인트라넷 구축 도구를 위한 프레임워크 모델러의 설계 및 구현

(Design and Implementation of a Framework Modeler for Intranet Construction Tool)

이 창 목 † 유 철 중 †† 장 옥 배 †† 이 상 덕 †††
(Chang-Mog Lee) (Cheol-Jung Yoo) (Ok-bae Chang) (Sang-Duck Lee)

요 약 객체지향 언어의 등장과 함께 재사용의 중요성이 증대되면서, 개발자는 기존에 있는 프레임워크를 재사용하기 위해 핫 스팟(Hot Spot)을 수정하여 개발자의 의도에 적합한 시스템을 구축함으로써 개발기간 단축은 물론 보다 견고하고 안정적인 개발을 하고자 한다. 이러한 작업을 하기 위한 환경을 제공하는 도구가 RAD(Rapid Application Development) 도구이다. 객체지향 프로그램 개발자라면 RAD 도구의 필요성을 누구나 다 인식하고 있으며 여러 업체에서 이러한 RAD 도구를 개발하고 있다.

본 논문에서는 사용자 중심의 인트라넷환경 구축도구 기술개발의 일환으로 프레임워크를 기반으로 하는 프로그램 생성을 위한 모듈단위의 모델러를 설계 및 구현하였다. 본 모델러는 구현언어로 플랫폼에 독립적인 Java 언어를 사용하였고, 객체 다이어그래밍 기술을 포함하고 있는 OMT 에디터를 분석하여 기술을 적용하되 OMT 에디터에서는 지원하지 않았던 UML(Unified Modeling Language) 표기법을 지원하고 있으며, 기존의 MVC(Model-View-Controller) 구조가 가질 수 없었던 여러 뷰(View) 사이에 발생하는 메시지를 전달하는데 있어 Agent라는 자체 설계 패턴을 개발하여 적용함으로써 본 도구를 이용하여 일관된 개발을 할 수 있도록 구조화하였다. 따라서 본 논문에서 설계 및 구현한 도구는 사용자의 요구사항 변경이나 기능확장시 보다 유연하게 대처할 수 있는 특징을 가지고 있다.

Abstract As reusability becomes recognized more importantly, with the introduction of Object-Oriented Programming Languages, developers not only want to reduce development duration, but also to develop a proper system robustly and safely by renovating the Hot Spot in order to reuse the existing framework. When we perform these works, we need the development environment which is the Rapid Application Development tool, and the RAD tools provide us with the convenient development environment. The need of RAD tools is recognized by every Object-Oriented programmer, and many business enterprises are developing them.

In this paper, we will present a design and implementation of module-based modeler as a method for developing technique to construct user-driven Intranet environment for the generation of the program based on the framework. The framework modeler used Java language that is independent on platform, and applied the technique of OMT editor that provides the UML notation partially. Additionally, The modeler also includes the notations that are not supported in OMT editor. In addition to this characteristic, it is structured to develop system consistently with applying the Agent pattern, which is a design pattern suggested by ourselves, to send messages occurred between various Views. The existing MVC(Model-View-Controller) architecture does not have this function. Thus, this tool has a flexibility when user's requirements are changed, or functions are extended.

† 학생회원 : 전북대학교 대학원 전산통계학과
cmlee@cs.chonbuk.ac.kr
†† 종신회원 : 전북대학교 컴퓨터과학과 교수, 전북대학교 영상·정보신기술연구소 연구원
cjyoo@moak.chonbuk.ac.kr
okjang@moak.chonbuk.ac.kr
††† 비 회 원 : 한국전자통신연구원 정보화기술연구부 소프트웨어시험연구센터 센터장
lsd@etri.re.kr
논문접수 : 2000년 7월 27일
심사완료 : 2000년 12월 23일

1. 서 론

소프트웨어 분야의 발전속도는 하드웨어의 발전속도에 미치지 못하는 것이 현실이다. 이러한 여건 속에서도 소프트웨어 개발자들은 특정 방법론을 적용하여 소프트웨어를 개발하고 있다. 과거의 구조적 프로그래밍 기법인 절차지향 언어에서 객체가 주가 되는 객체지향 언어로 프로그래밍 기법이 빠르게 변화해감에 따라 소프트웨어 산업에도 커다란 변화가 일고 있다[1]. 이러한 변화는 기존의 구조적 프로그래밍 기법과 달리 객체지향적 프로그래밍 기법중 재사용성이 강조되어 시시각각으로 변하는 사용자의 요구사항에 부응하기 위한 방법이 되었다. 즉, 빠르게 변화되는 소프트웨어를 처음부터 새로운 개념으로 작업하지 않고 기존의 방법을 부분적으로 변형하거나 확장 개발하여 소프트웨어를 빠르고 쉽게 재사용함으로써 비용뿐만 아니라 성능향상이라는 두 가지의 시너지 효과를 얻을 수 있기 때문이다[2]. 이러한 재사용의 일환으로 프레임워크라는 용어가 등장하였다.

프레임워크라는 용어는 본래 IBM사의 San Francisco 프로젝트에서 유래되었다. 프레임워크는 컴포넌트들간의 상호 작용과 객체들의 집합을 운영하는 조직화된 환경을 제공하며 객체들의 결합규칙을 정의한다. 이러한 서브시스템 제공을 통하여 코딩 및 디버깅 시간을 감소시키며 고수준의 추상화를 제공함으로써 복잡성을 줄이고 다른 프레임워크간의 상호연동을 지원하기 때문에 적은 양의 코드를 작성할 수 있는 장점이 있다[3]. 80년대부터 객체지향 방법론이 나오기 시작하여 90년대 초에 여러 방법론이 제기되었는데, OOD(Object-Oriented Design), OMT(Object-Modeling Technique), OOSE(Object-Oriented Software Engineering) [4, 5, 6]등이 대표적인 예이다. 그러나 이러한 많은 방법론들은 오히려 표준화된 일관성의 부족으로 개발자들이 각각의 방법론을 적용한 도구들로부터 생성된 결과물간의 상호 호환성 문제와 같은 혼란을 초래하였다. 그래서 OMG(Object Management Group)에서 1997년 객체지향 표준 모델 언어로 UML(Unified Modeling Language)을 발표하였다. 이로 인해 객체지향 방법론 및 UML을 지원하는 여러 RAD 도구들이 개발되었는데, Rational사의 Rose, OMG의 OMT 에디터, 그 외에 Select Enterprise[7], ObjectTeam 등이 대표적인 예이다.

이러한 도구들은 순공학(Forward Engineering)과 역공학(Reverse Engineering) 및 개발프로세스지원 등의 기능을 갖추고 있다[8]. 그러나 이러한 도구들은 복잡한 사용자 기능으로 인하여 사용자로 하여금 사용법을 익히는 데에도 많은 시간을 요구하며, 소프트웨어 설계보다는 주로 단순한 시스템 모델링에 사용되는 만큼 그 이용범위가 제한적이다. 따라서 본 논문에서는 간단하면서도 사용자중심의 소프트웨어 개발을 지원하도록 하기 위해 앞서 개발된 Model-View-Controller(이하 MVC) 구조 기반의 OMT 에디터를 개선한 프레임워크 기반 모델러를 설계 및 구현하였다. 또한 MVC 구조 중에서 컨트롤러의 독립성을 보장하기 위하여 위임형 이벤트 모델을 사용하고 서로 다른 뷰 사이의 메시지 전달을 위하여 *Agent* 패턴이라는 자체 설계 패턴을 적용함으로써 여러 개의 뷰와 뷰사이에 존재하는 종속성의 개념을 명확히 분리하였다.

분석한 MVC 구조는 OMT 에디터에 적용된 바 있고 사용자에게 편리성을 제공해 줄 수 있는 도구에 적용 가능하지만, 각각의 요소간의 결합력을 완화시키기 위해서는 MVC 구조를 좀 더 명확히 분리하여 유지보수 및 기능확장이 용이하도록 해야 한다[9].

명확히 분리된 MVC 구조는 요소들간의 결합력이 낮아지게 되므로 빠른 유지보수와 사용자의 요구사항에 맞는 기능을 쉽게 추가하여 개발비용을 낮출 수 있을 것으로 기대된다. 구현은 JDK 1.2 버전을 사용하였다. 본 논문의 구성은 다음과 같다. 2장 관련연구에서는 이와 같이 본 도구를 설계 및 구현하는데 기반이 되는 OMT 에디터의 내부구조에 대하여 살펴보고 3장에서는 도구설계에 대하여 4장에서는 도구의 구현 및 평가에 관한 내용을, 마지막으로 5장에서는 결론으로 기대효과 및 향후연구방향을 제시한다.

