객체 지향 소프트웨어 개발 환경을 위한 지역 버전관리자

최 동 운† · 김 수 용†† · 송 행 숙†††

요 약

분산 시스템에서 객체 지향 소프트웨어를 개발하는 과정에서 다양한 버전들이 발생할 수 있다. 이런 다양한 버전들을 효율적으로 관리하는 방법론에 대한 연구가 필요하다. 본 논문에서는 분산 환경에서 객체 지향 소프트웨어를 개발하는 과정에서 발생하는 버전들을 관리하기 위해서 5차원 객체 공간을 기반으로 하는 버전 모델을 제안하고 있는데, 이는 본 연구팀에서 제안하였던 FONASSE 데이터 모델을 확장한 것이다. 또한 버전들을 효율적으로 제어하기 위해서 새로운 형태의 버전 번호를 부여하는 방법을 개발하였다. 그리고, 소프트웨어 개발자들이 버전들을 효과적으로 관리할 수 있도록 지역 버전관리자를 설계하여 Solaris 2.5 환경에서 Tcl/TK와 C++를 이용하여 구현하였다.

The Local Version Manager for Object Oriented Software Development Environment

Dong-Oun Choi† · Soo-Yong Kim†† · Hang-Sook Song†††

ABSTRACT

Various versions can be generated in course of development of object-oriented software in a distributed system. Research methods are required for the efficient management of these various versions. For this reason, this paper proposes a model of version which is based on five dimensional object space; this is extended from FONASSE data model which we have suggested. We developed a method of version numbering for the effective manage of the versions. To make software developers manage versions efficiently, we design and implement local version manager by using TCl/TK and C++ in Solaris 2.5.

1. 서 론

소프트웨어 개발 방법론은 잘 정의된 기법들과 표현법들을 이용하여 소프트웨어를 조직적으로 생산하는 과정으로, 이러한 소프트웨어 개발 환경은 소프트웨어 개발의 각 단계를 지원하는 자동화된 도구들을 통해 구축된다[5, 6]. 그러나, 서로 독립된 형태의 도구들 간에는 상호간의 연관성이 적어 연속적이고 통합적인 작업을 하기가 어렵다. 따라서, 각 도구들 간의 유기적인 관계를 통해 이들을 통합화된 환경으로 구축하기 위해 통합 정보저장소에 대한 연구가 진행중이다[3, 10, 11, 12, 17, 18]. 즉, 각각의 자동화된 도구들에 의해 생성되어 중복되고 일치되지 않는 정보들을 하나의 공유된 정보 저장소를 통해 통합적으로 관리하려는 것이다. 소프트웨어 개발 공정의 자동화, 소프트웨어 모듈의 재

† 정 회 원 : 서남대학교 전산정보학과 교수
†† 준 회 원 : 서남대학교 대학원 전산학과
††† 정 회 원 : 한일장신대학교 전자통신학부 교수

논문접수 : 1998년 4월 20일, 심사완료 : 1998년 10월 15일

사용을 지원하여 개발 생산성을 높이는 객체 지향 방법론이 널리 쓰이고 있다. 이를 지원하는 많은 객체 지향 CASE 도구들이 사용되고 있지만 이들은 모두 지리적으로 인접한 지역에서 독립적인 소프트웨어 시스템을 개발하는 도구만을 제공하고 있다. 그런데, 최근에는 분산 환경에서 협동적인 소프트웨어 개발을 위해 많은 연구가 진행되고 있다[7]. 그러나 분산환경에서 객체 지향 소프트웨어를 개발하는 과정에서 발생하는 버전 관리 기법에 대한 연구가 아직 미비한 상태이다.

객체 지향 분산 소프트웨어 개발 환경에서의 서로 다른 장소, 다른 시간에 하나의 프로젝트를 수행하는 과정에서 다양한 버전들이 발생하게 된다. 이렇게 발생하는 다양한 버전들은 정보 저장소에 저장되는데, 이들을 효율적으로 관리할 수 있는 새로운 버전 관리 방법에 관한 연구가 필요하다. 따라서 본 논문에서는 객체 지향 소프트웨어 개발 방법론인 OMT(Object Modeling Technique)[2, 9]를 기반으로 한 분산 소프트웨어 개발 환경에서 발생하는 다양한 버전들을 관리하기 위해서 본 연구팀에서 제안하였던 FONASSE (FOur imensional NAvigation Spaceship for Software Engiering) 데이터 모델[13]을 확장한 버전 모델을 제안하고, 지역 버전관리자에서 버전들이 발생하였을 때 이들에게 버전 번호를 부여하는 기법을 개발하였으며, 버전들을 효율적으로 관리하기 위한 지역 버전관리자 인터페이스를 설계하고 구현한 내용이다. 또한 버전 관리자는 개발자가 재사용 가능 소프트웨어 부품을 검색하는 데에도 유용하게 사용될 수 있다. 제안한 버전관리자는 다음과 같은 특징을 가진다. 첫째, 분산 환경에서 객체 지향 소프트웨어를 개발하는 과정에서 다양한 버전들을 객체 지향 시스템이 가지는 시멘틱을 손실하지 않도록 객체 지향 데이터베이스 시스템에 관리한다. 둘째, 기존의 버전 제어 시스템들은 대부분 프로그램 코드 수준의 버전을 관리하는 것에 비해 제안한 버전 관리자는 객체 지향 소프트웨어를 개발하는 전 과정 즉, 요구 문장, 분석, 설계, 프로그램 코드 등에서 발생하는 모든 객체 버전들을 관리할 수 있다는 장점을 가지고 있다.

본 논문의 2장에서는 기존의 버전 제어에 대한 특징을 살펴보고, 3장에서는 분산 객체 지향 소프트웨어 개발 환경에서의 버전 제어 모델을 제안하며, 버전 번호를 부여하는 기법을 기술하고, 4장에서는 버전관리자 인터페이스를 통해서 버전 객체 검색의 예를 보인다. 마지막 5장에서는 결론 및 앞으로의 연구과제를 논의한다.

2. 관련 연구

2.1 객체 지향 소프트웨어 개발 환경

객체 모델링 기법(OMT)은 제너럴 일렉트릭(GE) 연구소에서 개발된 새로운 객체 지향 소프트웨어 개발 방법론이다[2, 9]. 이 방법은 시스템을 모델링하고, 설계하고, 구현하는 소프트웨어 개발 주기의 전 과정에 객체 지향 개념을 적용한다. 기본적으로 객체 모델, 동적 모델, 기능 모델이라는 세 가지 모델을 각각의 표기법을 이용하여 개발해 나감으로써 실세계를 보다 정확하게 모델링하고 유지보수가 용이하며 재사용성이 높은 소프트웨어를 생산하는 방법론이다.

