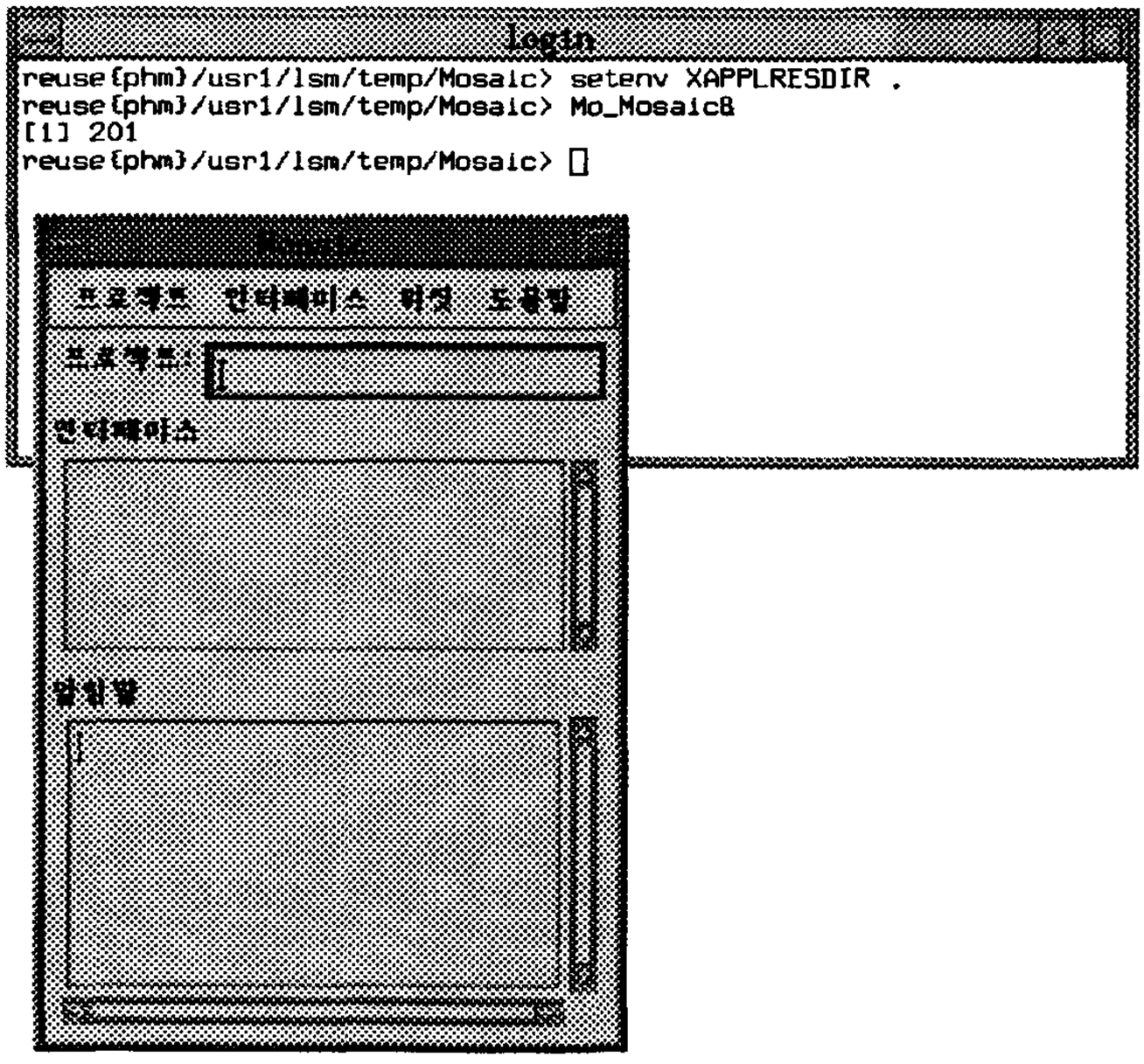


[그림 B-1-1] 모자이크 시스템 구성

2. 모자이크 시작 화면

모자이크 시스템을 구동하기 위해서는 실행 파일인 Mo_Mosaic 화일과 리소스 화일 Mo_Mosaic.rc 화일, C++ 소스코드 생성을 위한 Mo_Gen 화일과 C 소스코드 생성을 위한 Mo_CGen 화일 및 C++ 코드를 생성하기 위해 모자이크에서 제공하는 라이브러리인 libGen.a가 있어야 한다.

프로그램의 수행은 리소스 화일의 환경을 지정하거나 X_Window의 app-default 디렉토리에 모자이크의 리소스 화일을 갖다 놓고 Mosaic을 구동시키면 모자이크의 메인 메뉴가 화면상에 나타나게 된다. [그림 B-2-1]은 모자이크 시스템의 시작 화면이다.



[그림 B-2-1] 모자이크 시스템 시작 화면

가. 프로젝트 창

프로젝트 메뉴의 '불러오기' 메뉴에 의해 불리워지거나, 프로젝트 메뉴의 '새로운..' 메뉴에의해 생성된 프로젝트의 경로명(path)을 나타내주는 창이다.

나. 인터페이스 리스트 창

프로젝트 메뉴의 '불러오기' 메뉴에 의해 불리워진 프로젝트와 관련된 인터페이스의 리스트를 나타내는 창이다.

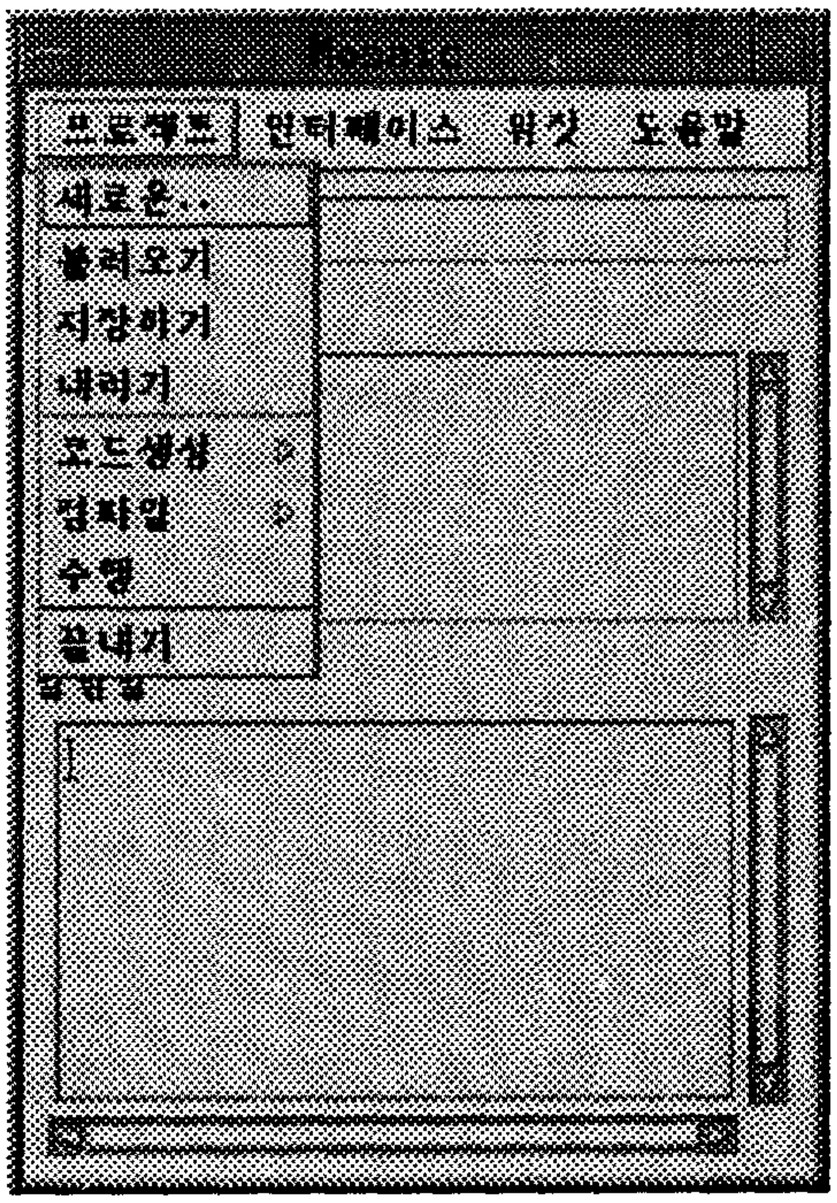
다 알림말 창

모자이크 시스템에서 제공하는 알림말(message)을 나타내주는 창이다.

3. 주화면(Main menu)