2. 관련 연구

첫 번째 관련연구로 본 논문의 기반이 되는 프레임워크의 개념에 대하여 먼저 알아본다. 두 번째는 프레임워크 모델러를 설계 및 구현하기 위하여 소스코드가 공개되어 있는 기존의 OMT 에디터를 분석하였다. OMT 에디터는 동적모델링과 객체모델링을 할 수 있는 통합 에디터[10]이며, 이 도구의 소스코드에서 나타난 패턴들을 추출하여 분석함으로써 본 논문에서 재사용하고자 하였다.

2.1 프레임워크의 개념

프레임워크는 특정 도메인에서 다양한 애플리케이션들이 보편적이며 공통으로 사용하는 데이타 및 기능들을 모델링하고 이를 부분적으로 구현한 재사용 부품이다[11]. 또한 완전한 애플리케이션이 아닌 부분적인 애플리케이션으로 설계 정보와 코드의 재사용을 동시에

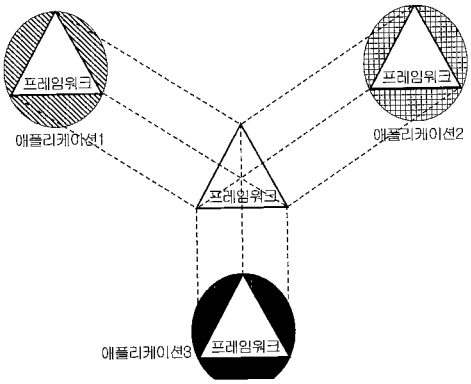


그림 1 프레임워크의 재사용

지원한다[11]. 프레임워크 자체는 아직 구현되지 않은 부분이 포함되어 있다. 애플리케이션 개발자는 이 부분을 변경함으로써 애플리케이션 요구사항을 만족하는 프로그램을 작성하게 된다.

(1) 추상클래스와 구체클래스

추상클래스는 구현되지 않은 채로 남겨진 하나 이상의 오퍼레이션을 가진 클래스이다. 어떤 오퍼레이션이 구현되지 않았기 때문에 추상클래스는 인트턴스를 가질 수 없고 상위 클래스로만 사용된다. 그러므로 하위 클래스를 정의하기 위한 템플릿으로 사용하기 위해 설계된다. 반면에 프레임워크는 상호 작용하는 객체들의 집합에 대한 설계이다. 추상클래스를 제외한 클래스는 구체클래스이다. 추상클래스를 구현한 구체클래스는 구현이 필요한 모든 오퍼레이션의 구현을 제공하며 다른 오퍼레이션의 구현을 상속받는다. 프레임워크는 보통 추상클래스와 미리 구현된 일부 구체적 클래스로 이루어진다. 그림 2는 추상클래스와 구체클래스를 보여주고 있다.

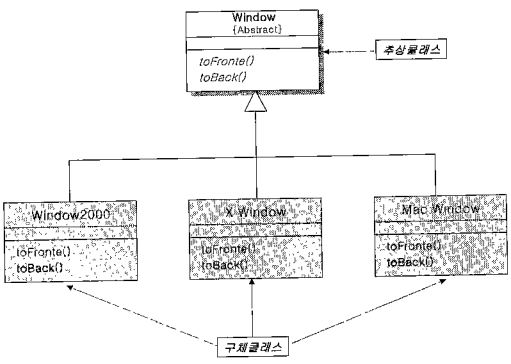


그림 2 추상클래스와 구체클래스의 관계

Window 클래스는 추상 클래스로 구현 부분이 없는 추상 오퍼레이션인 'toFront()'와 'toBack()'을 정의하고 있으며 구체적 클래스인 Window 2000, XWindow, Mac Window는 Window의 추상 오퍼레이션을 자신들에 맞게 구현된다.

(2) 핫 스팟(Hot spot)과 프로즌 스팟(Frozen spot)

핫 스팟은 애플리케이션 개발자들이 프레임워크를 확장시킬 수 있는 부분으로서, 완성도가 높은 프레임워크 개발을 위해 특정 도메인의 핫 스팟을 식별하는 과정이 필수적이다. 핫 스팟은 애플리케이션 도메인에 있는 일종의 슬롯(slot)이며, 개발자는 이러한 슬롯들을 자신들의 코딩으로 채우게 된다. 프로즌 스팟은 프레임워크 중에서 애플리케이션 개발시에 변경되지 않고 고정된 부분을 말한다.

(3) 훅 메소드(Hook method)와 템플릿 메소드(Template method)

프레임워크 구현시에 메소드는 크게 훅 메소드와 템플릿 메소드로 구분된다. 템플릿 메소드는 훅메소드에 기초한 메소드로 프레임워크의 프로즌 스팟을 구현하고 훅 메소드는 추상 메소드, 구체적 메소드, 템플릿 메소드로 구분되며 프레임워크의 핫 스팟을 구현한다. 그림 3은 훅 메소드와 템플릿메 메소드가 한 클래스에 있는 경우이지만 두 메소드가 서로 다른 클래스에 정의될 수 있다는 것을 보여주고 있다. 그림 3에서 'Class' 클래스의 메소드 'Method1()'은 훅 메소드인 *'Method2()'*와 *'Method3()'*에 기초한 템플릿 메소드이다.

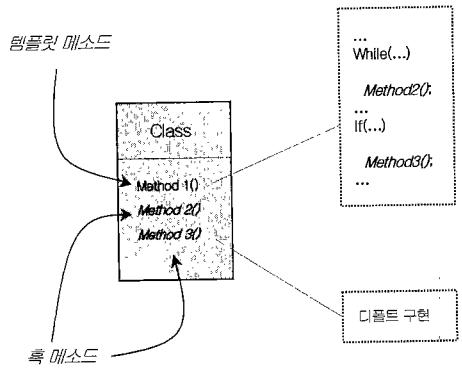


그림 3 템플릿 메소드와 훅 메소드

(4) 화이트박스(Whitebox) 프레임워크와 블랙박스(Blackbox) 프레임워크

프레임워크는 핫 스팟을 채우는 방식에 따라서 화이

트박스 프레임워크와 블랙박스 프레임워크로 구분된다.

- 화이트박스 프레임워크

클래스 상속에 근간을 둔 프레임워크를 화이트박스 프레임워크라 한다. 애플리케이션 개발자는 새로운 하위 클래스를 만들어내고 멤버함수를 오버라이딩함으로써 화이트박스 프레임워크를 구성한다.

- 블랙박스 프레임워크

블랙박스 프레임워크는 객체 컴포넌트에 기반을 두고 있다. 즉 객체를 조립하고 구성해서 새로운 기능을 얻는다. 애플리케이션 개발자는 애플리케이션 개발을 위해 이런 조합된 컴포넌트들을 사용한다.

그림 4는 프레임워크의 구성을 재사용단계별로 도식화 한 것이다. 프레임워크는 컴포넌트와 패턴의 중간단계의 재사용 기술이다. 즉 설계 관점에서 보면 컴포넌트에 비해 좀 더 추상적이고 융통성이 있으며 단순한 설계 정보보다는 더 실용적이고 직접 응용될 수 있는 코드 형태이다.

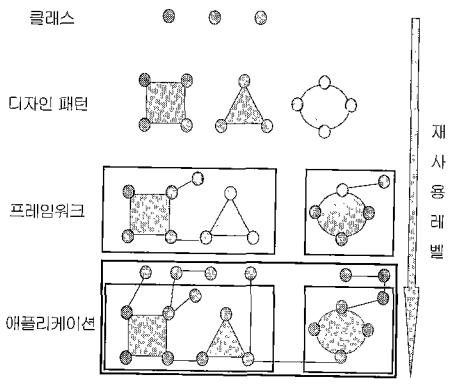


그림 4 재사용단계별 구성도

2.2 MVC 패턴을 이용한 에디터의 구조

OMT 에디터는 OMT 방법론을 기반으로 Java 언어를 이용하여 구현되었으며, 사용자로 하여금 동적 모델과 객체 모델을 구분하여 만들 수 있도록 구성되어 있다. 소스코드에서도 이러한 동적 모델과 객체 모델을 구분할 수 있는데, 객체 모델에 속하는 모든 파일이름은 'Obj'로 시작하고, 동적 모델에 속하는 파일 이름은 'Dyn'으로 시작한다.