이 방법론은 문제를 표현하는 문장 형식인 요구 명세로부터 시작한다. 그런데 대개 요구 문장은 불완전하고 규정에 맞지 않기 때문에, 분석 단계에서 요구 사항의 모호함과 불일치성을 해결하여 실세계에 대한 모델링을 명확히 해주어야 한다. 이를 위해 분석 단계에서는 세 개의 모델이 만들어지는데 객체 모델(Object Model)은 실세계를 나타내는 개체의 정적 구조를 나타내고, 동적 모델(Dynamic Model)은 일의 진행 순서를 나타내며, 기능 모델(Functional Model)은 자료의 변환을 나타낸다. 이러한 분석을 수행한 후 개발할 시스템을 개괄적으로 설계하는 것이 시스템 설계이다. 시스템 설계 동안에는 고수준의 시스템 구조가 선택된다. 시스템 설계를 마친 후 객체(상세) 설계는 세 가지 분석 모델들을 더욱 세분화하여 구현에 대한 상세한 기초를 제공한다. 즉, 객체 설계에서는 분석 모델의 실세계 지향에서 시작하여 특정 구현을 위해 요구되는 컴퓨터 지향 쪽으로 이동한다. 마지막으로 객체 설계의 결과에 따라 구현 및 유지보수 활동이 수행된다. 그리고 필요에 따라 전 단계에서 이전 단계로 피드백(feedback)이 일어날 수 있다.

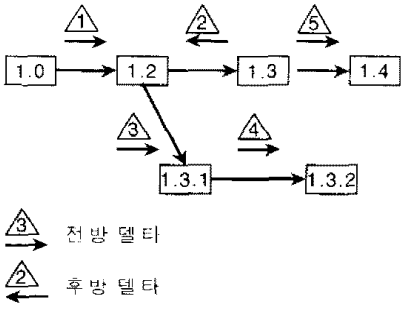
최근에 객체 지향 개발 방법론 중에서 OMG 표준화를 목표로 표기법과 모델을 통합하려는 시도로 UML(Unified Modeling Language)[15, 16]이 제안되었다. UML은 객체 지향 방법론에서 Rumbaugh의 객체 모델 표기법, Jacobson의 사용 사례들(Use Cases), 객체 상호 작용도(Object interaction diagram), Booch의 물리 모델(모듈들, 프로세스들, 분산) 등의 개념들을 통합하여 개발한 표준 모델링 언어이다. UML은 정보 시스템, 실시간 처리 시스템, 분산 시스템, 시스템 소프트웨어, 비즈니스 시스템 등에 유용하게 사용될 모델링 언어이다.

2.2 기존의 버전 제어 기법

소프트웨어 개발 중에 발생하는 식별 가능하고 기계가 인식할 수 있는 분석 종류들을 소프트웨어 객체(software object)라 정의한다[11]. 예를 들어 설계 객체, 설계 분석, 원시코드, 테스트 프로그램, 자료, 목적 코드(object code), 이진 코드(binary code) 등이 소프트웨어 객체의 예들이다. 모든 소프트웨어 객체는 유일한 식별자, 다른 소프트웨어 객체와 관계성을 기술하는 애트리뷰트들의 집합인 몸체(body), 생성된 일자와 시간 등으로 구성된다. 소프트웨어 객체는 원시 객체(source object)와 유도 객체(drived object)등으로 구성된다. 원시 객체는 소프트웨어 개발자에 의하여 생성되는 소프트웨어 객체이다. 유도 객체는 컴파일러, 링커, 문서 생성자(document formatter)와 같이 유도자(driver)라 불리는 프로그램이 원시 객체들로부터 자동으로 생성되는 객체이다. 유도 객체들은 유도자에 의하여 다시 생성 가능하기 때문에 삭제가 가능하다.

버전 제어는 소프트웨어 객체들의 다양한 버전들을 관리하는 방법이다. 소프트웨어 객체들은 버그의 수정, 유지보수, 추가 개발들 때문에 시간의 흐름에 따라 변화가 발생한다. 이들의 변화를 버전(Version), 수정(Revision), 또는 편차(Deviation)라 부르는데, 이들 동의어를 통일하여서 버전이라 부른다. 버전 제어의 목적은 같은 소프트웨어 객체에서 유도되는 많은 버전들을 효과적으로 저장·관리하는데 있다. 그리고 변화되는 과정에서 지켜야 할 규칙을 제어하기 위해서 이들 변화에 제약을 가한다. 부가적으로 소프트웨어 객체의 각 이력들을 기록하고 테스트가 요구된다.

고전적인 논문[8]의 버전번호 부여 방법을 살펴보면, 같은 구성요소의 두 개의 계승하는 버전들의 차이를 델타(delta)로 표현한다.



(그림 1) 전·후방 델타에 의한 버전들

(Fig. 1) A reversion tree with reverse and forward deltas

SCCS[8], RCS[11]와 같은 도구들의 기본 개념으로 사용하는 기본 모델은 RCS에서 원래의 1차원 경로에서 새로운 가지(branches)를 생성할 수 있도록 확장되었다. 또한 전방 델타(forward delta)에 후방 델타(reverse delta)의 개념을 추가하여 소개하였다. 후방 델타는 주어진 버전에서 이전 버전으로 변화하고, 전방 델타는 반대이다. 가장 최근의 버전을 효율적으로 저장할 수 있으며, 후방 델타를 이용하여 과거의 버전을 순방식으로 검색할 수 있다.

이러한 소프트웨어 객체의 버전들을 데이터베이스, 일반적인 파일 시스템(ordinary file), 애트리뷰트 파일 시스템(attribute file system), 버전 파일 시스템(Versioning file system)등을 이용하여 저장·관리한다. 그런데 데이터베이스는 다양한 애트리뷰트를 이용하여 소프트웨어 객체들의 선택을 보다 용이하게 하여 줄 뿐만 아니라 파일 시스템에 비하여 쉽게 변경이 가능하다. 그러나, 데이터베이스에서 야기되는 복잡성과 과부하는 무시할 수 없다. 파일 시스템은 쉽게 사용할 수 있어서 버전 제어 시스템에서 널리 사용하고 있다. 애트리뷰트 파일 시스템은 파일 시스템의 단순성과 데이터베이스의 편리성을 간직하였다. 다음에서는 기존의 분산환경을 위한 버전 제어 시스템들에 대한 장·단점을 알아 본다.

분산된 버전 제어 시스템(Distributed Version Control System : DVCS)[7]은 분산 환경에서 SCCS와 같은 소프트웨어 버전 제어 기법을 제공한다. 프로그래머가 버전들의 물리적인 위치를 알지 못해도 제어가 가능한 소프트웨어의 구성 요소의 버전 제어를 지원한다. DVCS는 다른 컴퓨터 싸이트 상에서 버전의 반복으로 증가된 신뢰도와 적응성을 제공한다. 더욱이 다른 컴퓨터 싸이트에 있는 원시 버전들을 동시에 컴파일 할 수 있고, 컴파일 된 목적 코드를 링크할 수 있다.