가. 프로젝트 메뉴

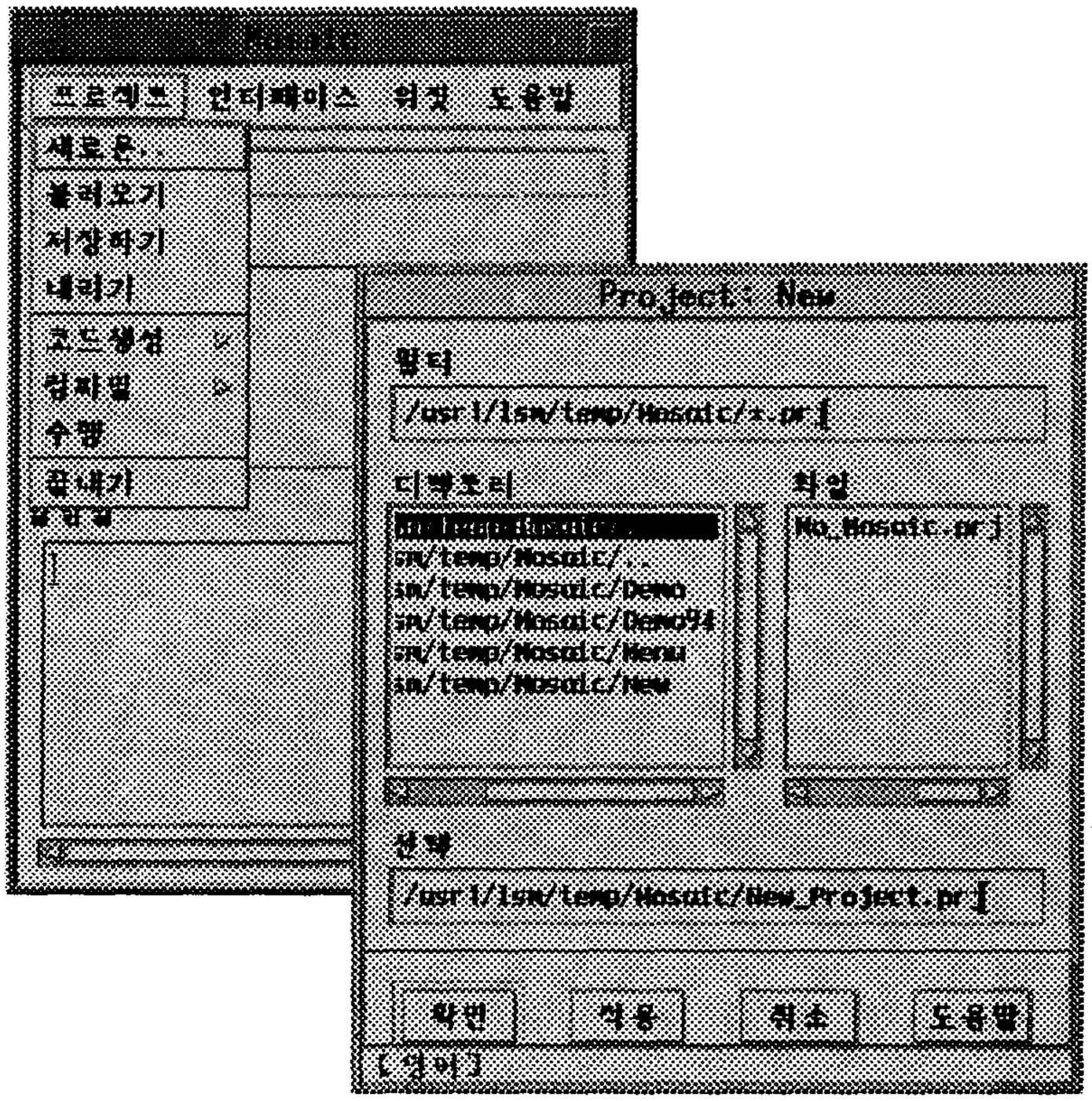
사용자가 작성하려는 프로젝트와 관련된 전반적인 작업을 수행, 관리하는 메뉴로써 모자이크 시스템내에서 인터페이스의 작성에서부터 소스코드의 생성 및 컴파일등과 같은 작업을 가능하게 해준다. 프로젝트 메뉴에는 사용자가 새로운 프로젝트를 시작할때 이 프로젝트를 정의해주는 '새로운..' 메뉴, 기존에 작성된 프로젝트중 원하는 프로젝트를 선택하여 불러오는(load) '불러오기' 메뉴, 새롭게 작성되거나 수정된 프로젝트를 저장하는 '저장하기' 메뉴, 불러온 프로젝트를 작업영역에서 내리는(unload) '내리기' 메뉴, 작성된 인터페이스를 C 혹은 C++ 소스코드 및 make 화일을 생성해주는 '코드생성' 메뉴, 생성된 코드를 모자이크 시스템상에서 컴파일 할 수 있게 해주는 '컴파일' 메뉴, 컴파일된 코드를 모자이크 시스템 상에서 즉시 수행해 볼수 있도록 해주는 '수행' 메뉴 및 모자이크 시스템을 작업영역에서 끝내는 '끝내기' 메뉴가 있다. [그림 B-3-1]은 프로젝트 메뉴 화면이다.



[그림 B-3-1] 프로젝트 메뉴

1)새로운..

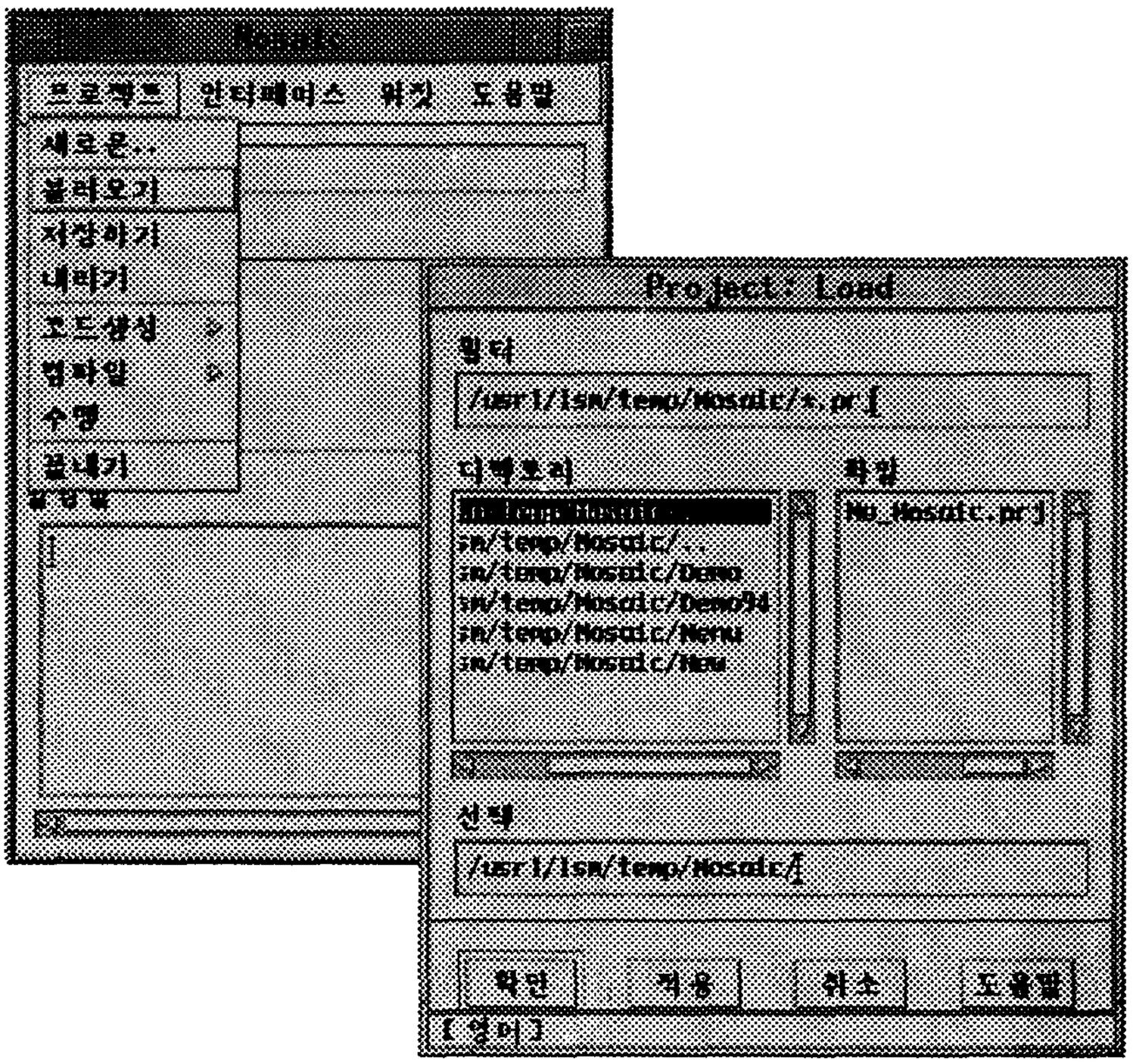
사용자가 새로운 프로젝트(사용자 인터페이스)를 작성하려고 할때 프로젝트 이름을 입력하기 위한 메뉴로써 확장자(extension)는 prj(Project_name.prj)로 사용해야 하며 선택 필드에 새로운 프로젝트 이름을 입력한 후 '확인' 버튼을 선택하면 메인 메뉴의 프로젝트 필드에 새로 입력한 프로젝트 이름이 나타나며 다이아로그가 닫히게 되면 '취소' 버튼을 선택하면 새로운 프로젝트 입력 작업은 취소되며 다이아로그를 닫게 된다. 이때 새로 작성한 프로젝트는 프로젝트 메뉴의 '저장하기' 메뉴를 선택해야만 시스템에 저장이 된다.프로젝트 이름은 한글도 가능하며 토글키는 Shift+Space 이다. [그림 B-3-2]는 새로운 프로젝트를 입력하기 위한 화면이다.



[그림 B-3-2] 프로젝트 '새로운..' 메뉴

2)불러오기

사용자가 기존에 작성한 프로젝트를 불러오기 위한 메뉴로써 마우스로 원하는 프로젝트를 선택하고 '확인' 버튼을 선택하면 원하는 프로젝트가 불리워진다. 이때 선택된 프로젝트 이름이 프로젝트 필드에 나타나며 해당 프로젝트에 관련된 인터페이스의 리스트들이 인터페이스 필드에 나타나게 된다. [그림 B-3-3]은 기존에 작성된 프로젝트중 필요한 프로젝트를 선택하여 불러오기 위한 화면이다.



[그림 B-3-3] 프로젝트 '불러오기' 메뉴

3) 저장하기

새롭게 작성된 프로젝트의 인터페이스 혹은 변경한 인터페이스를 저장하는 메뉴로써 현재의 디렉토리에 저장이되며 저장되어지는 화일에는 프로젝트에 관련된 화일에 대한 경로명(path) 및 인터페이스 화일의 이름을 저장하는 .prj 화일, 인터페이스에 관한 일반적인 정보와 어플리케이션과 관련된 클래스, 함수 및 전역 선언등을 저장하는 .if 화일 및 각 인터페이스의 위짓에 대한 자원(resource)과 위짓간의 계층관계를 저장하는 .bd 화일이 있다.

4) 내리기

불러온 프로젝트에 대한 인터페이스 혹은 작업중이던 인터페이스를 작업영역에서 내리는(unload) 메뉴로써 프로젝트 및 프로젝트와 관련된 모든 인터페이스를 작업영역으로부터 내린다.

5) 코드생성

작성된 인터페이스로부터 C 혹은 C++ 소스코드 및 자원화일(resource file)과 Make 화일을 생성해주는 메뉴로써 생성된 소스코드는 컴파일 작업후 즉시 실행할 수 있는 완전한 소스코드이다.

가) C++

작성된 인터페이스로부터 C++ 소스코드 및 자원 화일과 Make 화일을 생성해주는 메뉴로써 생성되어지는 화일에는 소스코드 화일인 ProjectName.C.C 화일, InterfaceName.C.C 화일, 응용코드 화일인 InterfaceName_IF.C.C 화일 헤더 화일인 ProjectName.C.H, InterfaceName.C.H, InterfaceName_IF.C.H 화일과 자원화일인 ProjectName.rc 화일 및 Make 화일인 ProjectName.C.mk 화일이 생성된다.

나) C

C++ 메뉴와 동일한 기능을 가지는 메뉴로써 생성되는 파일에는 소스코드 화일인 ProjecName.c 화일, InterfaceName.c 화일, 응용코드 화일인 InterfaceName_IF.c 화일, 헤더화일인 ProjectName.h, InterfaceName.h, InterfaceName_IF.h 화일과 자원화일인 ProjectName.rc 화일 및 Make 화일 ProjectName.c.mk 화일이 생성된다.

6) 컴파일

코드생성 메뉴로부터 생성된 소스코드 및 헤더파일을 이용하여 컴파일 작업을 수행해주는 메뉴이며 코멘드 라인(command line)에서 make를 수행하여도 되는데 이때의 명령어는 make -f Project_Name.C.mk 이다.