OMT 에디터의 객체 모델과 동적 모델의 구조는 MVC 패턴을 사용하여 정확하게 설명할 수 있다. Smalltalk-80에서 제안된 MVC 구조는 객체지향 구조에서 많은 연구 및 응용되고 있는 구조로서 입력, 처리, 출력을 모델(Model), 뷰(View), 컨트롤러(Controller)의

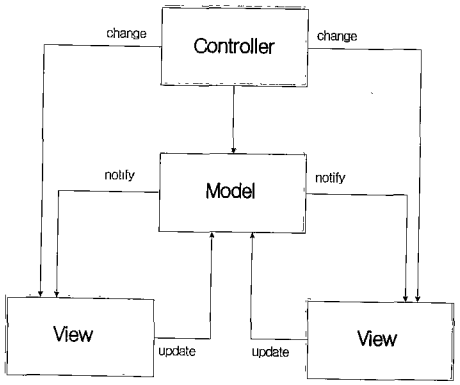


그림 5 모델, 뷰, 컨트롤러간의 관계

세 가지 구성요소로 분리하여 단순화시켰으며, 구성요소간의 결합도를 줄임으로써 유지보수와 확장 가능성을 높여준다[12]. 그림 5는 이러한 기본적인 모델, 뷰, 컨트롤러간의 관계를 보여준다.

MVC 구조에 대한 각 객체의 역할은 다음과 같다. 모델은 프로그램의 중요 기능과 데이타를 가지고 있다. 공통적인 정보를 관리하면서 외부로부터 어떤 변화가 발생하면 그 이벤트에 대한 결과를 처리하며 이 결과를 보여주기 위해 뷰에게 통보를 해주는 역할을 담당한다.

뷰는 사용자에게 처리된 결과를 디스플레이한다. 또한 같은 모델을 통해 여러 개의 디스플레이가 가능하고 뷰 자체가 여러 개의 서브 뷰를 가져서 단계별로 디스플레이 수행이 가능하다. 그러므로 뷰는 모델이 어떤 변화가 일어나고 있는지를 관찰자 입장에서 알아야 한다[12]. 컨트롤러는 애플리케이션과 사용자간의 상호작용을 담당한다. 따라서 사용자가 마우스 클릭이나 데이타 입력과 같은 이벤트를 발생시키면 이벤트에 따른 동작을 수행하기 위해 모델과 뷰에게 명령을 전달한다[13]. 이와 같이 각각의 객체가 하는 일은 명확히 분리되어 있지만 반면에 그들 각각은 서로 지속적인 관계를 유지해야 한다.

OMT 에디터는 뷰·컨트롤러 부분이 하나의 클래스에 묶여있다. 즉, 완전하게 분리되어 있지 않다. 이유는 이 도구가 JDK 1.0.2 버전에서 개발을 시작하였는데, JDK 1.0.2 버전은 이벤트 리스너(Event Listener)를 지원하지 않아 인터페이스의 모든 제어는 AWT 컴포넌트의 핸들 이벤트를 사용하였기 때문이다. 실제로 OMT 에디터의 데이타 구조를 살펴보면 모든 요소에 대하여 'ViewController' 클래스를 가지고 있다. 이러한 클래스들은 클래스명이 모두 'VCT'로 끝나고 있음을 확인할

수 있다.

본 도구에서는 기존 MVC 모델에 *Agent* 구조를 접목시킴으로써 이러한 뷰와 뷰 또는 뷰 와 컨트롤러 사이를 독립적으로 분리하는데 이용하였다. 이 외에 몇몇 상용 모델링 도구들이 존재하나, 본 프레임워크 모델러와는 직접적으로 관계가 없음은 물론 소스코드가 공개되어 있지 않아 설계 및 구현 수준에서 적용된 설계 패턴 등을 파악할 수 없기 때문에 더 이상 논하지 않는다.

3. 도구의 설계

3.1 사용자 인터페이스 구성

본 논문의 목표는 인트라넷 구축 도구에서 사용될 수 있는 프레임워크의 비주얼 인터페이스 및 브라우저를 설계 및 구현하는 것이다. 이를 위해 프레임워크 기반 모델러를 개발하여 인트라넷 구축 도구의 정보 저장소에 저장된 객체지향 분석 및 설계(이하 OOA/D) 모델을 분석한 후 이를 편집할 수 있도록 프레임워크 모델러 모듈을 구현하고자 한다. 따라서 사용자로 하여금 객체지향 기술을 이용한 고차원의 분석과 설계를 지원하고자 하는 것이다.

이 절에서는 먼저 인트라넷 구축 도구의 아키텍처 및 사용자 인터페이스 구성에 대하여 알아보고, 본 논문에서의 연구 영역인 프레임워크 모델러의 아키텍처에 대하여 자세히 논한다.

(1) 인트라넷 구축 도구

그림 6은 인트라넷 구축 도구의 모듈별 전체 아키텍처를 나타내고 있다. 인트라넷 시스템의 전체구성은 프레임워크 기반 모델러, 정보저장소 관리기, 애플리케이션 생성기, 애플리케이션 구현환경 등의 4가지 모듈로 크게 구분된다[11].

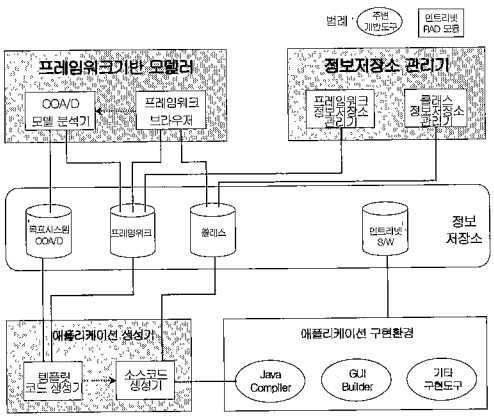


그림 6 인트라넷 전체 시스템 아키텍처

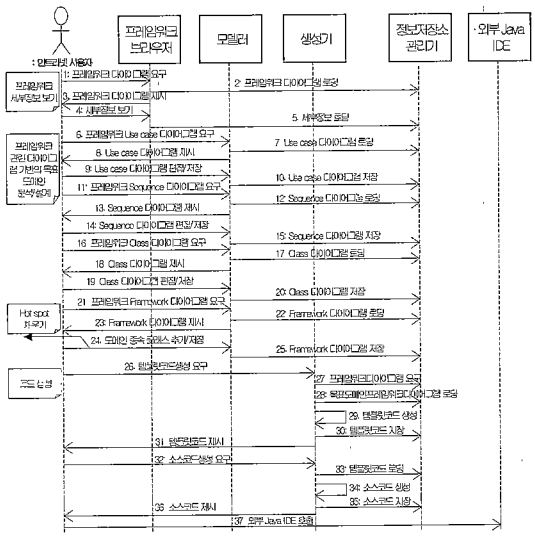


그림 7 인트라넷시스템 사용자 시나리오

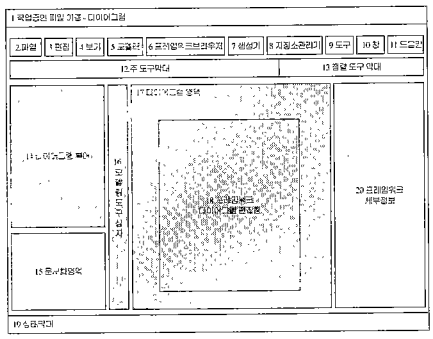


그림 8 인트라넷 구축 도구의 사용자 인터페이스

인트라넷 시스템 사용자가 본 논문을 통해 개발될 프레임워크 모델러를 통해 프레임워크 다이어그램을 요구하면 모델러는 정보저장소 관리기로부터 로딩하여 사용자는 프레임워크 다이어그램을 볼 수 있으며 세부정보를 얻을 수 있다. 즉, 핫 스팟 부분을 새로이 추가, 삭제 등의 변경을 통하여 사용자가 원하는 형태로 저장 가능하다. 또한 Use Case, Sequence, Class, Framework 다이어그램을 로딩하여 본 모델러를 통해 편집 및 저장을 할 수 있다. 이렇게 생성된 프레임워크는 애플리케이션 생성기로부터 템플릿코드를 생성하고 템플릿코드는 소스코드 생성기를 통하여 생성된 소스코드를 애플리케이션 구현환경에서 여러 구현도구와 함께 또다른 애플리케이션을 생성하는데 사용된다. 그림 7은 인트라넷 시스템의 사용자 시나리오를 Sequence 다이어그램을 통

해 나타내었다.

(2) 인트라넷 구축도구의 사용자 인터페이스

전체 도구와 관련된 도구의 화면 구성도는 그림 8과 같다. 그림 8에서 음영으로 표시된 부분이 본 논문의 연구 영역인 프레임워크 모델러 영역이다. 설계 및 구현되는 부분은 모델러 메뉴에 해당하는 5번 메뉴, 14번 다이어그램 뷰어, 16번 모델링 도구상자, 17번 다이어그램 영역, 18번 프레임워크 다이어그램 편집 창 부분이다[11].