광역 분산 시스템(Global Distribution system: Gdist)[1]은 네트워크 상의 어느 곳에서나 SCCS나 RCS와 같은 버전 제어 시스템을 접근할 수 있다. 모든 Gdist 명령어들은 스풀 기법을 이용하여 필요할 때 자동 분산 처리의 시작과 파일의 변경이 가능하다. Gdist는 에러를 체크하고 이를 메일을 이용하여 조정자에게 통보하여 준다. 시스템이 행위에 대한 로그를 유지하여 처리의 실패에 대한 회복을 도와줄 뿐만 아니라 버그의 추적이 가능하게 해준다.

위의 분산 환경을 위한 버전 제어 시스템들은 객체 지향 프로그래밍 환경을 지원하지 못하고, 또한 버전들을 파일 시스템을 이용하여서 관리하기 때문에 객체 지향 개발 환경에서 발생하는 다양한 버전들을 관리하기에는 적합하지가 않다. 본 논문에서는 분산 환경에서 객체 지향 프로그램을 개발하는 과정에서 발생하는 버전들이 가지는 관계성의 시맨틱들을 객체 지향 데이터베이스 시스템에 효율적으로 저장·관리할 수 있는 버전 모델을 제안한다.

3. 분산 소프트웨어 개발환경을 위한 버전 모델

3.1 버전관리자를 위한 데이터모델

본 연구팀에서 제안하였던 FONASSE 데이터 모델[4, 13]은 OMT 기법의 특성을 효과적으로 지원하기 위해 OMT 개발 환경에서 발생하는 한 설계 단위의 객체들을 4차원 관점으로 파악하는 4차원 사용자 뷰를 제공하는데, 이 논문에서는 분산 환경을 위해서 프로젝트명을 추가하여 1차원을 확장한 5차원 관점으로 본다. 본 버전 모델은 OMT 기법을 이용한 객체 지향 개발 환경에서 발생하는 객체도, 상태도, 기능도(데이터 흐름도)들을 구성하는 객체들의 버전들을 효율적으로 모델링하기 위한 데이터 모델이다. 이 장에서는 이 모델에서의 설계 객체 표현 방식, 여러 설계 객체들간의 의미들을 추적 관리할 수 있는 방법에 대해서 기술한다.

3.1.1 설계 객체 표현 스키마

분산 환경을 위해 버전 모델에서 설계 객체 식별자는 5 요소 튜플 <설계객체명, 개발단계명, OMT관점명, 버전번호, 프로젝트명>로 이루어진다. 여기서, 설계 객체명은 설계 객체가 복합 계층(Composition Hierarchy)내에서 이는 한 OMT 객체인가를, 개발단계명은 개발 주기에서 어느 단계의 객체인가를, OMT 관점명은 OMT의 어떤 모델의 객체인가를, 버전 번호는 버전 계층내의 어떤 버전 객체인가를, 프로젝트명은 분산환경에서 여러 프로젝트 중에 하나의 프로젝트를 각각 명시한다. 식별자의 5차원 구성을 데이터베이스 스키마의 형태로 표현하면 다음과 같다.

정의 1. 설계 객체 스키마 SD는 다음과 같이 5 요소 튜플로 정의된다.

S_D = <설계객체명:D_1, 개발단계명:D_2, OMT관점명:D_3, 버전번호:D_4, 프로젝트명:D_5>

영역 D_1 = {모든 가능한 세분화 단계의 설계 객체명}
영역 D_2 = {요구분석(RS), 분석(A), 시스템설계(SD), 객체설계(OD), 구현(I)}
영역 D_3 = {객체모델, 동적모델, 기능모델, ω}
영역 D_4 = {모든 가능한 버전 번호}
영역 D_5 = {모든 가능한 프로젝트명}

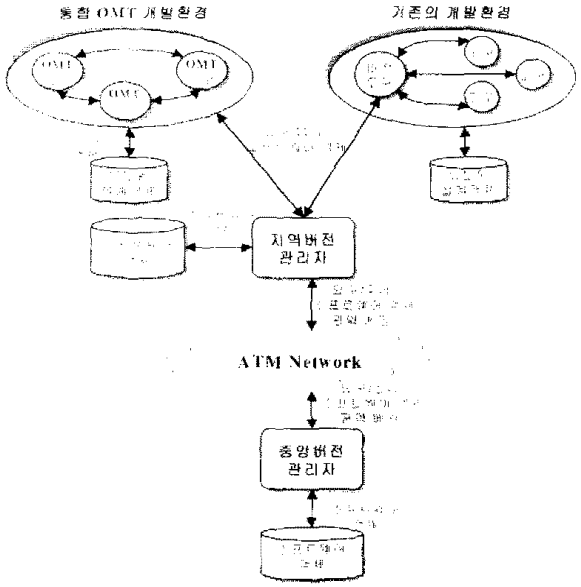
여기서 D_3의 ω는 세 모델이 구분되어 적용되지 않는 경우를 말한다. 예를 들어 시스템 설계 단계의 모든 설계 객체들은 OMT 관점명이 모두 ω이다. 한편 위의 스키마를 데이터 베이스에서 운용하기 위해서는 설계 객체 surrogate가 필요하며 이때 스키마는 다음과 같이 재구성된다.

정의 2. 설계 객체 스키마 SD는 다음과 같이 6 요소 튜플로 재정의 된다.

설계 객체 스키마 = <설계객체 Surrogate:S, 설계객체명:D_1, 개발 단계명:D_2, OMT관점명:D_3, 버전번호:D_4, 프로젝트명:D_5>

3.2 분산 소프트웨어 개발환경의 버전제어 방법론

최근에 소프트웨어 개발 환경은 통신의 발전에 힘입어 분산 환경의 성격을 갖게 되었다. 본 연구의 설계 형태는 통합된 객체 지향 소프트웨어 개발 환경과 전통적인 소프트웨어 개발 환경에서 발생한 설계 객체를 관리하는 지역 버전관리자와 지역 버전관리자에서 인증된 버전을 관리하는 중앙 버전관리자로 구성된다. 이러한 환경에서 발생하는 설계 버전 객체에 대한 관리 방법은 보통 기본적으로 두 가지 명령어(체크아웃(Checkout), 체크인(Checkin))를 사용하여 중앙 버전관리자에 저장되어 진다. 이러한 분산 환경에서 객체의 설계는 개발자들간의 공유가 가능하며 주어진 요구 명세를 만족하는 설계이어야 하고, 설계 작업 자체가 자유스러우며 효율적이어야 한다. 본 연구의 모델은 각 설계자들에게 자율성을 줄 수 있으며 투명하면서도 일관성을 유지할 수 있다. 즉, 각 시스템에서 한 프로젝트팀의 구성원으로서 특정 설계 객체를 설계하는 각 설계자는 다른 워크스테이션에서 발생시킨 동일 객체에 대한 버전명에 대해 사전에 모르고도 자신이 설계