가) C++

코드생성 메뉴의 C++ 메뉴에의해 생성된 C++ 소스코드(.C 화일) 및 헤더파일(.H 화일)을 이용하여 컴파일을 수행하는 메뉴로써 이때 이용되는 Make 화일은 ProjectName.C.mk 화일이며 컴파일된 코드는 즉시 수행할 수 있는 실행코드(execution code) 이며 실행화일명은 프로젝트명이다.

나) C

코드생성 메뉴의 C 메뉴에 의해 생성된 C 소스코드(.c 화일) 및 헤더화일(.h 화일)을 이용하여 컴파일을 수행하는 메뉴로써 이때 사용되는 Make 화일은 ProjectName.mk 화일이며 C++ 메뉴에서와 마찬가지로 컴파일된 코드는 즉시 수행할 수 있는 실행코드이다.

7) 수행

컴파일 메뉴에의해 컴파일된 실행코드를 즉시 수행하는 메뉴로써 사용자가 작성한 인터페이스의 실행모듈이 화면상에 나타나며 작성된 인터페이스가 사용자가 원하는 형태로 수행되는지 확인할 수 있게 하는 메뉴이다.

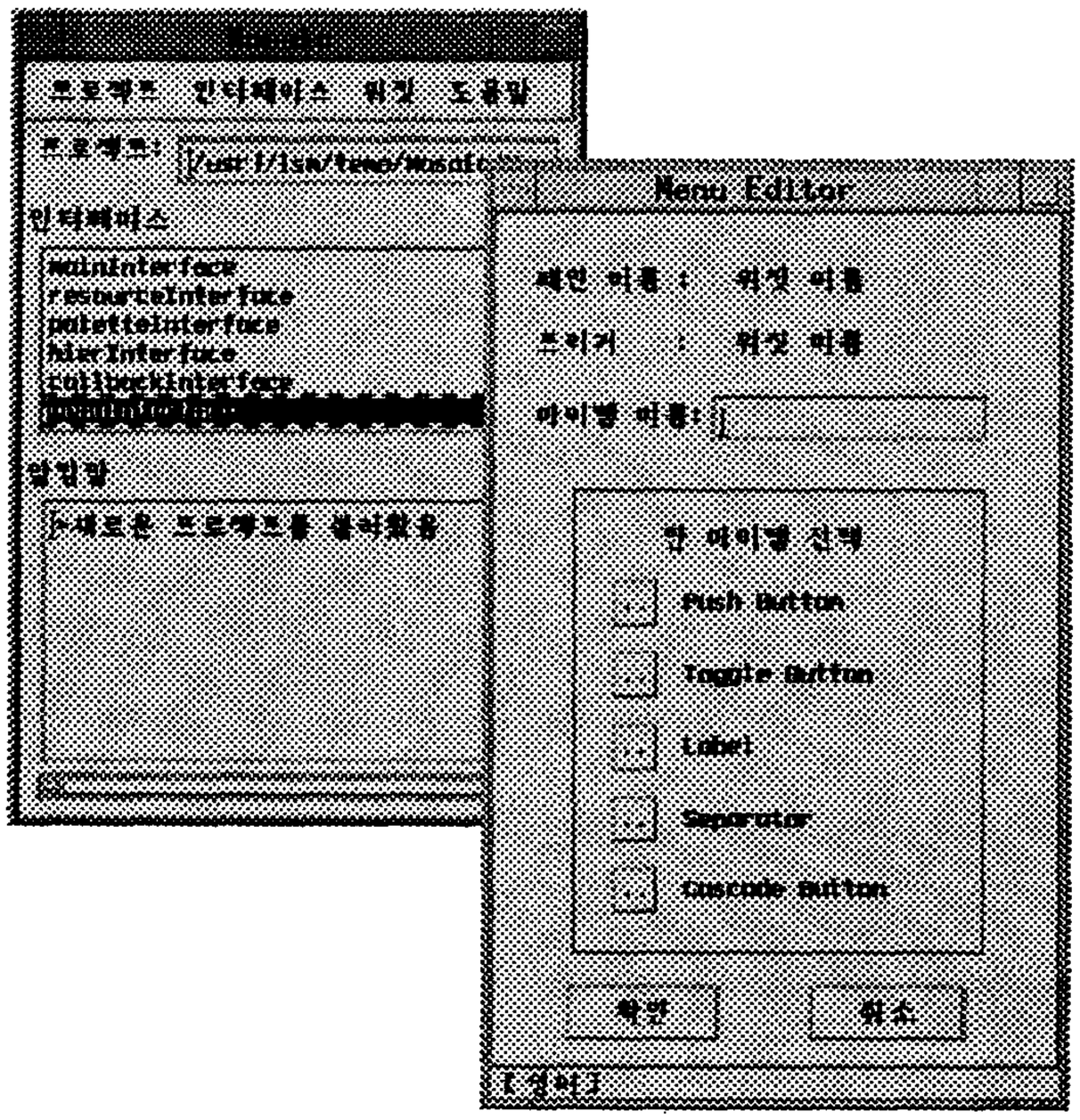
8) 끝내기

모자이크의 수행을 끝마치는 메뉴로써 모자이크의 모든작업을 작업영역에서 내리고 모자이크의 수행을 종료한다.

나. 인터페이스 메뉴

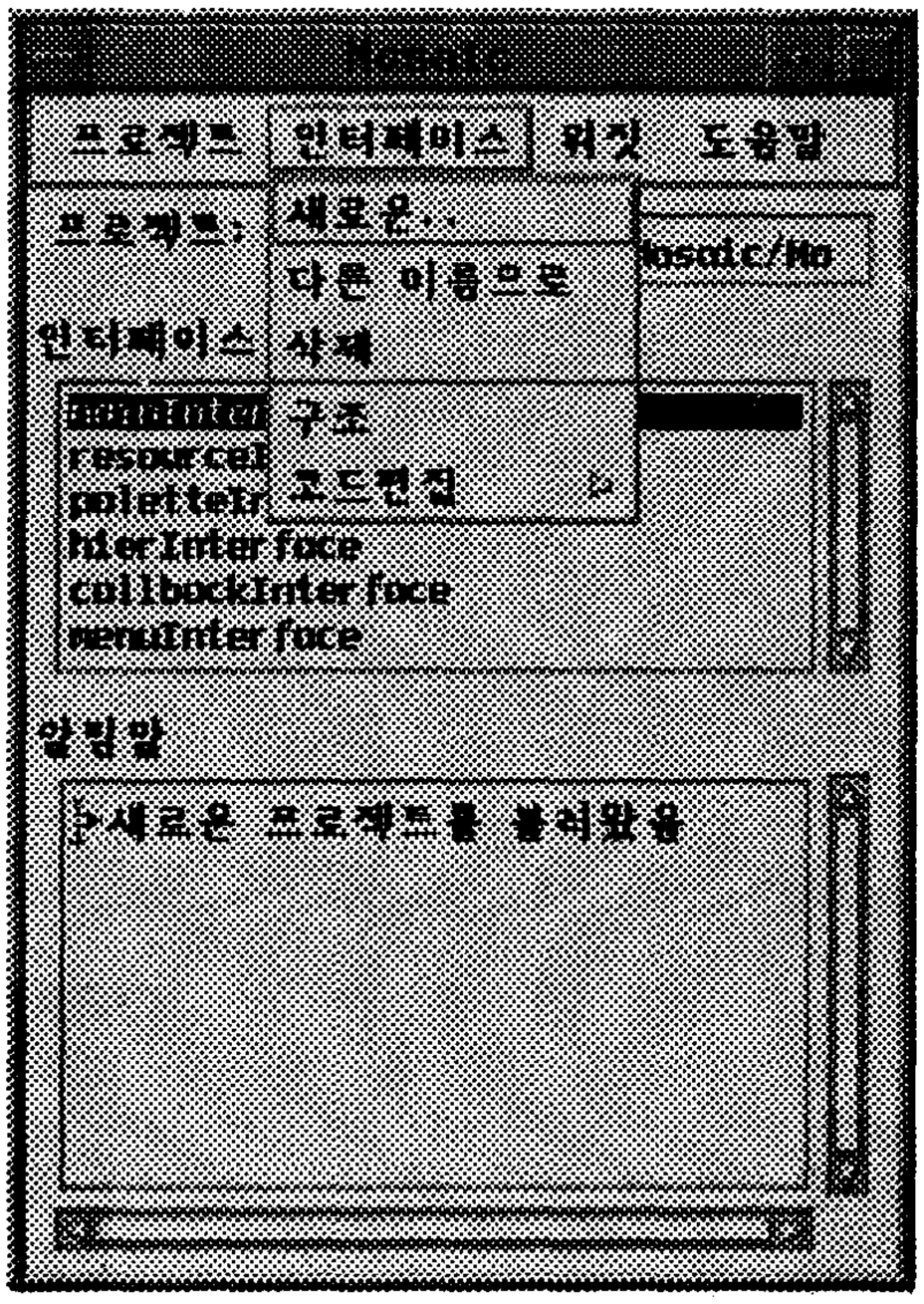
프로젝트와 관련된 인터페이스를 조작하는 메뉴로써 하나의 프로젝트에는 여러개의 인터페이스가 존재할 수 있으며 프로젝트 메뉴에서 하나의 프로젝트를 불러오면 이와 관련된 모든 인터페이스들이 주화면의 인터페이스 리스트에 나타나며 대상 인터페이스를 마우스로 선택한 후 작업을 수행하게 된다. 이때 선택된 인터페이스는 화면상에 나타나게 되며 이를 작업영역에서 내리고 싶을때(unload)는 주화면의 인터페이스 리스트에서 선택된 인터페이스를 다시한번 클릭(click)

해주면 된다. 또다른 인터페이스를 선택하면 이전의 인터페이스는 내려지게 되고 선택된 인터페이스가 화면상에 나타나게 된다. [그림 B-3-4]는 주화면의 인터페이스 리스트에서 해당 인터페이스를 선택했을 때의 화면이다.



[그림 B-3-4] 인터페이스 선택화면

인터페이스 메뉴에는 프로젝트와 관련된 새로운 인터페이스를 작성하는 '새로운..' 메뉴, 선택된 인터페이스의 이름을 다른이름으로 바꾸는 '다른 이름으로' 메뉴, 선택된 인터페이스를 삭제하는 '삭제' 메뉴, 선택된 인터페이스의 구조를 보여주며 이 구조를 편집할 수 있는 '구조' 메뉴 및 전역선언, 함수, 클래스의 코드를 편집할 수 있는 에디터로 구성된 '코드편집' 메뉴가 있다. [그림 B-3-5]는 인터페이스 메뉴 화면이다.