3.2 프레임워크 모델러의 아키텍처

(1) 주요 기능

① 비주얼 인터페이스는 프레임워크 기반 모델러의 기능을 사용자에게 시각화하여 주기 위한 API이다.

② 프레임워크 기반 모델러는 개발하려는 인트라넷 애플리케이션에 대한 분석 및 설계를 수행하는 모듈로서, OOA/D 모델 분석기와 프레임워크 브라우저로 구성되어 있다.

③ 프레임워크 메타정보 구조는 정보저장소의 프레임워크 구성 요소를 브라우징하기 위한 자료 구조이다.

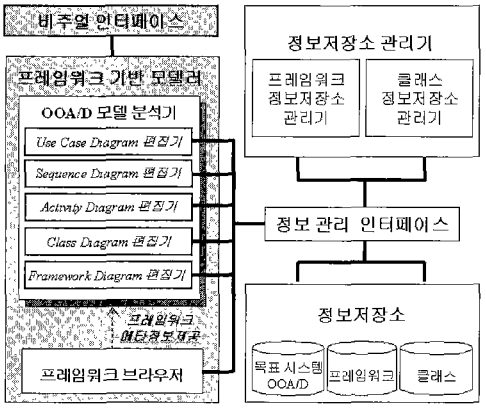


그림 9 프레임워크 모델러 구축 도구의 아키텍처

(2) 세부 기능

① 비주얼 인터페이스

- 본 도구인 인트라넷 시스템의 주메뉴 항목(모델러, 생성기, 저장소 관리기)들 중 본 논문에서 설계 및 구현할 모델러와 기타 주메뉴 항목의 서브 모듈간의 연결 및 호출을 지원하기 위한 API이다.
- 윈도 기반의 사용자 인터페이스를 제공하면 메뉴는 텍스트와 아이콘 형태로 제공한다.
- 깊이(depth)가 깊은 메뉴는 사용자로 하여금 혼동을 일으킬 수 있으므로 가능한 한 3단계 이상을 넘지 않도록 설계한다.
- 아이콘은 룩앤필 기능을 강조하여 일반적으로 사용되는 아이콘을 사용한다.

② 프레임워크 기반 모델러

- 개발하려는 인트라넷 애플리케이션에 대한 분석, 설계를 수행하는 모듈로서, OOA/D 모델 분석기와 프레임워크 브라우저로 구성한다.
- 프레임워크의 구성단위와 내부정보를 브라우징하는 기능을 제공한다.
- OOA/D 모델 분석기는 Use Case, Sequence, Class, Framework 다이어그램 편집기를 이용한 편집 기능이 가능해야 한다.

③ 프레임워크 메타정보 구조

- 정보저장소에 저장되어 있는 프레임워크의 이름, 설계 패턴, 클래스, 핫 스팟, 제어흐름, 인터페이스 등과 같은 정보를 메타정보로 제공하기 위한 자료구조이다 [11].

3.3 프레임워크 모델러에 적용한 설계 패턴

공통자료 저장소를 설계할 때 확장성과 재사용성을 고려하면서 프레임워크 모델러의 유지보수와 구조적 확장성을 위한 설계 패턴들을 식별하고 이를 이용한 분석 및 설계를 지원할 수 있도록 객체지향적으로 체계화된 모델러의 구조에 대하여 기술한다. 설계 패턴은 객체지향 소프트웨어 설계 및 구현에 있어서 클래스 구조를 설계하는 방법을 기술하는 상세한 계획이며 일반적으로 발생하는 문제들을 해결하기 위한 객체 상호작용들이다 [14].

Java 언어와 그에 관련된 API들은 이러한 객체 상호작용들을 널리 알려진 설계 패턴을 이용하여 개발되어 있다. 예를 들면 'java.awt' 패키지는 레이아웃 관리자들을 선택할 때 Strategy 패턴을 이용하고 있으며, 'java.io' 패키지와 보통 JDBC로 알려진 'java.sql' 패키지는 각각 Decorator Wrapper 패턴과 Abstract Factory 패턴을 이용하고 있다. Java 언어가 이렇게 다양한 종류의 설계 패턴을 이용하고 있는 이유는 객체지향 소프트웨어를 설계하는 것이 어려울 뿐만 아니라 재사용 가능한 객체지향 소프트웨어를 설계하는 것이 더욱 어렵기 때문에, 실제 소프트웨어 구조들로 인스턴스화 될 수 있는 구조적 추상화와 설계 패러다임들을 이용하고 있으며 이를 Java 사용자들에게 API로 패키지화하여 제공하고 있다[15]. 따라서 본 프레임워크 모델러에서 각 모듈간의 상호 운용성을 지원하기 위해서는 먼저 설계 패턴을 식별하고 이를 이용할 수 있는 기술들과의 관련성들을 연구하였다.

(1) Observer 패턴

객체지향 설계에서 시스템을 상호 작용하는 클래스들의 그룹으로 분할할 때 일관성에 대한 부작용이 발생한다. 강한 결합도를 가진 클래스들을 설계하여 일관성을 얻을 수 있을 지라도, 소프트웨어 디자이너들은 재사용성을 저하시키는 설계를 하지 않을 것이다[16]. 이러한 문제를 해결하기 위한 일반적인 방법은 Observer 패턴을 이용하는 것이다[17].

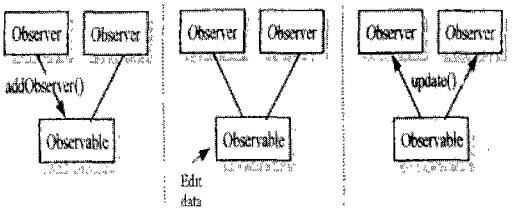


그림 10 Observer 패턴에서 Observer와 Observable과의 관계

그림 10에서 'Observable'은 임의의 데이타를 가지고 있는 객체이다. 'Observer'는 'Observable' 객체의 데이타가 변경되었는지를 감시하는 객체이다. 사용자는 'Observable' 객체와 함께 'Observer'들의 집합을 관련시킬 수 있다(그림 10(a)). 이 'Observable' 객체에서 변경이 발생하면(그림 10(b))이 'Observer'들의 집합은 이 변경사항을 통지 받는다(그림 10(c)). 일반적으로, 이 'Observable' 객체에서 변경이 일어나는 경우는 'Observer'들의 행위에 의한 경우이거나 그들의 어떤 상태 변경이 발생했을 경우이다.

(2) 뷰 사이의 메시지 전달을 위한 *Agent* 패턴

MVC 구조는 비록 객체지향 설계를 위한 구조이지만, 각 요소간의 결합도가 높기 때문에 설계 패턴을 이용하여 결합도를 낮게 할 수 있다. 이 절에서는 본 도구에 적용한 *Agent* 패턴의 적용 기법을 보이고자 한다.

본 논문에서 설계한 MVC 기반의 프레임워크 다이어그램 모델링을 지원하는 본 프레임워크 모델러는 사용자의 편리한 설계 및 구현을 위한 비주얼 모델링 도구로서 여러 뷰를 사용하고 있다. 각각의 뷰들은 메시지를 전달함으로써 다른 뷰들과 통신한다. *Agent* 패턴은 이렇게 통신하는 뷰들 사이의 구조를 정형화시키기 위해 설계한 패턴이다[18].

MVC에서 적용된 바와 같이 관찰자 패턴은 서로 다른 요소간의 상호작용을 위해 변경되는 상태를 초기화하고 전달하기 위해 적절하게 사용되고 있다. 이러한 Observer 패턴을 본 논문에 적용하여 본 결과 자료 구조를 전달하는데 유용하였다. 그러나 MVC 구조의 여러 뷰가 존재할 경우, 뷰와 뷰사이에 메시지를 전달할 수 있는 방법은 현재로서는 명확히 구분되어있지 않다. 본 도구를 설계 및 구현하는데 있어 이러한 뷰와 뷰사이의 메시지 전달 제어를 자체 설계 패턴인 *Agent* 패턴을 사용하여 여러 개의 뷰 사이에 발생하는 메시지를 전달함으로써 뷰와 뷰사이의 관계가 보다 덜 종속적이면서 도구전체가 일관된 개발을 할 수 있도록 하였다. 그림 11은 자체 설계 패턴인 *Agent* 패턴의 구조를 나타내고 있다.