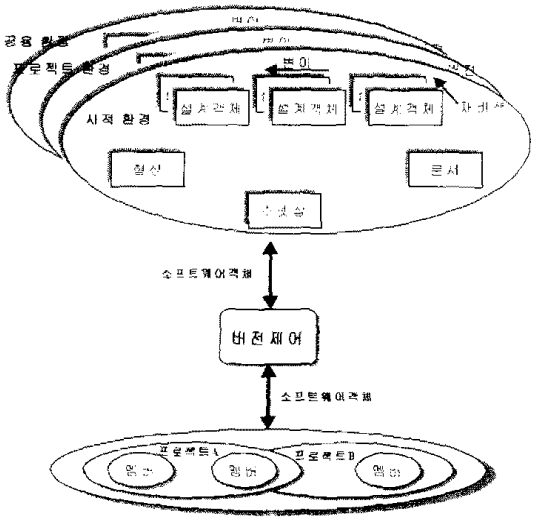


(그림 2) 버전관리자의 구조
(Fig. 2) Overview of Version Manager

(그림 2)에서 보는 바와 같이 분산 객체 지향 소프트웨어 개발 환경에서 버전관리자는 지역 버전관리자와 중앙 버전관리자로 나누어진다. 전자는 주로 버전 제어 시스템과 개발자와의 인터페이스 역할과 분산 설계 환경에서 설계 객체를 효율적으로 관리하기 위해 설계 트랜잭션들의 관리를 담당하고, 후자는 지역 버전관리자에서 요구하는 설계객체에 대해서 응답하는 역할을 한다. 이와 같이 두 레벨로 나눈 이유는 통합된 객체 지향 소프트웨어 개발 환경에서 발생하는 설계 객체와 전통적인 소프트웨어 개발 환경에서 발생하는 설계 객체를 효과적으로 관리하고, 보다 효율적으로 통합할 수 있게 하기 위함이다.

3.3 버전 관리 인스턴스의 운영

동일한 프로젝트의 두 멤버는 각각의 지역 버전관리자를 사용하여 소프트웨어를 개발한다. 각 멤버는 그것이 소유한 도구 집합과 지역 버전관리자와 지역 유도 객체를 연합한 지역 디렉토리를 소유하여 작업한다. 버전관리자의 다른 두 타입의 역할에 대해 살펴보면 다음과 같다. 첫째, 지역 버전관리자는 버전 제어 시스템을 위한 사용자의 주요 인터페이스의 역할을 한다. 그들은 사적 소프트웨어 객체의 요구에 의해 조작된다. 원격 버전관리자 인스턴스에 억세스하기 위한 서비스를 제공하고 성능을 증진하기 위해 소프트웨어 객체 캐시를 운영한다. 둘째, 중앙 버전관리자는 프로젝트와 공용 환경에 속한 소프트웨어 객체를 관리한다. 원시 객체의 초기 버전은 버전 관리 도구를 외부에서 관리한다. 이 경우에, 그 도구 또는 환경은 평소에는 지역 디렉토리에서 서로 상호 작용하며, 공용 라이브러리는 네트워크 파일 시스템을 거쳐 액세스하고 지역 유도 객체와 연합한다.



(그림 3) 다른 환경에서 소프트웨어 객체 관리
(Fig. 3) Software objects and their management in different contexts

분산 환경에서 버전들과 그들 간의 협동 관계는 (그림 3)과 같다. 모든 설계 객체 트랜잭션들은 시스템에 있는 지역 버전관리자와 중앙 버전관리자 사이의 상호 작용에 의해 처리된다.

원시 객체가 완성 레벨에 이르면 프로젝트 멤버는 버전 제어 하에서 새로운 객체를 결정할 수 있다. 누구나 지역 캐시 내에 객체만을 저장하는 지역 버전관리자를 포함하고 사적 환경인 모든 경우에는 중앙 버전관리자 보다 먼저 저장된다. 사적 원시 객체는 지역 버전관리자에 의해 총체적으로 관리된다. 구성요소를 억세스 한 경우에는 객체를 전송할 때에 오버헤드가 발생할 수 있는데, 이를 위해 중앙 버전관리자는 지역 버전관리자에서 인증한 객체를 전송 받아서 관리한다.

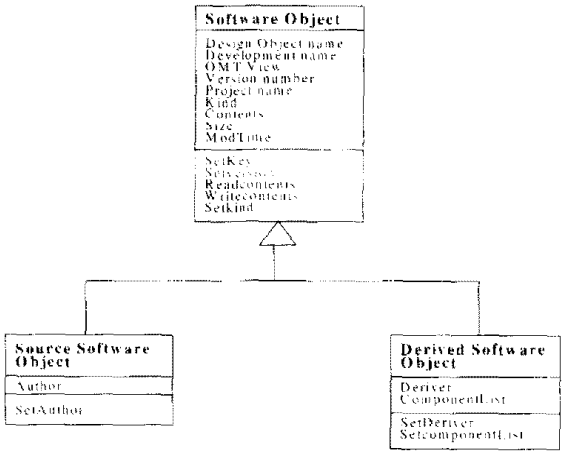
소프트웨어 객체는 버전관리를 조작하기 위한 정보의 단위이다. 그것은 식별자(설계객체명, 개발단계명, OMT관점명, 버전번호, 프로젝트명)로 구성되고, 그것의 환경, 그리고 메소드는 이들 데이터 구조에 액세스하기 위해 사용된다. 소프트웨어 객체는 개발 프로세스가 활동 중인 것을 참조하여 결과를 네트워크에 실어 항해한다.

3.4 버전관리자

버전 관리 도구는 다음 네 가지 기본 기능을 제공해야 한다. 첫째로 주어진 환경에서 소프트웨어 객체의 버전들을 제어하기 위해서는 프로젝트 이름을 줄 수 있는 기능을 가져야 한다. 둘째로 소프트웨어 객체를 추론하기 위해서는 버전 제어 아래에 버전들이 이미 존재해야 한다. 셋째로 수정 제어 시스템에서 수정된 각 소프트웨어 객체의 수정 이력 기능을 가져야 한다. 넷째로 연합 소프트웨어 객체로부터 소프트웨어 객체를 제거하기 위해서는 환경과 프로젝트 이름을 주어야 한다.

3.4.1 소프트웨어 객체 설계

시스템은 다른 환경과 분산 프로그램의 개념을 받아들인 버전 관리 서비스 제공을 강화했다. 여러 장소에서 많은 사람이 버전제어 서비스를 제공받으며 개발한다. 버전 도구는 분산환경에서 존재하고 새로운 서비스의 동합이 가능하게 하며 설계와 구현을 계속해서 유지한다. 소프트웨어 객체 클래스 계층의 객체도는 (그림 4)와 같다.