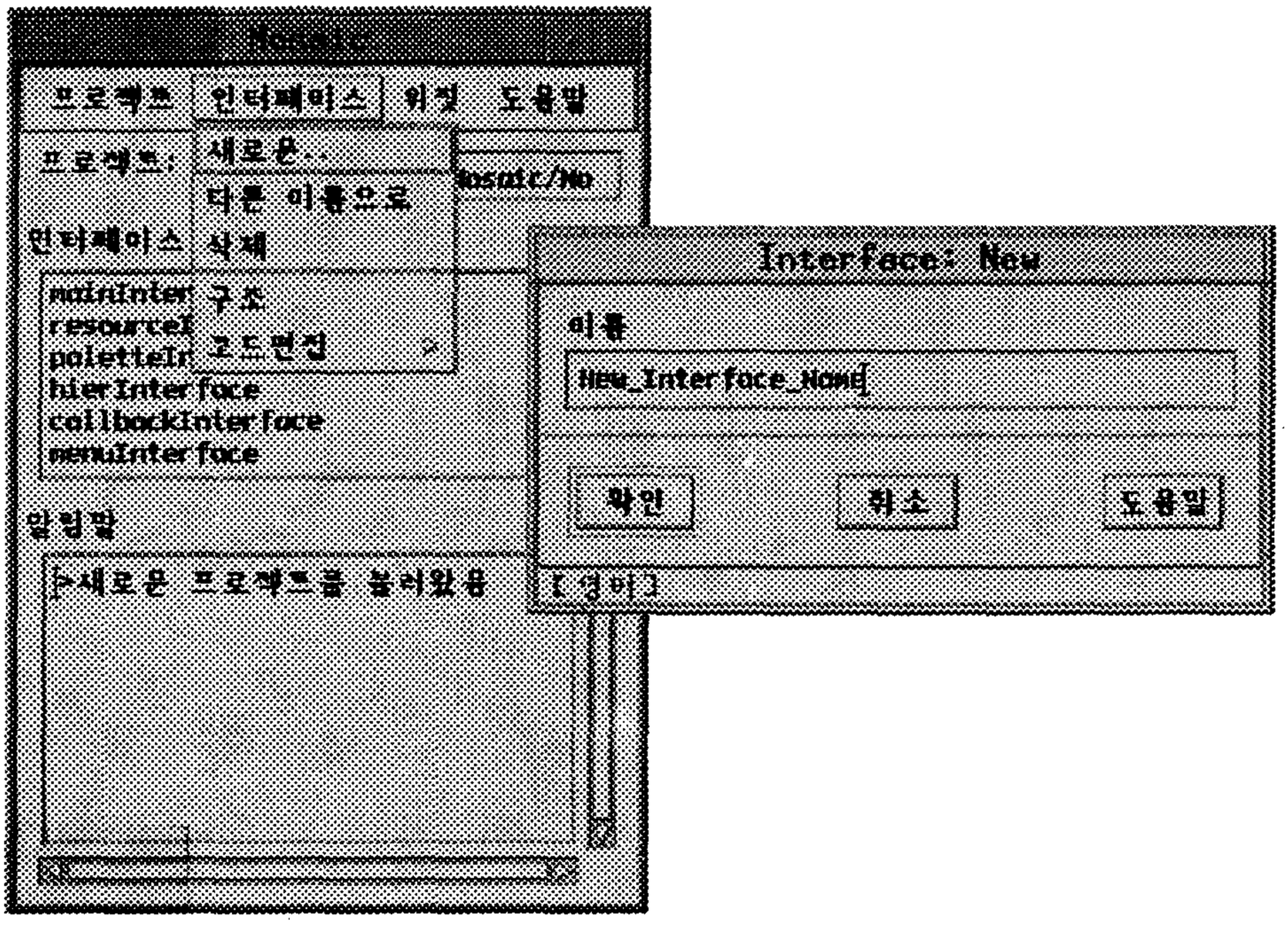


[그림 B-3-5] 인터페이스 메뉴

1) 새로운..

새로운 인터페이스를 작성하기 위하여 인터페이스의 이름을 입력하는 메뉴로써 '새로운..' 메뉴를 선택하면 'Interface:New' 다이아로그가 화면에 나타난다. 다이아로그의 이름 필드에 사용자가 새로운 인터페이스 이름을 입력한후 '확인' 버튼을 선택하면 입력한 인터페이스의 이름이 주화면의 인터페이스 리스트에 추가되며 '취소' 버튼을 선택하면 입력된 이름은 무시된다. 새로운 인터페이스를 작성하기 위해서는 주화면의 인터페이스 리스트에서 해당 인터페이스를 마우스

로 선택하면 작업이 시작된다. [그림 B-3-6]은 인터페이스 메뉴의 '새로운..' 메뉴 입력 화면이다.

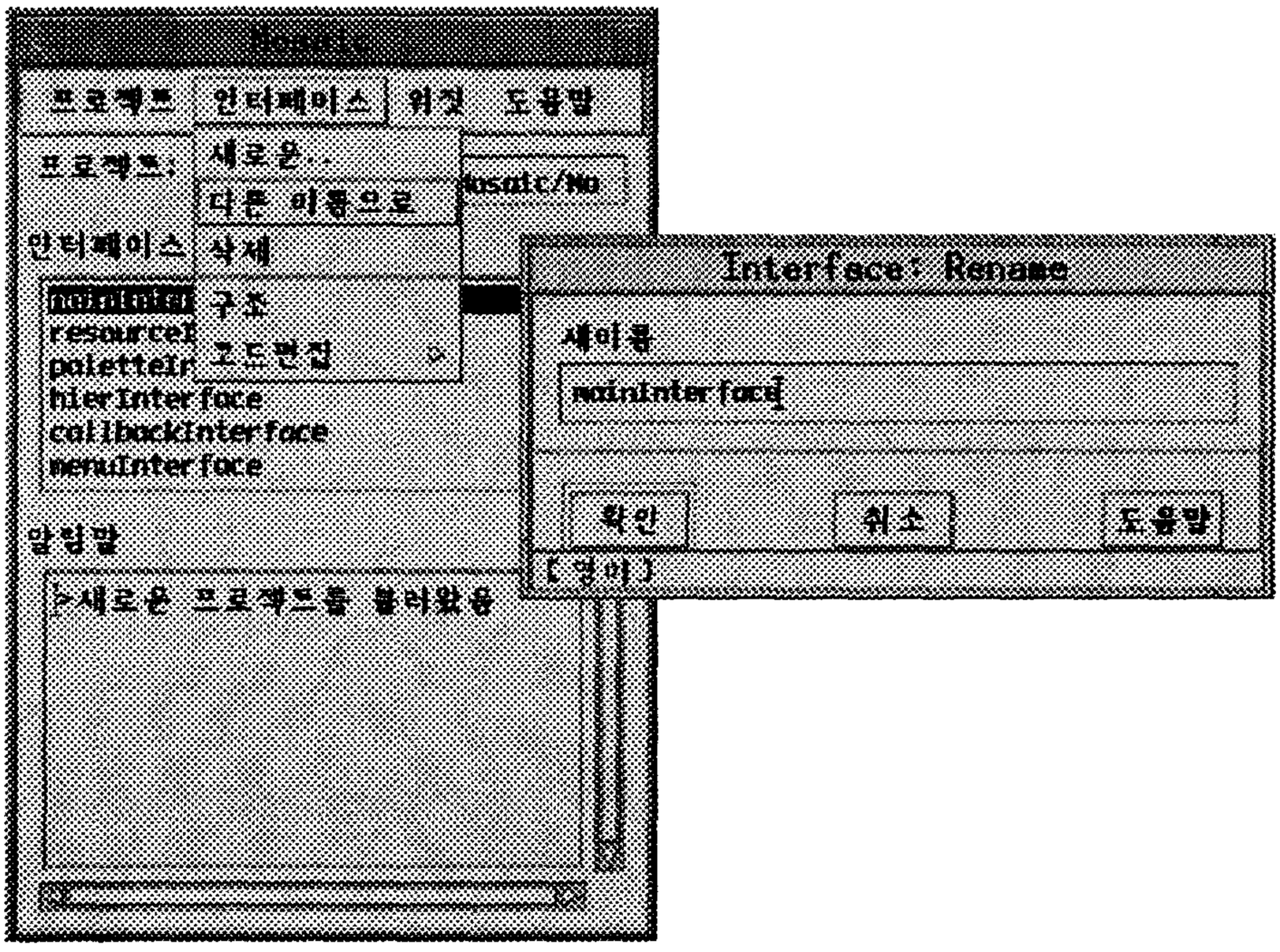


[그림 B-3-6] 인터페이스 '새로운..' 메뉴

2) 다른 이름으로

주메뉴의 인터페이스 리스트에서 선택한 인터페이스의 이름을 다른 이름으로 바꾸는(rename) 메뉴로써 '다른 이름으로' 메뉴를 선택하면 'Interface:Rename' 다이아로그가 화면에 나타난다. 새이름 입력필드에는 선택한 인터페이스의 이름이 나타나며 사용자가 원하는 이름을 입력한 후 '확인' 버튼을 선택하면 변경된

인터페이스의 이름이 주화면의 인터페이스 리스트에 등록이되며 '취소' 버튼을 선택하면 새로 입력한 작업은 취소되고 이전의 인터페이스 이름으로 남게 된다. 다른 이름으로 변경된 인터페이스는 그 구조와 인터페이스의 형태에는 아무런 변화가 없고 단지 해당 인터페이스의 이름만을 바꿀 뿐이다. [그림 B-3-7]은 인터페이스 메뉴의 '다른 이름으로' 입력 화면이다.