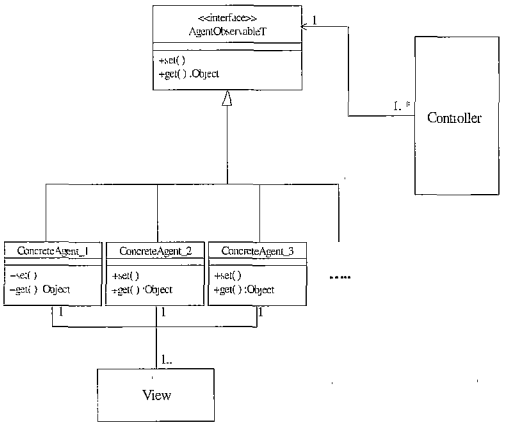


그림 11 Agent 패턴의 구조

Agent 패턴의 'AgentObservableT' 인터페이스는 그림 12와 같이 set() 메소드와 get() 메소드를 제공한다. 'ConcreteAgent' 클래스는 'AgentObservableT' 인터페이스의 메소드를 구현함으로써 메시지를 초기화하고 메시지를 요청하는 뷰에 지정된 메시지를 전달한다. 또한 인터페이스 패턴을 따르고 있기 때문에 이를 구현하여 기능확장을 할 수 있도록 하였다.

```
package jfree.data;

public interface AgentObservableT {
    public void set(Object source, String command,
Object arg);
    public Object get(Object source, String command,
Object arg);
```

그림 12 'AgentObservableT' 인터페이스

위와 같이 적용된 MVC 구조는 그림 13과 같은 구조로 되어 있다. 여러 개의 뷰와 컨트롤러 사이의 직접적인 통신을 할 경우에 컨트롤러로 하여금 서로 다른 뷰에게 메시지를 전달하는 과정을 나타내는 그림이다.

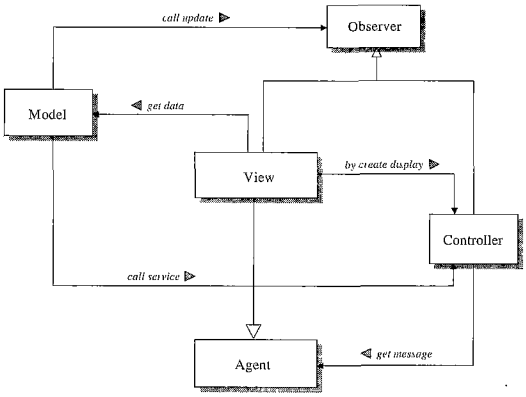


그림 13 *Agent* 패턴을 적용한 MVC 구조

위 구조에서 컨트롤러 부분의 이벤트 모델은 위임형 이벤트 모델(delegation event model)을 적용하였다. 위임형 이벤트 모델은 Java AWT 1.1 버전부터 제공하는 Java 언어의 특성이기 때문에 바로 적용가능하다. 위임형 이벤트 모델은 이벤트 객체를 이벤트 발생 객체가 생성하여 이를 이벤트 처리 객체(리스너)에게 위임하는 방식의 이벤트 모델이다. 위임형 이벤트 모델을 사용할 경우에 이벤트 객체의 인스턴스를 이용하여 기능을 추가하는 클래스를 작성함으로써 클래스의 독립적인 기능 확장과 유지보수를 용이하게 할 수 있다[18].

4. 프레임워크 모델러의 구현

본 도구는 두 가지 소프트웨어공학 관점에서의 개발을 지원한다. 첫 번째는 컴포넌트에 기반 하여 프레임워크 중심적인 설계 및 구현을 가능하게 하고, 두 번째는 애플리케이션을 개발하는데 있어서 상호 작용성을 제어하는 관점에서의 설계 및 구현을 지원한다. 이를 위해 시각적인 사용자 인터페이스(GUI)를 사용하여 모델링을 할 수 있는 다이어그램을 구현한다. 본 장에서는 사용자 위주의 개발을 위한 구현 내용을 살펴보고자 한다.

4.1 도구의 모듈별 세부 구현

본 논문에서 설계 및 구현한 프레임워크 기반의 애플리케이션을 개발하기 위한 모델러의 기본 구조를 모듈별로 설계 및 구현하였다. 각각의 모듈별 구조를 살펴보면 다음과 같다.

프레임워크 모델러가 시작될 때 처음으로 실행되는 'StartForm' 클래스는 'Main' 클래스를 호출함으로써 그림 14와 같이 메모리에 각 모듈을 초기화시키고 뷰 사이에 메시지를 전달할 구조를 형성하도록 하였다.

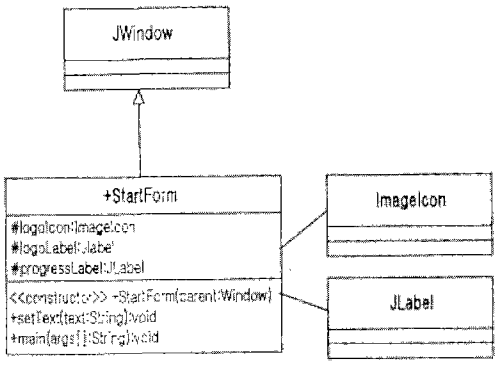


그림 14 'StartForm' 클래스

그리고 'StartForm' 클래스가 실행된 다음 그림 15의 'MainForm' 클래스는 컴포넌트를 초기화하며 각각의 기능 컴포넌트들을 생성하도록 하였다.

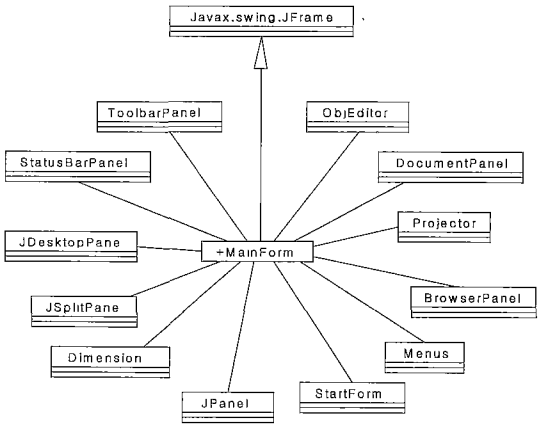


그림 15 'MainForm' 클래스

본 프레임워크 모델러의 동작 구성을 설계할 때 'MainForm' 생성과 진행과정을 'StartForm'에 출력하도록 클래스를 설계 및 구현하였다. 또한, 윈도가 화면상에 표현될 때 룩앤필 메소드를 호출하여 각 시스템 플랫폼(운영체제)에 맞게 적용하도록 하고, 초기 화면의 타이틀에는 현재 본 도구의 버전을 나타내도록 설계하였다. 본 도구가 화면상에 나타내어 질 때 'Menus' 클

래스를 호출하도록 하였다. 그림 16과 같이 메뉴를 나타낼 때는 'Menus' 클래스의 'Menus()' 생성자에서 'initializeControls()' 메소드와 'registerListener()' 메소드를 호출하여 각종 메뉴 및 메뉴 아이템을 생성하도록 하였다.

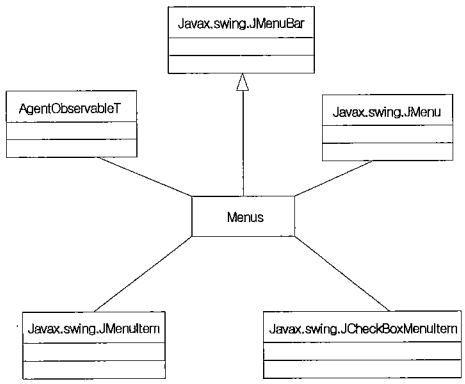


그림 16 'Menus'의 클래스 다이어그램

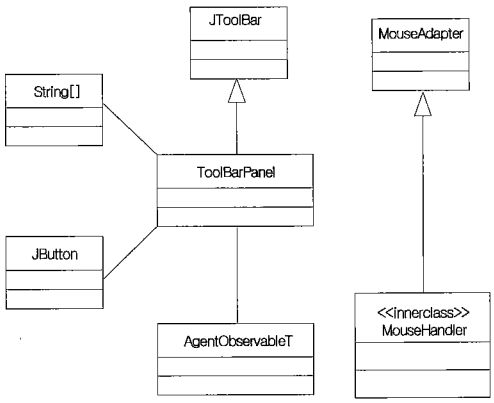


그림 17 'ToolBarPanel' 클래스 다이어그램

이러한 메뉴에 해당하는 클래스의 내부는 'register Listener()' 메소드에서 각 메뉴 아이템들을 'action Listener'에 등록하여 'AddWindow', 'New', 'Open', 'Close', 'RecentFile' 등의 기능을 가능하게 하고 'Menus' 클래스의 수행을 완료한다. 이러한 메뉴를 화면에 등록시킬 때는 배치 관리자(Layout Manager)를 사용하게 되는데, 본 도구를 설계할 때 'ContentPane()' 메소드를 'BorderLayout()' 메소드로 지정하여 위치 인자값을 'North'로 설정하여 메뉴에 추가하도록 하였다. 그리고 'CenterPane' 클래스의 중앙에 'CenterPanel'을 지정하고 'CenterPanel' 또한 'BorderLayout'으로 지정하도록 하였다. 그리고 'BrowserPanel'을 구현 하기 위하여 'JPanel'의 객체인 'topPanel'을 생성하여 'BorderLayout'으로 선언하고 프레임 크기 및 'Border'를 'EmptyBorder()', 'BevelBorder()' 등의 메소드를 이용하여 설정한다. 그런 다음 그림 18과 같이 'Browser Panel'의 객체를 생성하여 'topPanel'의 'Center'에 지정한다.