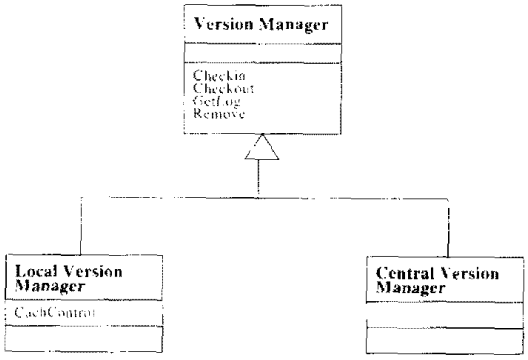


(그림 4) 소프트웨어 객체에 대한 클래스 정의
(Fig. 4) Class definition for software objects

멤버 기능으로 Setkey(IN 설계객체명, IN 개발단계명, IN OMT관점명, IN 프로젝트명)는 소프트웨어 객체 집합을 식별하는 키이다. Setversion(IN Version)은 명확한 버전에 의해 소프트웨어 객체를 식별하는 것이다. ReadContents()는 설계객체를 데이터베이스에서 주기억장치로 읽어드린다. SetAuthor(IN Author)는 원시 소프트웨어 객체를 만든 사람을 정의한다. SetComponentList(IN Component List)는 구성요소와 수정이라는 유도 소프트웨어 객체들을 결합했을 때 발생한다.

3.4.2 버전관리자 클래스 정의

버전관리자는 본 연구가 제안한 버전 관리 도구를 제공받으며, 클래스 정의 서비스는 기본이 되는 버전 관리 도구에 의해 영향을 받는다.



(그림 5) 버전관리자에 관한 클래스 정의
(Fig. 5) Class definition for version manager

(그림 5)는 버전관리자의 클래스 정의와 계층을 보여준다. 각 멤버들의 기능을 보자. CheckIn(In swobject, OUT response, IN userid, IN option, IN logmessage)은 수정 제어에서 수정한 소프트웨어 객체를 명세하여 놓는다. 로그 메시지는 변경된 객체에 관해 설명한다. 모든 로그는 유사한 구성요소를 모아서 저장하고 재검토한 후에 이력을 변경하는 성질이 있다. CheckOut(INOUT swobject, OUT response, IN userid)은 명확한 환경과 프로젝트 경우에는 수정제어를 이용해 소프트웨어 객체를 지울 수 있다.

3.4.3 지역 버전관리자

객체 지향 설계 환경은 시스템을 반복해서 상세 설계하는 과정에서 많은 버전들을 발생시킨다. 따라서

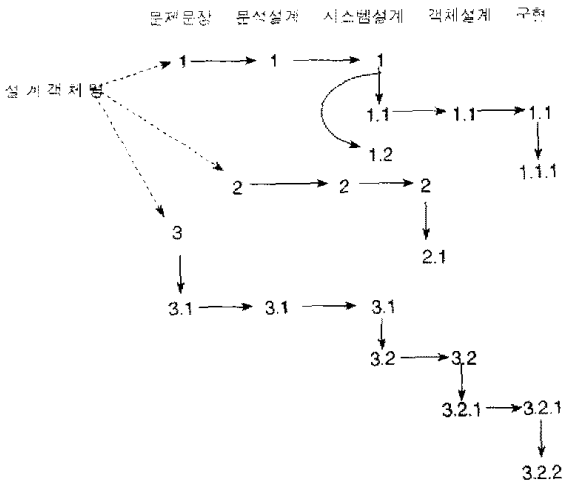
객체 지향 소프트웨어 개발 환경에서 발생하는 다양한 설계 객체 버전들을 상호 관련 지워 효율적으로 관리할 수 있게 하는 버전 제어 방법이 필요하다. 객체 지향 설계 환경에서는 각 개발 단계에서 상세화하는 동안에 다양한 설계 객체들이 상호 관련되어 버전들을 발생시키기 때문이다. 즉, 종류가 다른 설계 객체들 사이의 다양한 유도 관계성을 관리해야 하고 특히, 다른 개발 단계들간의 버전들의 상호 관련성을 유지하여야 한다. 그리고 설계계를 나타내는 세 가지 모델들의 버전들을 통합적으로 관리하여야 한다. 이러한 이유로 객체 지향 설계 환경을 지원하는 버전 제어 방법은 기존의 방법들[7]과는 몇 가지 다른 점을 가진다.

본 절에서는 OMT 기법을 기반으로 하는 소프트웨어 개발 환경의 전체 개발 단계에서 발생하는 설계 객체들의 버전들이 일관성 있게 상호 관련되도록 버전번호를 관리하는 버전 제어 방법을 제시한다. 이러한 설계 버전 제어 방법은 OMT 기반의 설계 환경뿐만 아니라 다른 객체 지향 개발 방법론에도 유사하게 적용될 수 있다.

가. 지역 버전관리자의 버전번호 부여 규칙

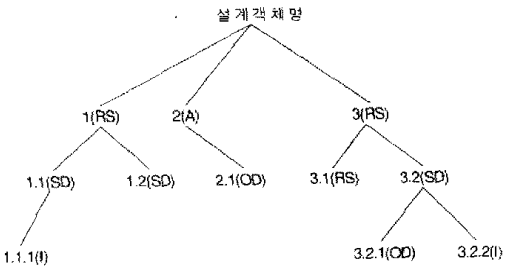
OMT 설계 환경에서는 세 가지 방법으로 버전이 발생한다. 즉, 독립적으로 발생되는 기원 버전(generic version)과 다른 설계 객체로부터 유도되는 두 가지 형태의 유도 버전들(drived versions)이 있다. 첫 번째 형태의 유도 버전은 앞의 개발 단계에 있는 설계 객체에서 유도되는 버전으로 그 객체의 형제 버전(sibiling version)이라 하고, 두 번째 형태는 같은 개발 단계에 있는 설계 객체로부터 유도되는 버전으로 자식 버전(child version)이라 한다.

(그림 6)은 한 설계 객체의 버전 유도 트리의 예를 보이고 있다. 이러한 버전 유도 트리는 시스템의 버전 클래스에서 각 설계 객체명에 대해 하나씩 관리된다. (그림 6)에서 보는 바와 같이 OMT 개발 환경에서의 각 버전은 버전번호로 관리된다. 그리고 전체 개발 과정의 모든 개발 단계에서 발생되는 같은 이름의 설계 객체 버전들은 모두 통합적으로 관리된다. (그림 6)에서 화살표는 유도되는 관계를 보이고 있다. 즉, 옆 방향의 화살표는 형제 버전, 아래 방향의 화살표는 자식 버전의 유도 관계를 보인다. 그리고 점선 화살표는 한 설계 객체의 기원 버전들을 가리키고 있다.