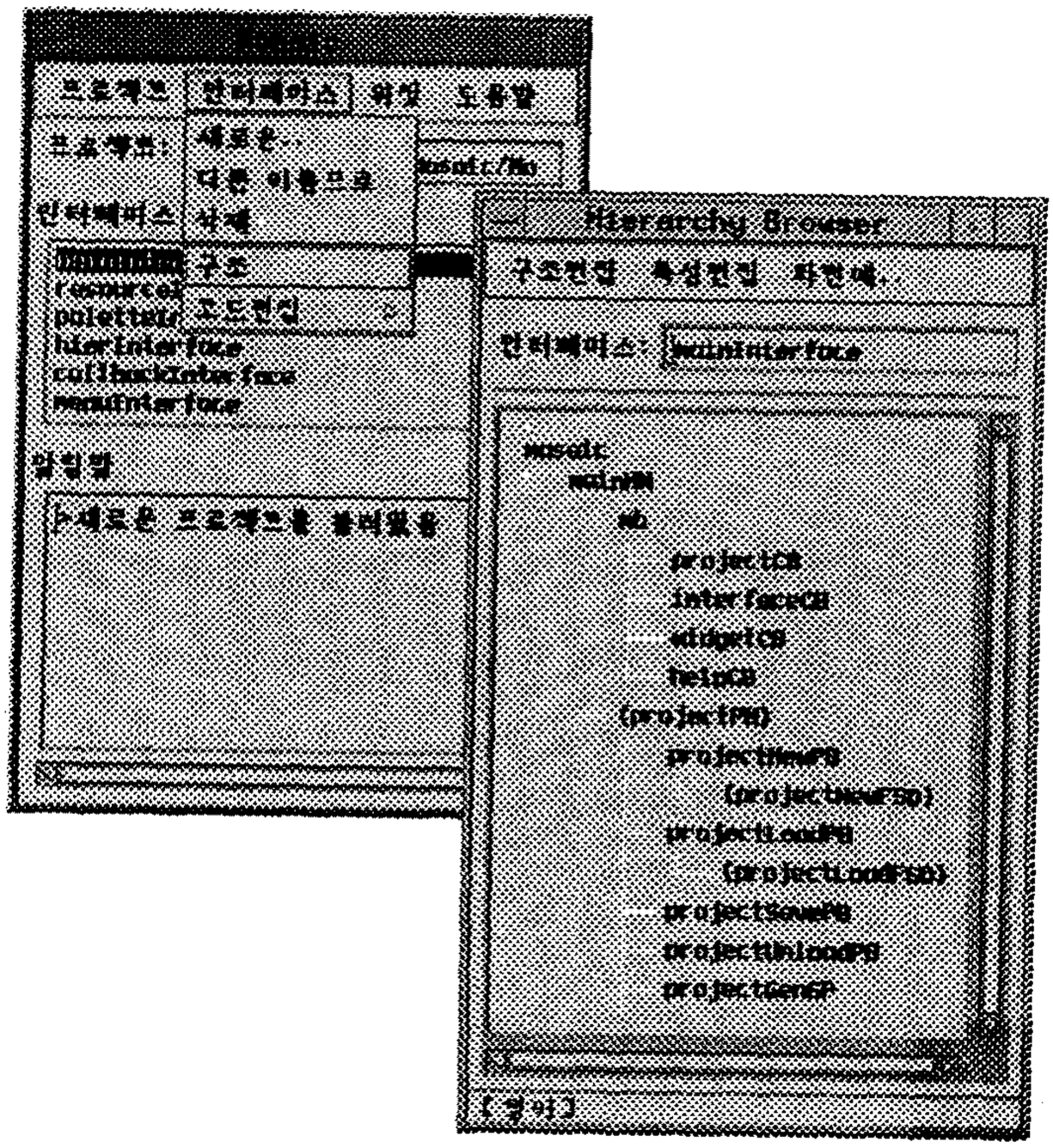
[그림 B-3-7] 인터페이스 '다른이름으로' 메뉴

3) 삭제

작성한 인터페이스중 필요없는 인터페이스를 삭제하는 메뉴로써 주화면의 인터페이스 리스트 필드에서 삭제하려는 인터페이스를 선택한 후 '삭제' 메뉴를 선택하면 선택된 인터페이스가 삭제되며 인터페이스 리스트로부터도 제거된다.

4) 구조

선택된 인터페이스의 구조를 편집하는 메뉴로써 '구조' 메뉴를 선택하면 구조 편집기(Hierarchy Browser)가 화면상에 나타난다. [그림 B-3-8]은 구조 편집기를 불러온 화면이다. 구조 편집기의 자세한 내용은 '4. 구조 편집기'에서 다루도록 한다.



[그림 B-3-8] 구조 편집기

5) 코드 편집

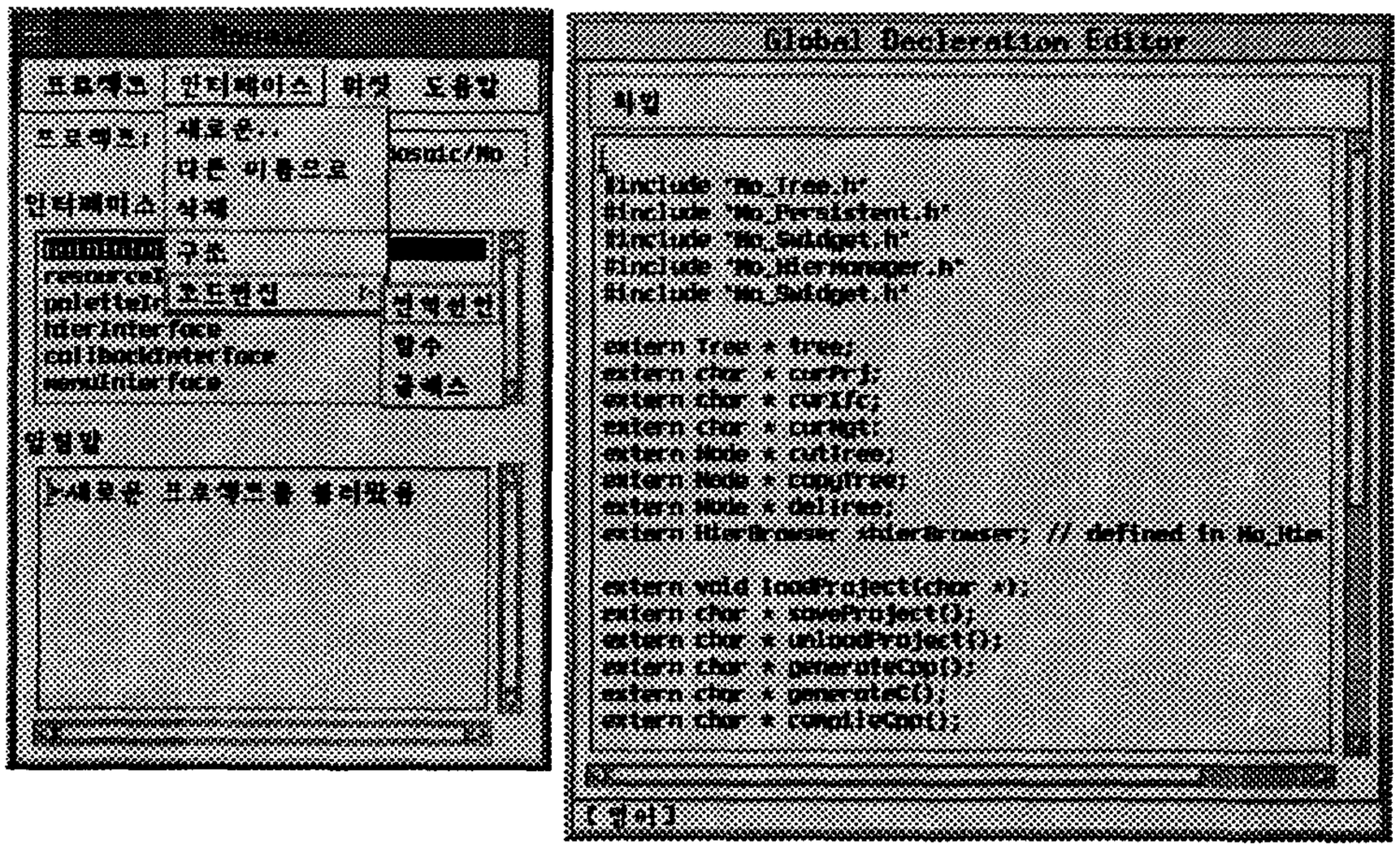
선택된 인터페이스와 관련된 전역선언(Grobal Decleration), 함수(function) 및 클래스(class)의 입력을 위한 전용 편집기(editor)를 불러오는 메뉴로써 이러

한 편집기가 불리어질때 기존에 작성된 데이타들이 이 편집기상에 출력된다. 추가, 변경, 삭제하고 싶은 내용들을 기존의 편집기에서와 같이 편집하면 된다.

가) 전역선언

선택된 인터페이스의 전역선언(Grobal Decleration)을 위한 편집기를 불러오는 메뉴이다. 여기서는 응용 프로그램을 위한 전역변수 및 전역함수등을 선언하고 헤더 화일등을 포함(include) 시키는 작업을 수행한다. 편집이 끝난후 화일메뉴의 '적용' 을 선택하면 편집된 내용이 해당 인터페이스와 관련된 헤더화일(InterfaceName_IF.C.H)에 저장되며 편집기는 화면상에 계속 존재하게되며 '적용후 닫기' 메뉴를 선택하면 편집된 내용을 해당 화일에 저장한후 편집기를 닫는다(close). '닫기' 메뉴는 변경된 내용을 무시하고 편집기를 닫게된다.

[그림 B-3-9]는 코드편집의 '전역선언' 편집기를 불러온 화면이다.



[그림 B-3-9] 전역선언 편집기

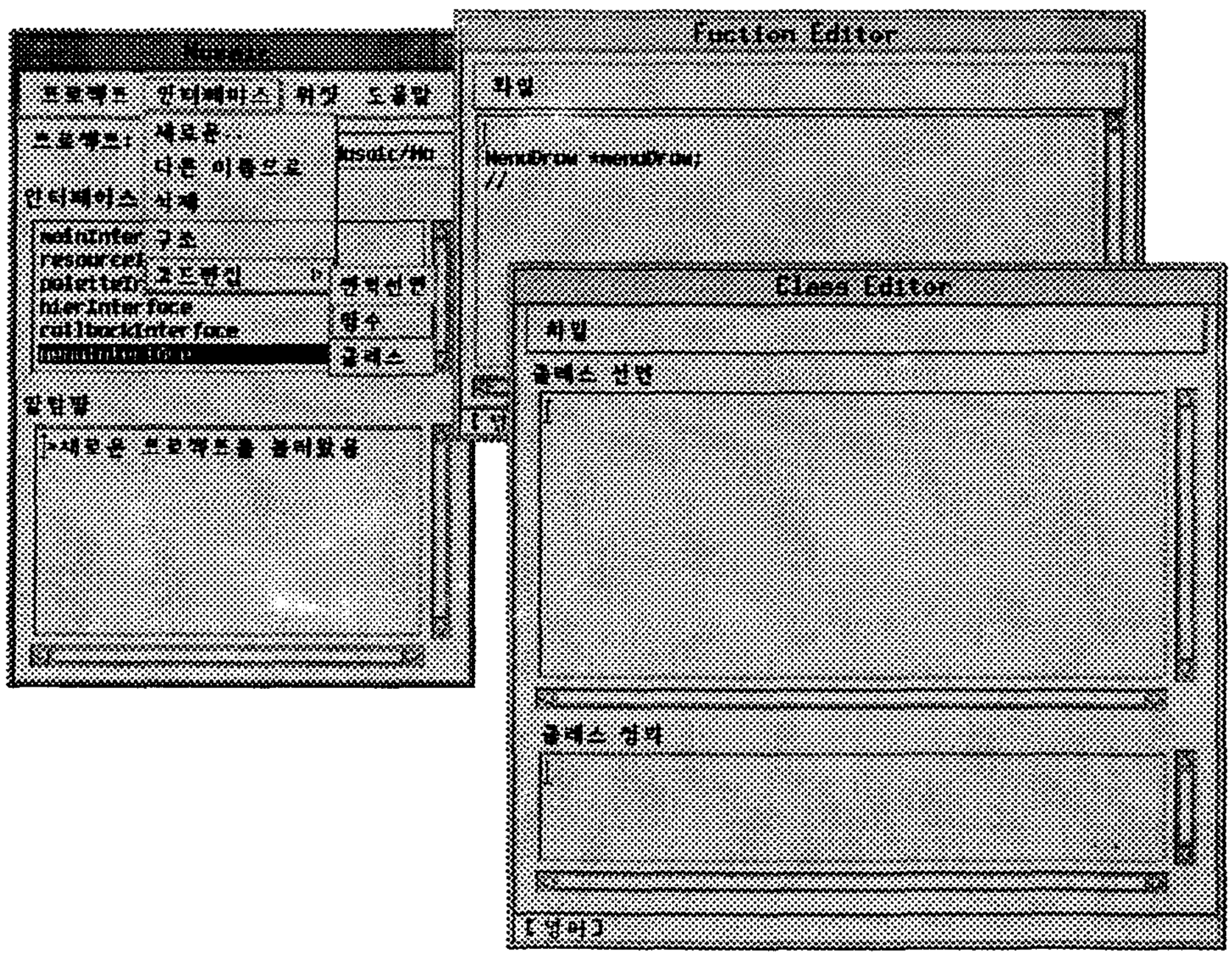
나) 함수

인터페이스와 관련된 함수(function)를 편집하는 메뉴로써 '함수'를 선택하면 화면상에 함수 편집기(function editor)가 불리워지며 '전역선언' 편집기의 메뉴와 같이 적용, 적용후 닫기, 닫기 메뉴가 있다. 편집된 내용은 해당 인터페이스와 관련된 InterfaceName_IF.C.C 화일에 저장된다.