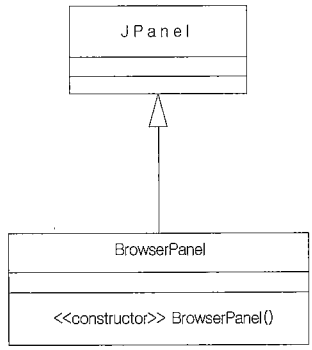


그림 18 'BrowserPanel'의 클래스 다이어그램

다음으로 'BottomPannel'의 객체를 생성하여 'Border Layout'으로 지정하고 'Border'를 'EmptyBorder()' 메소드와 'BevelBorder()' 메소드를 이용하여 'Border'를 지정하였다. 그림 19는 'DocumentPanel()' 메소드의 객체를 생성하여 'bottomPanel'의 'Center'에 위치를 지정하도록 설계한 다이어그램을 나타내고 있다.

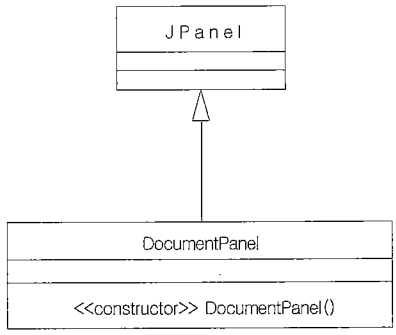


그림 19 'DocumentPanel' 클래스 다이어그램

그림 19와 같이 'BrowserPanel'과 'DocumentPanel'을 구현한 후 'JSplitPane'의 객체를 생성하여 'topPanel'과 'bottomPanel'의 수직 분계선을 나타내는 'leftSplitPane'

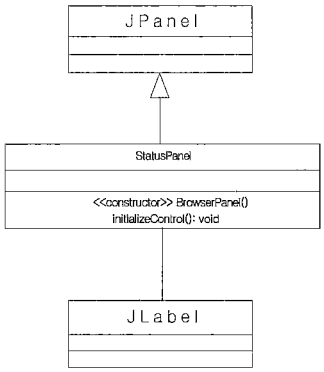


그림 20 'StatusBarPanel'의 클래스 다이어그램

을 만든 후, 'OneTouchExpandable()' 메소드를 'true'로 설정하여 마우스로 한 번 클릭할 때마다 확장되고 축소되는 기능을 설정하였다. 그림 20은 'JSplitPane'의 객체를 생성하여 'topPanel'과 'bottomPanel'의 수직 분계선을 나타내는 'rightDesktop Pane' 객체를 생성하고, 'SplitPanel'의 객체를 생성하여 'leftSplitPanel'과 'rightDesktopPanel' 사이에 수평 분계선을 표현하기 위한 클래스 다이어그램을 나타내고 있다. 또한, 'Status Bar' 객체를 생성하여 'centerPanel'의 'South'로 위치가 설정되도록 구현하였다.

각 모듈에 대한 클래스 다이어그램의 시작은 'initializeForms()' 메소드가 호출되면서 'Projector' 클래스의 객체 생성과 'initializeControls()' 메소드를 호출함으로써 각각의 모듈이 초기화 및 가시화되고 이벤트가 발생될 수 있도록 구현하였다.

4.2 프레임워크 모델러의 모듈별 화면 아키텍처

본 논문에서 주 구현 내용이면서 프레임워크 기반의 모델링이 가능한 에디터인 모델러는 그림 21과 같다. 시

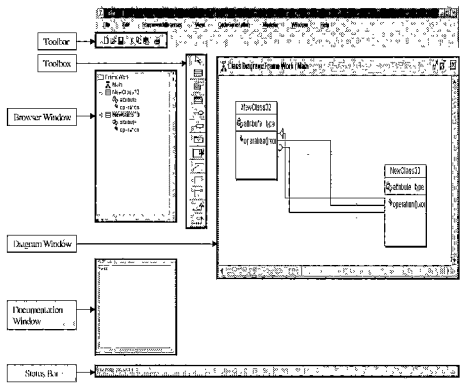


그림 21 프레임워크 모델러의 모듈별 화면 아키텍처

각적인 개발을 지원하기 위한 메뉴들과 툴바 등은 사용자로 하여금 사용하기 쉽게 구성하는데 중점을 두었다. 위와 같은 도구를 이용하여 사용자가 애플리케이션을 개발하도록 하는데 있어서 다음과 같은 이점을 제공한다.

- 모델의 모든 요소들에 대한 계층적인 뷰를 제공
- 모델의 구성 요소에 대하여 드래그앤드롭 방식의 구현을 지원함으로써 시각적인 개발 지원
- 도구의 각 뷰간의 메시지를 통한 내부 구조로서 모델링 영역의 요소의 변화에 대한 추적 기능

4.3 실행 결과

먼저 본 프레임워크 모델러를 실행하면 컴포넌트의 초기화(initialization) 및 적재(loading) 작업을 수행하고 각각의 기능 컴포넌트들을 생성한다.

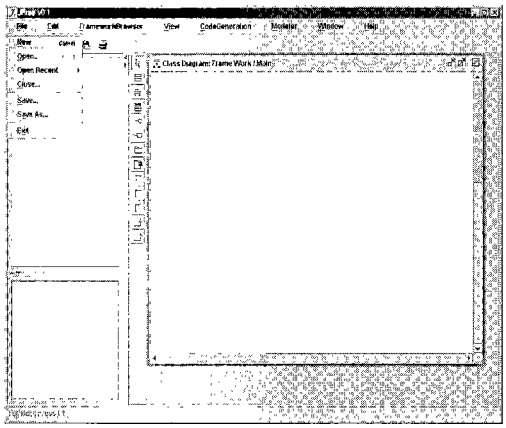


그림 22 프레임워크 모델러의 초기 인터페이스

초기화 작업이 정상적으로 수행되면 그림 22와 같이 새로운 프레임워크 다이어그램을 작성하기 위한 템플릿 기반의 이름이 없는 빈 모델링 영역이 나타나고 사용자는 'File' 메뉴의 'New' 메뉴항목을 선택함으로써 시작하게 된다.

4.4 테스트

(1) 목적 및 환경

구현된 본 모델러가 프레임워크 다이어그램 편집창에서 정확히 편집작업을 할 수 있는지, 또한 소스코드가 정확히 생성이 되는지를 확인하기 위해 견본 프레임워크 다이어그램을 작성해 태스트하고자 한다. 테스트는 Pentium Ⅲ 600MHz 시스템으로 Windows 2000 환경에서 실시하였다.

(2) 시험대상

검증되어야 할 요구사항으로는 정보저장소에 저장된

OOA/D 모델을 분석한 후 프레임워크 다이어그램 편집기 모듈을 구현하여 사용자로 하여금 OOA/D를 할 수 있도록 지원하는지의 여부이다.

(3) 시험 전략과 사용될 기법 및 도구

구현된 프레임워크 모델러를 이용하여 프레임워크 다이어그램을 작성하는 예를 들어보기로 한다. 작성하고자 하는 프레임워크 다이어그램은 사용자관리 프레임워크 다이어그램이다. 먼저, 툴 상자에서 클래스모양의 버튼을 선택하여 클래스를 정의한 상태에서 마우스 오른쪽 버튼을 클릭하여 'Open Specification' 메뉴를 선택한다. 'General' 탭에서는 클래스의 이름, 스테레오 타입, 클래스의 형, 가시성 등을 정의한다. 같은 방식으로 'Attribute' 탭에서는 해당 클래스의 속성을 'Operation' 탭에서는 해당 클래스의 오퍼레이션을 정의할 수 있다. 아래 그림 23, 24는 'General' 탭과 'Attribute' 탭을 선택하였을 때의 예이다.