(그림 6) 버전 유도 트리의 예
(Fig. 6) Example of version drived tree

한편 전체 버전들의 발생 이력(유도) 관계를 관리하기 위해 각 레벨에서의 현재 버전을 전역적으로 관리한다((그림 7) 참조). 이 역시 시스템의 버전 클래스에서 각 설계 객체명에 대해 하나씩 유지된다. (그림 6)에서 각 버전번호 뒤에 첨가되어 있는 기호는 해당 버전이 최초로 발생한 개발 단계(이를 발생 기원 단계라 한다)를 가리킨다. 즉, 전역 버전 관리 트리의 각 노드에는 해당 버전번호가 최초로 발생된 단계에 대한 정보를 유지하고 있다. 예를 들어, 3.2.1(OD)은 버전 3.2.1의 발생 기원이 객체설계 단계임을 나타낸다.



(그림 7) 전역 버전 관리 트리
(Fig. 7) A war zone version management tree

OMT 개발 환경에서 이러한 전역 버전 관리 트리를 참조하여 버전번호를 부여하는 세가지 규칙을 정의한다.

정의 3. 유도되는 버전이 기원 버전인 경우는 전역

버전 관리 체계에 따라 다음 차례의 하위 레벨 버전번호를 사용한다.(기원 버전번호)

정의 4. 앞 단계 버전에서 유도되는 형제 버전인 경우는 유도되는 버전번호와 같은 자릿수의 버전번호를 사용한다.(형제 버전번호)

정의 5. 같은 단계에서 유도되는 자식 버전인 경우는 유도되는 버전번호의 마지막 자리에 한 자리 더 첨가하여 1을 부여한다.(자식 버전번호)

예를 들어, (그림 6)에서 요구분장 단계의 버전 1은 기원 버전번호, 객체설계 단계의 3.2는 3.1의 형제 버전번호, 구현 단계의 3.2.2는 3.2의 자식 버전번호이다. 그런데 버전 관리 트리에서 발생 기원 단계를 유지하는 이유는 임의의 설계 객체 버전이 앞의 세 가지 종류의 버전들(기원, 형제, 자식) 중 어느 종류인가를 곧바로 알아 낼 수 있게 하기 위함이다. 알아내는 방법은 먼저, 버전번호가 한 자리 숫자이면서 전역 버전 관리 트리에 나타난 해당 버전번호의 발생 기원 단계가 일치할 경우는 기원 버전이다. 예를 들어, 요구분장 단계의 버전 1, 분석 단계의 버전 2 등이다. 그리고 버전번호가 한 자리 이상의 숫자이면서 전역 버전 관리 트리에 나타난 해당 버전번호의 발생 기원 단계가 일치할 경우는 자식 버전이다. 예를 들면, 시스템 설계 단계의 버전 1.1, 객체설계 단계의 버전 3.2.1 등이다. 마지막으로, 전역 버전 관리 트리에 나타난 해당 버전번호의 발생 기원 단계가 일치하지 않을 경우는 형제 버전이다. 객체설계 단계의 버전 1.1, 구현 단계의 버전 3.2.1 등을 예로 들 수 있다.

한편, 버전 설계 객체를 저장하는 방법은 크게 두 가지로 나눌 수 있다. 첫 번째는 모든 버전의 정보를 명시적으로 저장하는 방법(전체 저장법)이고, 두 번째는 유도된 버전에서 변경된 내용만 저장하는 방법(증분 저장법)이다. OMT 개발 환경에서는 버전들의 발생 요인이 다르기 때문에 버전들의 종류에 따라 각기 다른 저장법을 사용한다. 즉, 버전 발생의 성질에 따라 기원 버전과 형제 버전은 전체 저장법을 사용하고 자식 버전은 증분 저장법을 사용한다. 왜냐하면 형제 버전은 유도된 앞 단계 버전과 완전히 다른 표현법을 사용하고 있고 자식 버전은 유도된 전 단계 버전과 동일한 표현법을 사용하고 있기 때문이다. 따라서 자식 버전의 경우는 변경된 내용만 저장하여 관리함으로써 저장 장소의 효율성을 증진시킨다.

나. 지역 버전관리자의 버전 상등 관계 추적 규칙

OMT 개발 환경에서 이렇게 버전번호를 부여함에 따라 각 단계들에 있어 버전들 간에 일치하는 관계성(버전 상등 관계)의 추적을 위하여 다음의 두 가지 규칙을 정의한다.

정의 6. 두 단계에서 같은 버전번호를 갖는 두 버전의 경우는 버전 상등 관계에 있다.(상등 규칙 1)

정의 7. 같은 버전번호가 아닌 경우는 뒷 단계에 있는 버전의 버전번호 자릿수들 중 앞부분에서부터 차례로 최대로 일치하는 버전이 버전 상등 관계에 있다.(상등 규칙 2)

예를 들어, 시스템설계(SD)의 3.2와 객체설계(OD)의 3.2는 상등 규칙 1에 의해 버전 상등 관계에 있으며, 요구분장(RS)의 1과 구현(I)의 1.1.1은 상등 규칙 2에 의해 버전 상등 관계에 있다.

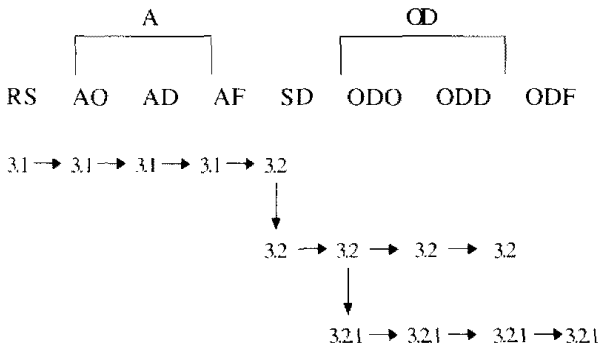
그런데 상등 규칙 2의 버전 상등 관계는 (그림 7)의 전역 버전 관리 트리를 이용하면 효과적으로 찾을 수 있다. 예를 들어, 구현 단계(I)의 1.1.1 버전에 대한 분석 단계(A)의 버전 상등 관계에 있는 버전을 찾을 경우 (그림 7)에서 루트로부터 자신(버전 1.1.1)에 이르는 경로에서 분석 단계(A) 뒤에 나타나는 기원 단계의 바로 위에 있는 기원 단계를 채택한다. 따라서 이 경우는 버전 객체 1.1은 시스템 설계(SD)가 A > SD이기 때문에 1이 채택된다. 즉, 구현 단계(I)의 버전 1.1.1에 대해 상등 관계를 갖는 분석 단계의 버전은 1이다.