다) 클래스

인터페이스와 관련된 클래스를 편집하는 편집기로써 클래스 선언(class declaration)부와 클래스 정의(class definition)부로 나누어 편집할 수 있다.

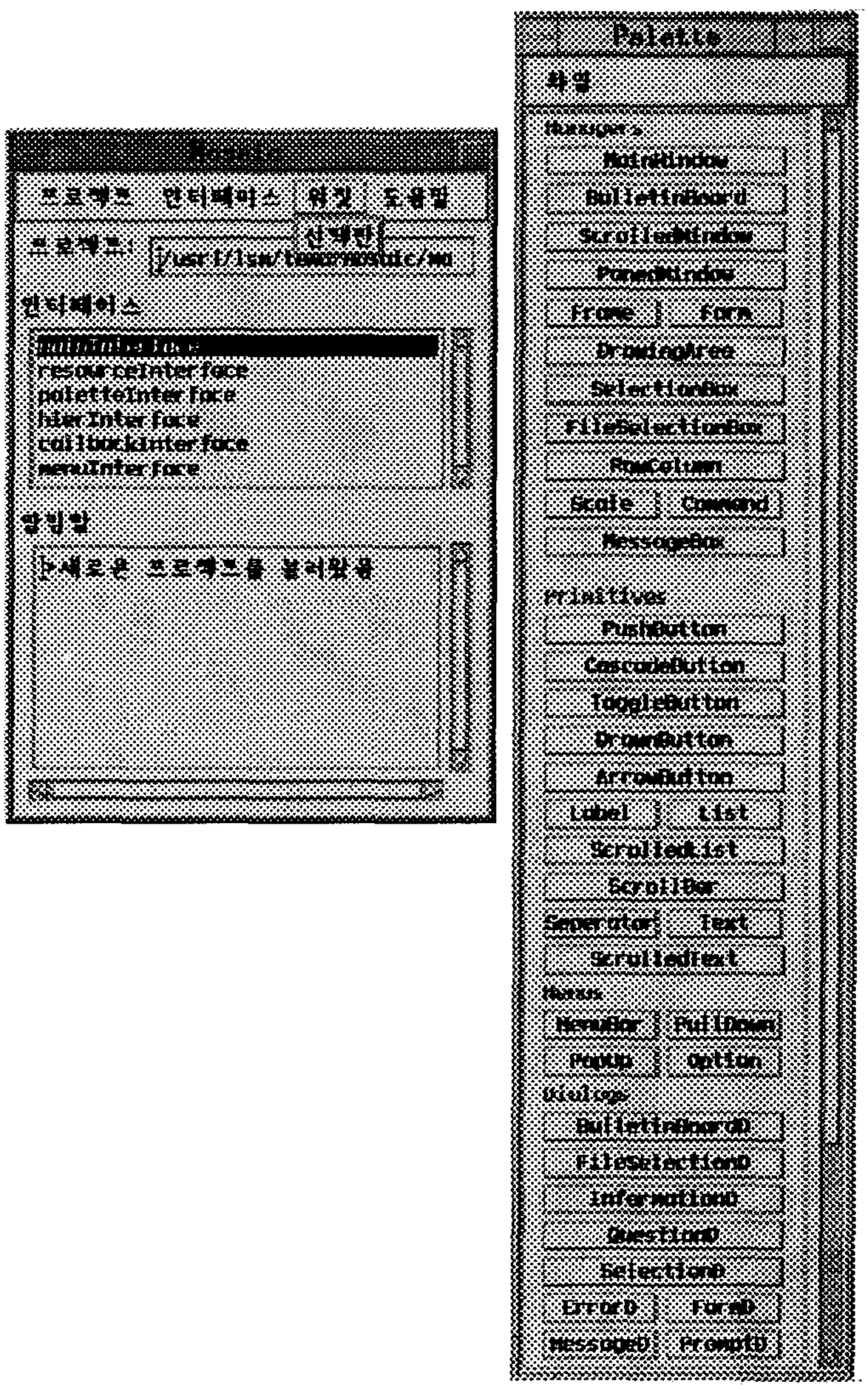
[그림 B-3-10]은 함수 편집기 및 클래스 편집기를 불러온 화면이다.



[그림 B-3-10] 함수 편집기 및 클래스 편집기

다. 위짓 메뉴

OSF/Motif에서 제공하는 다양한 위짓을 표시하며 필요한 위짓을 마우스로 선택 합성하여 사용자 인터페이스를 작성할 수 있도록 도와주는 선택판을 불러오는 메뉴이다.



[그림 B-3-11] 선택판(palette)

1) 선택판(palette)

Motif 에서 제공하는 위짓 구조(widget class hierarchy)를 기본으로 하여 매니져(manager), 프리미티브(primitives), 메뉴(menus), 다이아로그(dialogs), 개짓(gadgets) 으로 분류하였으며 필요한 위짓들을 올바른 합성규칙에 의해 적절히 선택하여 화면에서 레이아웃을 작성할 수 있도록 도와주는 도구이다. [그림 B-3-11]은 선택판(palette)을 불러온 화면이다.

4. 구조 편집기(Hierarchy Browser)

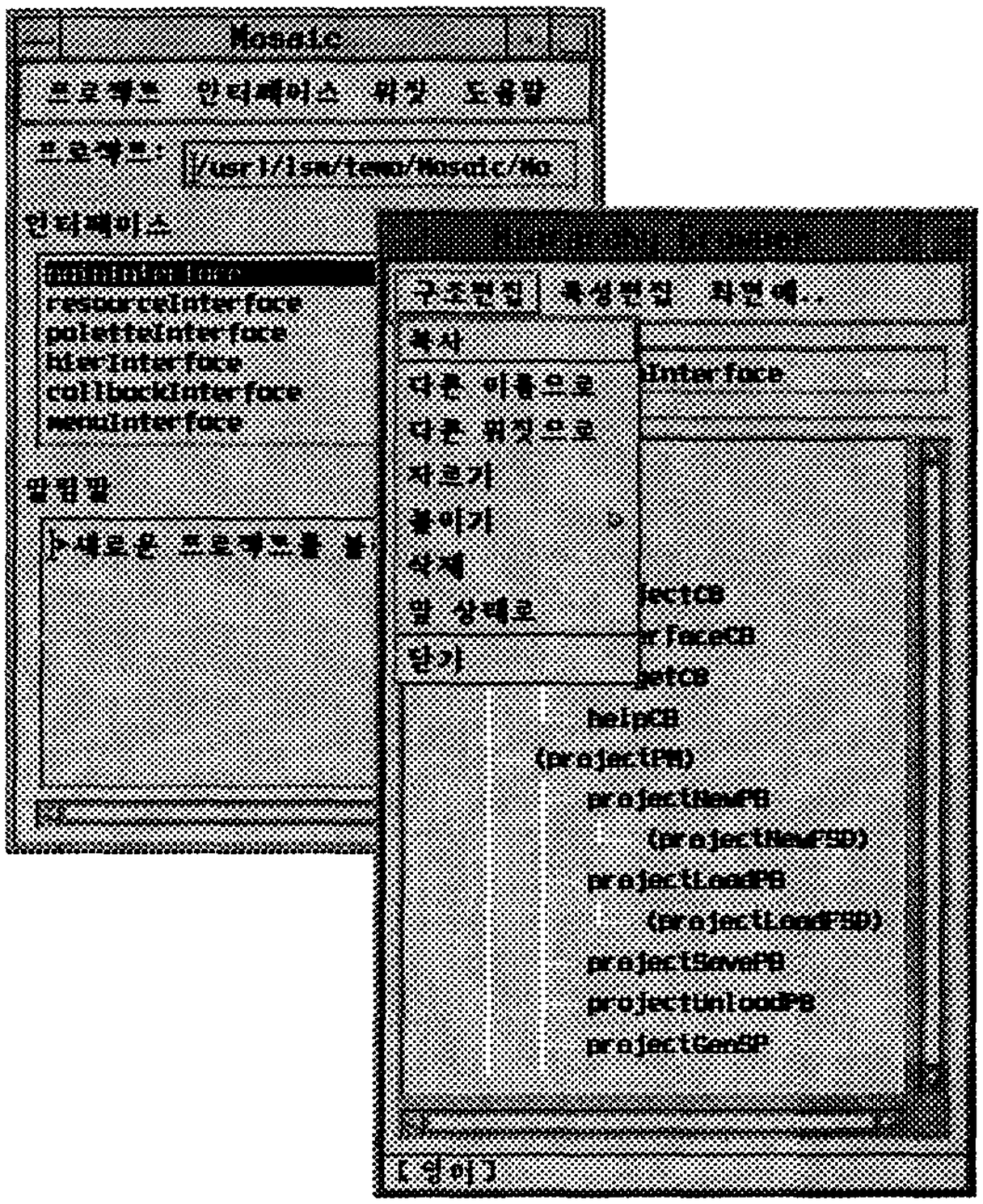
구조 편집기는 선택한 인터페이스의 구조를 트리형태로 보여주며 인터페이스의 구조를 편집하기 위하여 사용한다.

가) 구조편집

선택한 인터페이스의 구조를 편집하기 위해서는 구조 편집기상에서 해당 위짓을 마우스로 클릭(선택)한 후 작업을 시작하게 된다. 구조편집의 메뉴로는 해당 위짓을 일시적으로 복사해 놓는 '복사' 메뉴, 위짓의 이름을 변경하기 위한 '다른 이름으로' 메뉴, 위짓의 이름은 변경하지 않고 위짓의 클래스를 바꾸기 위한 '다른 위짓으로' 메뉴, 인터페이스의 구조를 변경하기 위한 '자르기', '붙이기' 메뉴, 선택된 위짓을 삭제하는 '삭제' 메뉴, 기존의 상태로 되돌리기 위한 '앞상태로' 메뉴 및 구조 편집기를 종료하는 '닫기' 메뉴가 있다. [그림 B-4-1]은 구조편집 메뉴 화면이다.