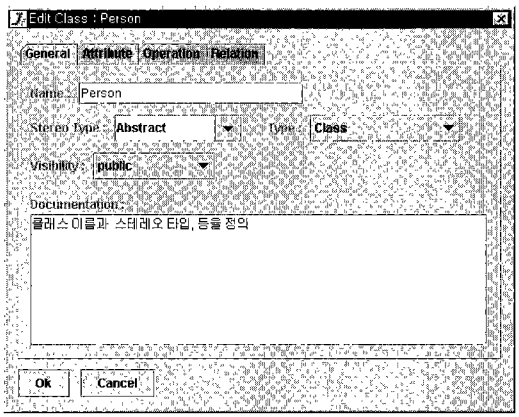


그림 23 'General' 탭을 선택한 경우

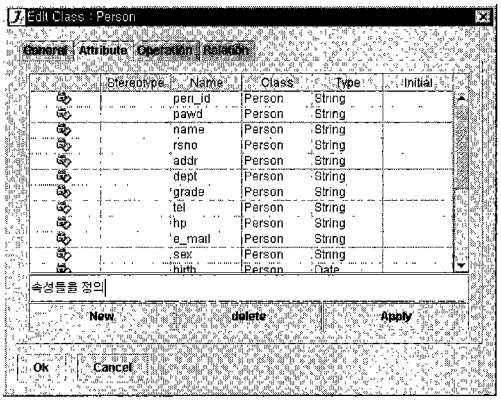


그림 24 'Attribute' 탭을 선택한 경우

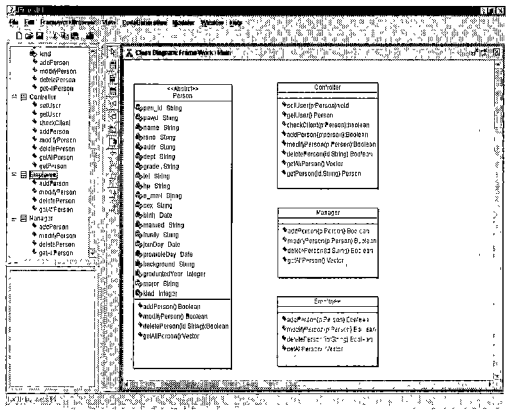


그림 25 'Person', 'Controller', 'Manager', 'Employee' 클래스의 설계

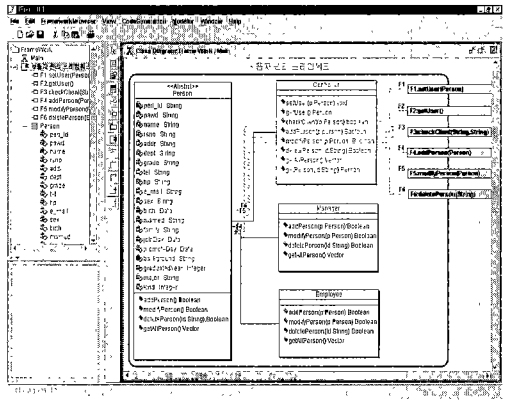


그림 26 완성된 프레임워크 다이어그램의 예

그림 25, 26의 예는 'Person' 클래스의 속성과 오퍼레이션을 정의하는 예로서 위와 같은 방법으로 'Control' 클래스, 'Manager' 클래스, 'Employee' 클래스 등을 정의한다.

그림 25는 'Person' 클래스, 'Controller' 클래스, 'Manager' 클래스, 'Employee' 클래스를 설계한 모습을 보여주고 있다. 이들 4개의 클래스를 프레임워크를 나타내는 아이콘을 선택하여 프레임워크 도메인으로 정의하고 프레임워크 도메인 우측상단에 인터페이스를 정의한다. 인터페이스와 클래스들간의 관계를 제어흐름 선을 나타내는 아이콘을 선택하여 이들간의 관계를 정의해준다. 이때 인터페이스와 클래스들간의 관계를 나타내는 제어흐름 선이 복잡할 경우 겹쳐 보이는 등, 제어흐름 선을 구분하기가 모호해질 수 있다. 본 도구에서는 이러한 단점을 없애기 위해 각각의 제어흐름 선을 더블클릭

하여 개발자가 원하는 색으로 제어흐름 선을 구분할 수 있도록 하였다. 이렇게 해서 완성된 하나의 프레임워크 다이어그램은 그림 26에서 보여주고 있다. 다이어그램 뷰어 창에 나타나있는 각각의 인터페이스 및 클래스들에 대한 소스코드 정보가 자동으로 생성되어 마우스 오른쪽 버튼을 클릭하여 'View Source' 메뉴를 선택하면 그림 27과 같이 해당 클래스의 소스코드를 볼 수 있게 되어있다. 한편, 그림 26에서 보여주고 있는 프레임워크 모델링 기법은 참고문헌 [3]에서 연구된 결과를 사용하였다.

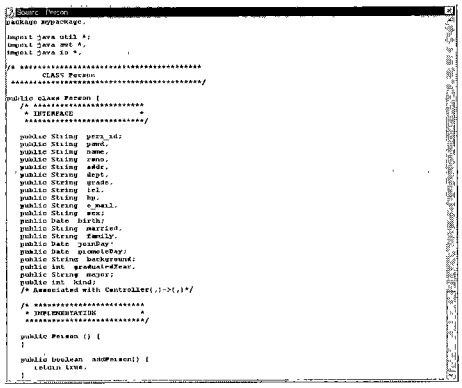


그림 27 'Person' 추상클래스의 소스

4.5 도구의 비교 분석

본 논문에서 설계 및 구현한 프레임워크 모델러와 직접적으로 비교할 수 있는 도구는 없지만, 본 도구의 설계시 기반이 된 OMT 에디터와 간접적인 비교를 함으로써 구현도구의 우위성을 보이고자 한다. 도구의 비교분석은 또한 최종 테스트의 대안으로서 고려되었다. 표 1은 기존 OMT 에디터와 본 논문에서 설계 및 구현한 프레임워크 모델러를 비교 분석한 결과이다. 표 1에서도 알 수 있듯이 OMT 에디터는 객체지향 구조에서 많이 연구 및 응용되고 있는 MVC 패턴을 사용하여 유지보수와 확장을 용이하게 한다는 의도로 개발되었지만 개발당시에 JDK의 초기 버전으로 개발되어 완전하게 객체지향적으로 되었다고는 볼 수 없다.

인터페이스의 가장 큰 특징으로는 OMT 에디터는 동적모델과 객체모델링을 따로 작성할 수 있도록 메뉴버튼이 각각 존재한다는 것에 반해 본 프레임워크 모델러는 다이어그램 뷰어 영역, 편집 창, 문서화 영역, 모델링 도구상자로 구분되어 있어 그 역할이 명확히 구분되어 있다는 것을 한눈에 알 수 있으며, 한번의 실행으로 여러 작업 창을 이용해 다수의 작업이 가능하다는 특징이 있다. 결론적으로 본 논문에서 설계 및 구현한 프레임워크 모델러는 표 1에서 보는 바와 같이 OMT 에디터의 여러 단점을 보완하여 보다 더 체계적으로 연구되었음

표 1 OMT 에디터와 본 프레임워크 모델러와의 비교 분석

비교항목 \ 비교도구		기존 OMT 에디터	본 프레임워크 모델러
적용된 설계패턴		• MVC 패턴 • Observer 패턴 • Visitor 패턴 외 기타	• 자체 설계 패턴인 *Agent* 패턴을 기존 MVC 구조에 적용 • Observer 패턴 외 기타
인터페이스	주요 특성	• 초기화면에 동적모델과 객체모델링을 할 수 있는 메뉴버튼이 존재	• 크게 다이어그램 뷰어영역, 편집창 영역, 문서화 영역, 모델링 도구상자 영역 등으로 나뉘어 짐
	특징	• 객체모델과 동적모델을 별도로 구분하여 별도의 작업창이 여러 개가 존재할 수 있기 때문에 작업 시 많은 번거로움 초래	• 다이어그램 편집창에서 객체모델과 동적모델링 작업이 가능하고 한번 실행으로 동시에 여러 개의 작업창을 띄워서 다수의 작업 가능
		• 다이어그램 뷰어와 같은 기능이 없어 사용자가 작업하고 있는 파일의 정보를 손쉽게 알 수 없음	• 윈도우 탐색기와 같은 다이어그램 뷰어가 있어 사용자가 작업하고 있는 정보를 계층화하여 트리 형태로 디스플레이함으로써 사용자의 편리성을 도모함
		• 문서화영역이 없어 각 객체(클래스)에 원하는 주석을 지원하지 않음	• 별도의 문서화 영역이 있어 다이어그램 요소에 대한 부가적인 설명을 할 수 있도록 지원함
주요기능		• 객체모델링에는 클래스 다이어그램과 동적 모델링에는 상태 다이어그램만 지원 • 모델링한 다이어그램 토대로 스켈리톤 수준의 Java 소스코드 생성 가능 • 다이어그램 편집창에서 직접 편집기능을 하지 못하고 별도의 창에서 가능 • OMT 방법론을 기반으로 구현되어 UML 표기법을 완전히 지원하지 않음	• 클래스 다이어그램, 프레임워크 다이어그램 등 UML에서 제공하는 주요 다이어그램 지원 • 모델링한 다이어그램을 토대로하여 스켈리톤 수준의 Java 소스코드 생성 가능 • 편집창에서 직접 클래스의 메소드와 속성의 편집 가능 • UML 표기법을 기반으로 구현됨

을 알 수 있다.