다. 지역 버전 관리자의 세분화된 버전 관리 체계

앞 절에서 분석(A) 단계와 객체설계(OD) 단계는 사실 각각 세 모델(객체 모델(O), 동적 모델(D), 기능 모델(F))로 구성되어 있다. 따라서 이를 반영하는 버전 관리 체계가 필요하다. 그런데 (그림 8)에서 보는 바와 같이 이들을 각각 AO, AD, AF, 그리고 ODO, ODD, ODF로 단계를 세분화하여(펼쳐) 버전 관리 체제에 넣으면 앞의 OMT 버전 관리 방법을 그대로 적용할 수

한다. 대신에 항상 앞 단계에서의 유도는 바로 직전의 단계에서만 유도된다고 가정하며(만약 이러 단계 앞에서부터 유도하고자 할 경우는 중간 단계에 가상의 (virtual) 버전(그림 8)에서 점선으로 표시)을 발생시켜야 한다. 이 경우에 저장되는 내용은 거의 없다). 뒤 단계에서 앞 단계로 유도되는 것은 OMT 개발 환경에서 직접 지원하지 않고 설계자가 관리하도록 한다. 즉, 나중에 설계자가 필요에 따라 본래의 유도되는 버전을 새로운 버전 트리에 연결을 옮기는 것으로 가정한다. 이 경우 종래의 버전 유도 트리는 끊어지고(이 경우도 가상의 유도 관계만이 남는다.) 사용자가 의도하는 버전 트리의 새로운 노드로 연결하게 된다.

그런데 여기서 세 모델의 설계 순서를 객체 모델, 동적 모델, 기능 모델의 순서대로 개발한다고 가정하였다. 이 순서는 사실 약간의 문제점을 안고 있다. 즉, 개발되는 시스템의 성격에 따라 세 모델의 중요도가 다르기 때문에 순서가 바뀔 수도 있다. 그리고 일반적으로도 이 모델링 순서에 대해서는 학자마다 견해가 약간씩 다르다[14]. 따라서, 이러한 문제를 극복하기 위해서는 이 세 모델을 함께 통합하는 새로운 버전 관리 체계가 필요하다(즉, 항상 세 모델을 함께 묶어 하나의 버전 객체로 관리하는 방법). 그러나 이번 연구에서는 버전 제어의 복잡성 때문에 고려하지 않았다.

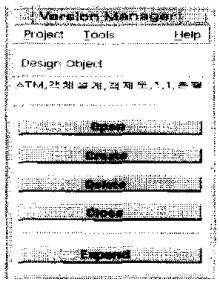


(그림 8) 세분화된 버전 관리 체계
(Fig. 8) Detailed version management tree

본 절에서는 OMT를 기반으로 한 소프트웨어 개발 환경의 각 단계에서 발생하는 다양한 설계 객체들의 버전들을 효과적으로 관리하는 방법을 제안하였다. 제안한 버전제어 방법은 OMT 개발 방법론이 아닌 다른 객체 지향 개발 방법론들에서도 약간의 수정만으로 쉽게 적용할 수 있다.

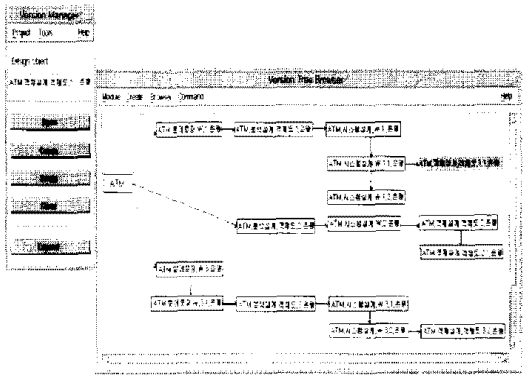
4. 버전관리자 인터페이스 상에서의 객체 버전 검색

이장에서는 지금까지 기술한 버전관리자가 어떠한 방식으로 OMT 설계 객체들을 검색할 수 있도록 지원하는가를 쉽게 이해할 수 있도록 하기 위해, 전형적인 예들을 사용자 인터페이스 상에서 설명한다. 다음 (그림 9)는 버전관리자의 주 메뉴를 보여주고 있다.



(그림 9) 버전 관리자의 주메뉴
(Fig. 9) Main menu of version manager

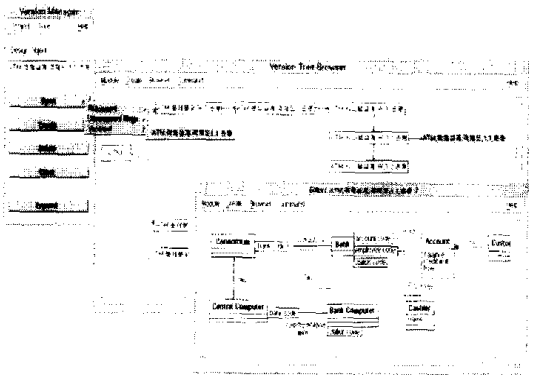
메뉴에서 Open을 클릭하면 Version Tree를 보여주며, Create는 새로운 버전을 생성하고, Delete는 버전을 삭제하는 것이며, Close는 보여준 Version Tree를 닫는 메뉴이고, Expand는 Version Tree를 확장하는 것이다. 현금 자동 지급기(ATM)의 예를 가지고서 설명하면 (그림 9)에서 (ATM, 객체설계, 객체도, 1.1, 은행)과 같은 설계 객체 명을 입력하고서 Open, Version Tree를 차례대로 선택하면 앞에서 입력한 설계 객체를 중심으로 한 기원버전 ATM(Automated Teller Machine)에서 파생된 버전들이 검색된다. 이때에 버전 트리는 버전 상등관계 추적 규칙에 의해 버전 트리들이 생성되어 (그림 10)과 같이 화면에 나타난다.



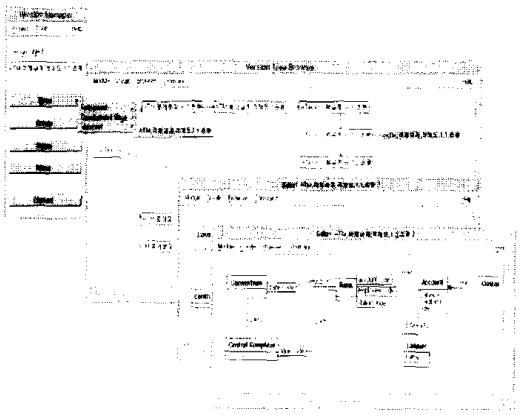
(그림 10) 브라우저에서 버전 트리를 보여주는 화면
(Fig. 10) Browsing windows for version tree

(그림 10)의 화면에서 마우스로 <ATM, 객체설계, 객체도, 1.1, 은행>을 선택한 후 'Version Manager'윈도우에서 'Open - OMT'버튼을 클릭한다. 그러면 선택된 객체도를 보여주기 위한 객체도 윈도우가 생성되고 (그림 11)과 같이 이 윈도우에서 해당 설계객체를 보여준다.