1) 복사

붙이기 메뉴를 위해 해당 위짓을 일시적으로 복사(copy)해 주는 메뉴로써 선택된 위짓의 자식이 있는 경우 자식도 같이 복사된다.

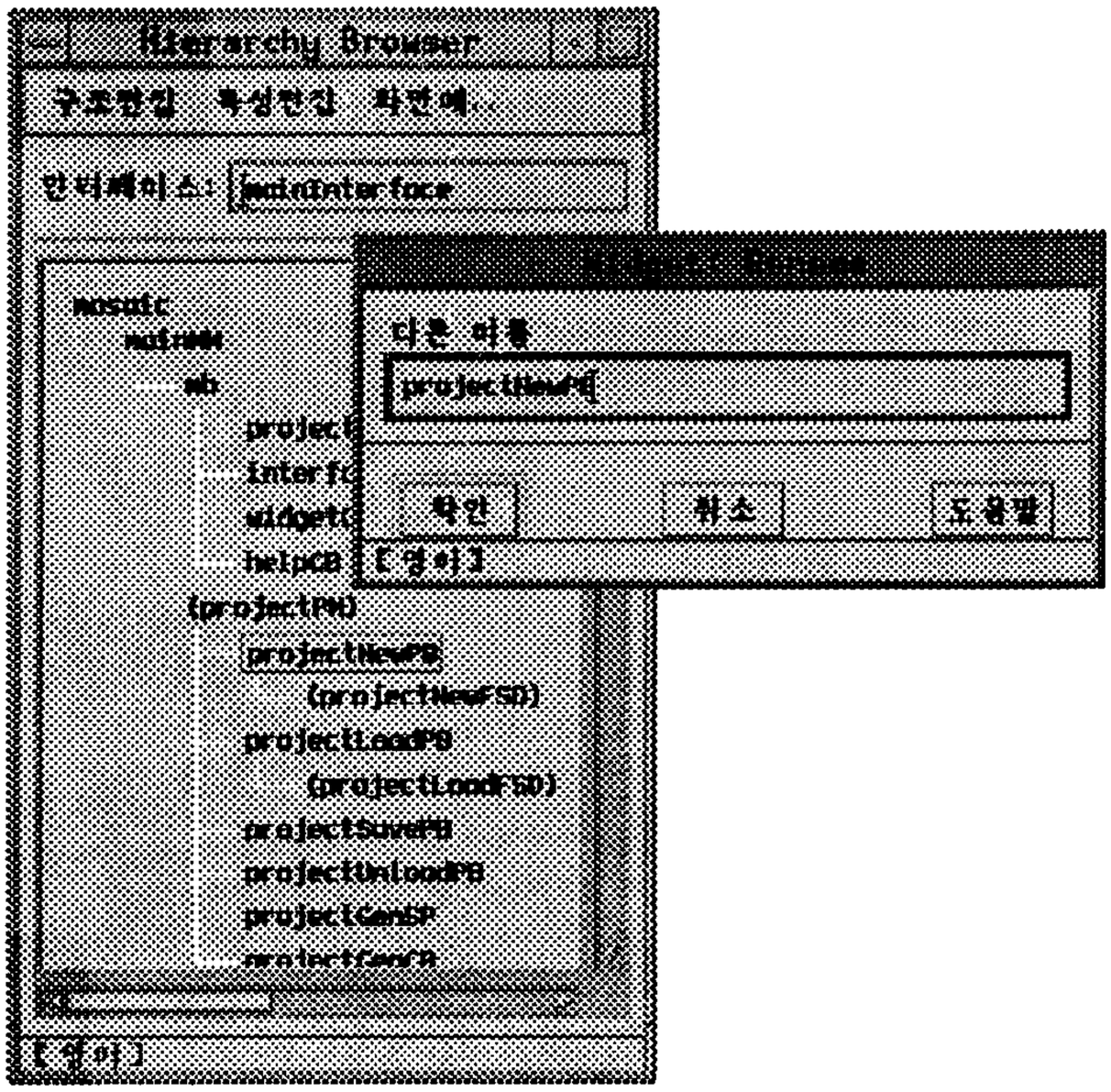


[그림 B-4-1] 구조편집 메뉴

2) 다른 이름으로

편집기상에서 선택한 위짓의 이름을 변경하기 위한 메뉴로써 메뉴를 선택하면 'Widget: Rename' 다이아로그가 화면에 나타나며 입력 필드에 바꾸고 싶은 이름을 입력한후 '확인' 버튼을 선택하면 편집기의 위짓 이름이 변경되게 되며, '취소' 버튼을 선택하면 기존의 이름이 변경되지 않고 다이아로그를 닫게 된다.

[그림 B-4-2]는 위짓의 이름을 변경하기 위한 화면이다.



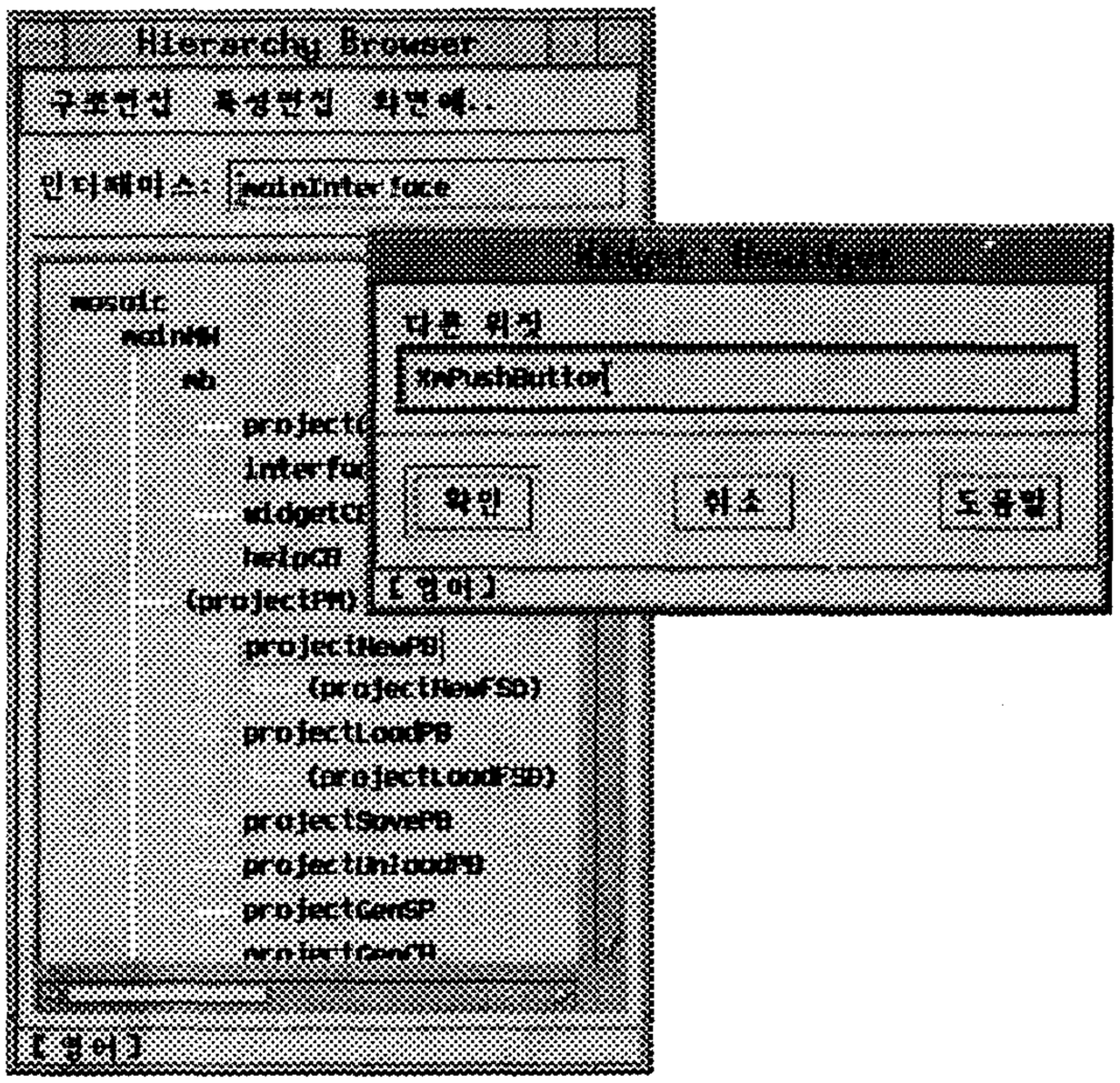
[그림 B-4-2] 위짓 이름변경 다이아로그

3) 다른 위짓으로

편집기상에서 선택한 위짓을 다른 위짓으로 바꾸기 위한 메뉴로써 메뉴를 선택하면 'Widget: Rewidget' 다이아로그가 화면에 나타나며 입력필드에 모자이크 시스템 혹은 Motif에서 제공하는 위짓의 이름을 입력한 후 '확인' 버튼을 선택하면 입력한 위짓으로 바뀌게 되며 '취소' 버튼을 선택하면 변경작업이 취소되며 다이아로그를 닫는다. [그림 B-4-3]은 위짓을 바꾸기 위한 다이아로그 화면이다.

4) 자르기

대상 위짓을 현재의 위치로부터 다른곳으로 옮기기 위해 구조 편집기상의 트리로 부터 일시적으로 자른다(cut). 잘려진 위짓은 구조 편집기상에서 지워지며 대상 위짓의 자식이 있는 경우에는 자식도 같이 잘라진다.



[그림 B-4-3] 위젯변경 다이아로그

5) 붙이기

자르기 메뉴에서 자른 위젯을 옮기고자하는 위치에 붙이는(paste) 메뉴로써 붙이고자 하는 위젯을 먼저 선택한 후 붙이기 메뉴를 선택한다.

가) 첫 자식으로

자르기 메뉴를 통해 잘려졌던 위젯군이 붙이기를 위해 선택한 위젯의 첫번째 자식으로 옮겨진다.

나) 다른 형제로

자르기 메뉴를 통해 잘려졌던 위젯군이 붙이기를 위해 선택한 위젯의 형제로 옮겨진다.

6) 삭제

현재 선택되어져 있는 위짓을 구조 편집기상에서 삭제 하는 메뉴로써 화면에 나타나있는 인터페이스에도 즉각적으로 반영된다. 선택된 위짓에 자식이 달려있는 경우 자식도 같이 삭제된다.

7) 앞 상태로

구조 편집기의 상태를 편집하기 이전의 상태로 되돌리기 위한 메뉴이다.