5. 결론 및 향후 연구과제

본 논문에서는 인트라넷 구축 도구에서 사용될 수 있는 프레임워크에 기반한 비주얼 인터페이스 및 브라우저를 설계 및 구현하였다. 모델러의 분석 및 설계에서 소스코드의 생성을 위한 각각의 기능 컴포넌트들과 이들간의 상호 연동을 지원하는 구조는 Java 언어의 특성을 반영하는 독창적 구조로 설계되어 있다. 그리고 개발 시스템을 프레임워크 단위로 다루기 위한 프레임워크 아이콘과 기본 컨트롤러를 제공한다. 이러한 프레임워크를 이용한 시스템 개발은 사용자 프로그램의 원시 코드와 각 모델의 요소들을 프로젝트 단위로 조직하여 관리한다. 프레임워크 관리 모델은 본 프레임워크 모델러의 사용자 프레임워크를 효과적으로 조직화하여 관리하며 프레임워크내의 클래스들을 재사용 가능하도록 지원하는 특징을 가지고 있다.

관련연구로 J. Rumbaugh의 OMT 방법론을 기반으로 한 OMT 에디터 내에 이용된 관련 패턴들을 대부분 흡수함으로써 MVC 구조와 이에 적용된 관찰자 패턴에 대하여 연구하고 자료구조를 전달하는데 이를 적용하였다. 그리고 뷰와 다른 뷰 사이의 메시지 전달 과정을 구조적으로 체계화하기 위해 MVC 구조에 자체 설계 패턴인 *Agent* 패턴을 적용함으로써 인터넷/인트라넷 개발 환경에서 소프트웨어를 개발하는데 시각적인 분석 및 설계할 수 있는 모델러의 구조를 *Agent* 패턴을 이용하여 구현하였다.

텍스트 기반 프로그래밍을 지원하는 Java 원시 코드 편집기는 Java 원시 코드를 생성하는 모듈을 이용하여 모델링 영역에서 추출된 기본 코드로 사용자가 작성하고 수정한 컴포넌트의 정보를 Java 코드로 변환하여 그래픽한 문자로 디스플레이 한다. 이러한 코드 생성 과정은 Java 기반의 모델링을 위한 개발 도구로서 핵심이 되는 기능이다. 현재 설계 및 구현한 모델러의 구조는 앞으로의 객체지향 소프트웨어 생성을 위한 모델링 도구를 개발할 때 재사용 가능하여 요구사항이 변경될 때 도구 자체를 다양하고 쉽게 변경할 수 있는 사용자 중심의 프레임워크 모델러가 될 것으로 기대된다.

향후 연구과제로는 본 도구를 설계 및 구현하는데 이용되었던 패턴 외에도 OMT 에디터의 자료저장소와 관련된 방문자 패턴(Visitor Pattern)외에 OMT 에디터에서 사용되지 않은 새로운 패턴을 이용하여 도구의 구조를 더욱 체계화하는 것이 필요하다. 또한 *Agent* 패턴의 시험적 검증 단계를 완성하고 사용자 중심의 인터페이스 구축을 위해 모델링 레이블 다이얼로그박스를 추가하는 것이 필요하다.

참 고 문 헌

[1] Martin Fowler, UML Distilled, Second Edition: A Brief Guide to the Standard Object Modeling Language, Addison-Wesley Pub. Co., 1999.

[2] James Martin, Rapid Application Development, Prentice Hall, 1991.

[3] 조은숙, 김수동, 류성열, "UML 기반의 객체지향 프레임워크 모델링 기법", 한국정보과학회논문지, 제26권 4호, pp. 533~545, 1999.

[4] Booch, Grady, Objected-Oriented Design with Application, Benjamin-Cummings, 1991.

[5] Jacobson, Object-Oriented Software Engineering : A Use Case Driven Approach, Addison- Wesley Pub. Co., 1992.

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

[7] Hans-Erik Eriksson, Magnus Penker, UML Toolkit, John Wiley & Sons, Inc., 1998.

[8] 김형국, 주예환, 이민규, 김명호, "모델링을 고려한 RAD 도구의 설계", 한국정보과학회 가을 학술발표 논문집, 제25권 2호, pp. 517~519, 1998.

[9] Martin Fowler, Object-Oriented Methods UML Edition, Addison-Wesley Pub. Co., 1999.

[10] 배명남, 김훈희, 양재동, 최완, "객체 모델링을 지원하는 의미 기반 설계 도구의 개발", 정보과학회논문지(C), 제3권 제3호, pp. 248~261, 1997.

[11] 시스템공학연구소 S/W공학연구부 개방형 S/W연구실, '인트라넷 구축도구 개발기술' 과제 기술현황분석서, 1998.

[12] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, Design Patterns, Addison-Wesley Pub. Co., 1994.

[13] Paul R., Reed Jr., Developing Applications with Visual BASIC and UML, Addison-Wesley Pub. Co., 1999.

[14] D'Souza, Desmond Francis, Wills, and Alan Cameron, Objects, Components, and Frameworks with UML, Addison-Wesley Pub. Co., 1998.

[15] Mark Grand, Patterns in Java, A Catalog of Reusable Design Patterns Illustrated with UML, John Wiley & Sons, 1998.

[16] Douglass, Bruce Powell, Real-Time UML, Developing Efficient Objects for Embedded Systems, Addison-Wesley Pub. Co., 1997.

[17] 유철중, "Java 통합 개발 환경에서 기능 컴포넌트들의 상호연동 기법", 정보처리학회 논문지, 제5권 제11호, pp. 2862~2873, 1998.

[18] 김영진, 김대헌, 유철중, 장옥배, 정호택, 양영종, 이상덕, "객체지향 분석 및 설계를 지원하는 모델러의 구

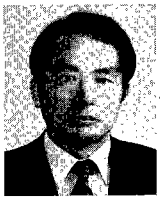
조", 한국정보과학회 추계학술발표 논문집, 제26권 2호, pp. 475~477, 1999.

이 창 목

1999년 2월 호원대학교 전자계산학과(학사). 1999년 3월 ~ 현재 전북대학교 전산통계학과 석사과정. 관심분야는 객체지향개발방법론, 객체지향 프레임워크, 컴포넌트개발 디자인 패턴 등임

유 철 중

1982년 2월 전북대학교 전산통계학과 졸업(이학사). 1985년 8월 전남대학교 대학원 계산통계학과 졸업(이학석사). 1994년 8월 전북대학교 대학원 전자계산학과 졸업(이학박사). 1982년 9월 ~ 1985년 3월 전북대학교 전자계산소 조교. 1985년 4월 ~ 1996년 12월 전주기전여자대학 전자계산과 부교수. 1997년 1월 ~ 현재 전북대학교 자연과학대학 컴퓨터과학과 조교수. 관심분야는 소프트웨어공학, 에이전트공학, 컴포넌트기술, 분산객체기술, 지리정보시스템, 멀티미디어, 인지과학 등임.



장 옥 배

1966년 고려대학교 수학과 졸업. 1980년 죠지아 주립대, 오하이오 주립대 박사과정 수료. 1987년 산타바바라대학교 졸업(Ph.D). 1990년 ~ 1991년 영국에딘버러대학교 객원교수. 1980년 4월부터 현재 전북대학교 컴퓨터과학과 교수. 관심분야는 소프트웨어공학, 전산교육, 수치해석, 인공지능 등임.

이 상 덕

1980년 부산대학교 대학원 경영학과 졸업(석사). 1978년 ~ 현재 한국전자통신연구원 책임연구원