8) 닫기

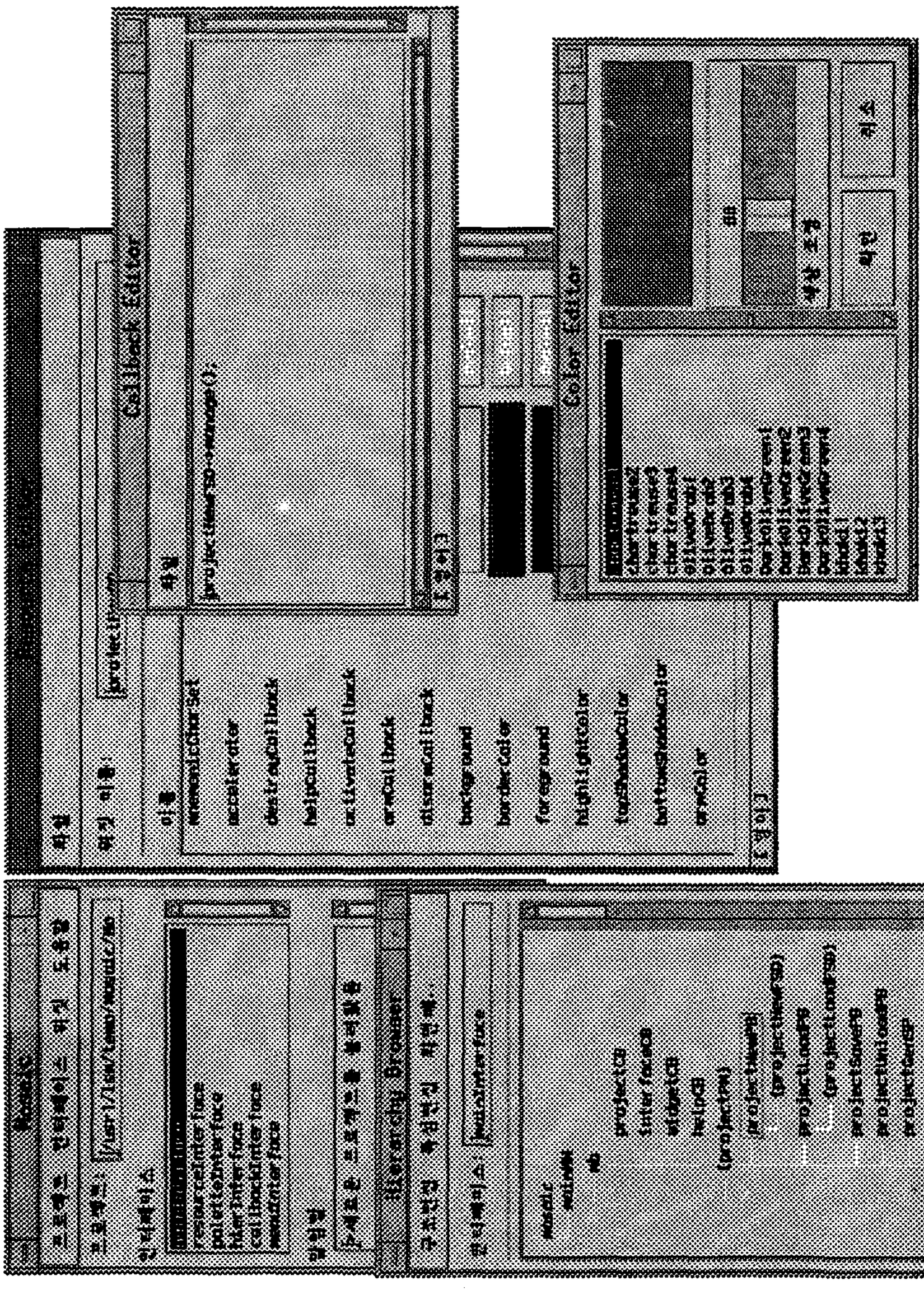
구조 편집기를 닫기(close) 위한 메뉴이다.

나. 특성편집

구조 편집기에서 선택한 위짓의 특성을 편집하기 위한 메뉴로써 위짓의 자원(resource)을 편집하기 위한 전용 편집기를 불러오는 'X 자원' 메뉴 및 위짓이 수행하는 행동과 관련된 '사건 처리기(event handler)' 메뉴 와 '행위 번역표(translation table)' 메뉴가 있다.

1) X 자원

구조 편집기에서 선택한 위짓에 관한 자원(resource)을 편집하기 위한 전용 편집기로써 선택된 위짓의 자원에 대해 현재의 정보를 보여주며 새로운 값을 입력한 후 '화일' 메뉴의 '적용'을 선택하면 변경된 자원에 대한 정보가 저장된다. '화일' 메뉴의 '적용후 닫기'를 선택하면 변경된 자원에 대한 정보가 저장되고 편집기는 닫히게 된다. '화일' 메뉴의 '닫기'를 선택하면 변경 작업은 무시되고 편집기는 닫히게 된다. 자원 편집기 메뉴 색상에 대한 편집을 수행하는 색상 편집기(color editor) 및 콜백(callback)을 편집할 수 있는 편집기인 콜백 편집기(callback editor)를 준비하고 있는데 해당되는 콜백이나 색상의 값 필드를 마우스로 클릭하면 이러한 에디터들이 자동으로 구동된다. [그림 B-4-4]는 자원 편집기, 색상 편집기 및 콜백 편집기를 나타낸 화면이다.



[그림 B-4-4] 자원 편집기, 색상 편집기 및 콜백 편집기

색상 편집기의 사용은 색상에 대한 리스트중 원하는 색상을 마우스로 선택하거나 색상 조정 슬라이드 버튼을 좌우로 움직이면 해당 색상 및 색상의 이름이 나타난다. 이때 사용자가 '확인' 버튼을 선택하면 변경된 색상이 자원편집기에 반영되며 색상편집기를 닫게되며, '취소' 버튼을 선택하면 변경 작업은 취소되고 편집기는 닫히게 된다.

콜백 편집기는 위짓에 관련된 콜백을 편집하는 전용 편집기로써 초기의 상태는 현재 서술된 콜백의 내용을 출력해준다. 사용자는 일상의 편집기와 마찬가지로 편집 작업을 수행한후 '화일' 메뉴의 '적용후 닫기'를 선택하면 편집된 내용이 저장되며 '닫기'를 선택하면 변경 작업은 취소되고 에디터를 닫게 된다. 색상 편집기나 콜백 편집기를 통해 편집된 내용은 자원 편집기의 '적용' 버튼을 선택하는 순간에 실제로 반영된다.

다. 화면에..

구조편집기 상에서 선택한 위짓에 대해 사용자가 작성한 인터페이스 화면상에서 해당 위짓을 보이게 하거나(manage), 감추는(unmanage) 메뉴이다.

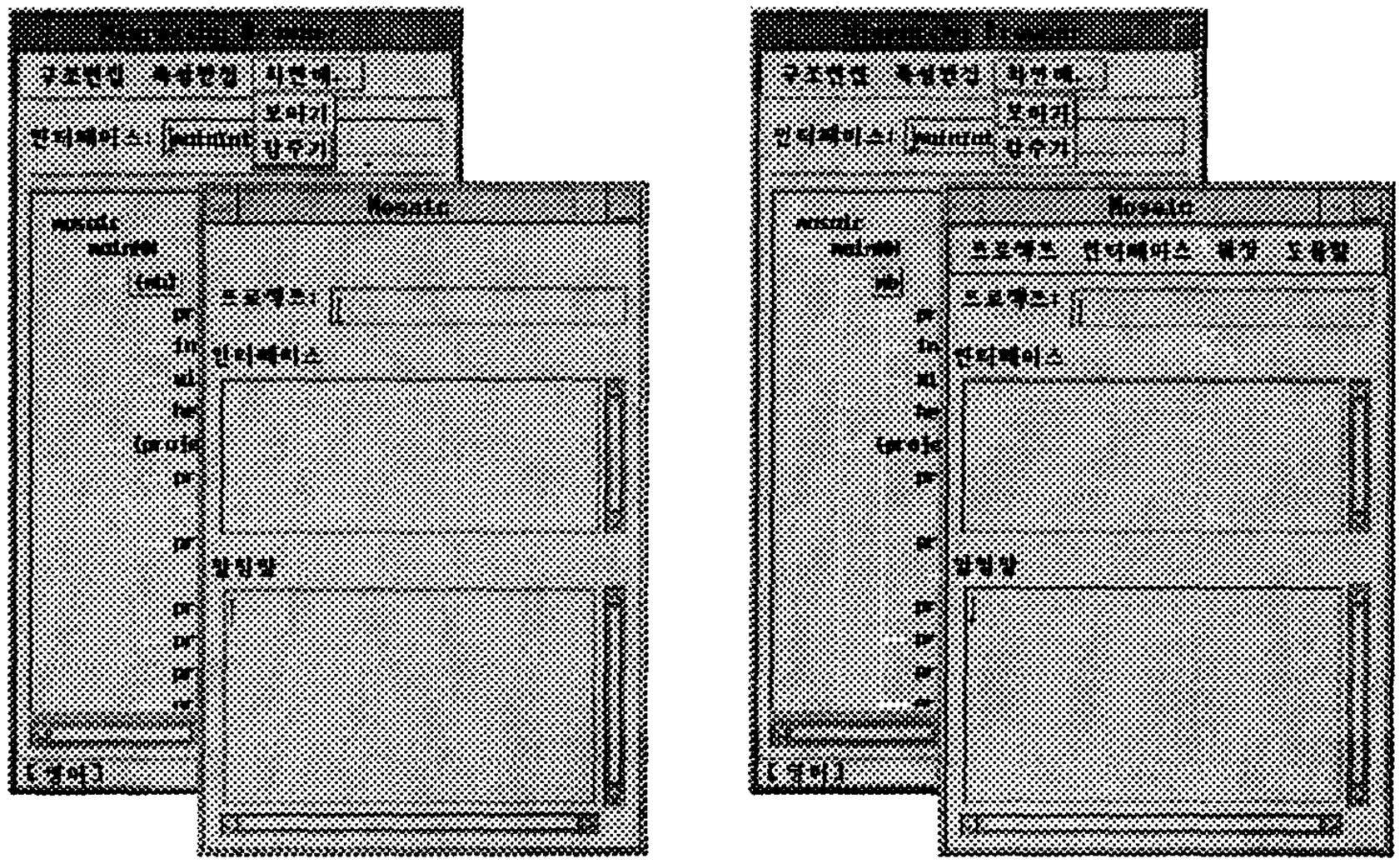
1) 보이기

구조 편집기상에서 선택한 위짓을 화면에 보이게(manage)하는 메뉴로써 이미 보여지고 있는 위짓에 대해서는 의미가 없고 감추어진 위짓을 화면상에서 보려고 할때 필요한 메뉴이다.

2) 감추기

구조도상에서 선택한 위짓을 인터페이스 화면에서 감출때 사용하는 메뉴이다.

[그림 B-4-5]의 왼쪽 화면은 기존에 작성된 인터페이스에서 메뉴바(menu bar)를 감춘 화면이고, 오른쪽 화면은 감추어진 위짓을 다시 보여주는 화면이다.



[그림 B-4-5] 화면에 감추기/보이기 화면