이와 같은 방법을 이용하여 찾고자 하는 객체도 버전을 쉽게 찾을 수 있을 뿐만 아니라, 동적도, 기능도 등도 같은 방법으로 검색이 가능하다. <ATM, 객체설계, 객체도, 1.1, 은행> 객체도를 검색한 후에 이 객체도를 새로운 버전으로 등록할 경우 새로운 버전이 생성되며 이때, 버전 번호는 시스템에서 자동으로 부여하는데, 이는 <ATM, 객체설계, 객체도, 1.1.1, 은행>이라는 새로운 자식 버전이 생성된다. (그림 11)에서 만들어진 객체도에 대한 다른 버전의 객체도를 (그림 12)에서 보인다.



(그림 11) 선택된 객체도를 보여주는 화면
(Fig. 11) Browsing windows for selected object diagram



(그림 12) 객체도에 대한 새로운 버전의 객체도 생성
(Fig. 12) Object diagram of creating new Version

5. 결론 및 향후 연구과제

본 논문에서는 객체 지향 소프트웨어 개발 방법론 중의 하나인 OMT를 이용한 분산 소프트웨어 개발 환경에서 다양하게 발생되는 버전들을 효율적으로 관리할 수 있는 버전제어 방안을 제시하였다. 5차원 버전 객체 공간을 기본으로 하는 버전 모델을 분산 설계 환경에서 효과적으로 버전 객체를 관리할 수 있도록 고안되었다. 따라서, 버전관리자를 사용하면 각 시스템들이 자신의 버전을 자율적으로 유지하고 개발자들은 이를 공유할 수 있으리라 기대된다. 또한, 버전들을 효율적으로 제어하기 위해서 새로운 형태의 버전 번호를 부여하는 방법을 개발하였다. 그리고, 소프트웨어 개발자들이 버전들을 효과적으로 관리할 수 있도록 지역 버전관리자의 인터페이스를 설계하여 Solaris 2.5 환경에서 Tcl/TK와 C++를 이용하여 구현하였다. 향후 연구로는 완전한 시스템을 위한 하부 저장구조가 연구과제로 남아있다.

참 고 문 헌

[1] Black, E. Paul, "GDIST : a distributed configuration control system," in Berichte des German Chapter of the ACM, Inter. Workshop on Software Version and Configuration Control, Teubner, pp.276-284, 1988.

[2] B. Bruegge, J. Blythe, J. Jacson, and J. Shufelt, "Object-Oriented System Modelling with OMT," Conf. on OOPSLA '92, Vancouver, Canada, pp. 359-376, Oct. 1992

[3] M. Chen and R. Norman, "A Framework for Integrated CASE," IEEE Software, pp.18-22, Mar., 1992.

[4] 최동훈, "객체 지향 설계 데이터 관리자를 위한 지각 질의 처리기", 박사학위논문, 전북대학교, 1997.

[5] P. Coad and E. Yourdon, 'Object-Oriented Analysis', Englewood Cliffs, New Jersey, Yourdon Press, 1990.

[6] P. Coad and E. Yourdon, 'Object Oriented Design', Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[7] B. Korel, Wedde, Horst, Nagaraj, Srinivas,

Nawas, Kaliaputc, Dayana, Venugopal, Santhanam, Babu and Xu, Mandai, "Version management in distributed network environment," in Proc. of the 3rd Inter. Workshop on software Configuration Management, Trondheim, June 12-14, 1991.

[8] M. Rochkind, "The source code control system," in IEEE transaction Software Engineering, vol.1 (4), pp.255-265, December, 1975.

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

[10] I. Thomas, "Tool Integration in the pact Environment," Proc. of the 11th Int'l Conf. on Software Eng., pp.16-18, May, 1989.

[11] W. F. Tichy, "RCS - A System for Version Control," Software Practice and Experience, Vol. 15, No.7, 1986.

[12] A. Wasserman, "Tool Integration in Software Engineering Environment," Int'l Workshop on Environment,. F. Long, ed., Springer Verlag, Berlin, pp.37-149, 1990.

[13] 양재동, 배명남, 장재우, 이종강, "객체 지향 개발 환경을 지원하는 항해형 데이터 모델", 한국정보과학회 논문지, Vol.25, No.1, pp.85-98, 1998.

[14] F. Hayes and D. Coleman, "Coherent Models for Object-oriented Analysis," OOPSLA '91, pp.171-183, 1991.

[15] Craig Larman, 'Applying UML and Patterns : An Introduction to Object Oriented Analysis and Design', Prentice Hall, 1998.

[16] Hans-Erik Eriksson, Magnus Penker, 'UML Toolkit', Wiley, 1998.

[17] 최동운, 배명남, 강현석, "객체 지향 기법을 이용한 설계 데이터 관리에 관한 연구", 정보통신 연구관리단 최종연구보고서, 경상대학교, 1993.

[18] 최동운, 양재동, 장재우, 배명남, 지동해, 강현석, "객체 지향 소프트웨어 개발 환경을 지원하는 설계 데이터 관리자", 한국정보과학회 논문지, 제21권 3호, pp.446-454, 1994.

[19] 최동운, 배명남, 양재동, 최완, "객체 모델링 기법을 사용한 소프트웨어 개발 환경에서의 시각적 질의 처리", 한국정보과학회 논문지, 21권, 10호, pp.1909-1918, 1994.

최 동 운

e-mail : edo@ozzy.chonbuk.ac.kr

1984년 전북대학교 전산통계학과 졸업(학사)

1986년 전북대학교 대학원 전산통계학과 졸업(이학석사)

1997년 전북대학교 대학원 전산통계학과 졸업(이학박사)

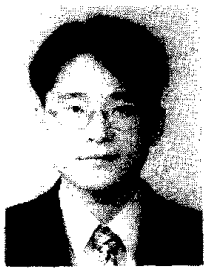
1994년~1997년 8월 서남대학교 전자계산소 소장

1996년~현재 남원시지역정보센터 이사 겸 개발책임자

1998년~현재 GLMS(주) 기술고문

1994년~현재 서남대학교 전산정보학과 조교수

관심분야 : OODBMS, CASE DB, Multimedia DB, 객체 지향 시스템, Visual Programming임.



김 수 용

e-mail : heaven@namwon.seonam.ac.kr

1996년 서남대학교 수학과 졸업(학사)

1998년 서남대학교 대학원 전산학과 졸업(이학석사)

1998년~현재 서남대학교 대학원 전산학과(박사과정)

1997년~현재 서남대학교 전자계산소 조교

관심분야 : CORBA, Multimedia DB, 객체 지향 시스템, CASE DB임.

송 행 숙

e-mail : songhs@hanil.ac.kr

1985년 우석대학교 수학과 졸업(학사)

1988년 전북대학교 대학원 전산통계학과 졸업(이학석사)

1995년 아주대학교 대학원 컴퓨터공학과 졸업(공학박사)

1997년~현재 한일장신대학교 전자통신학부 전임강사

관심분야 : Computer Graphics, Fractal, Computer Animation, CASE DB, Multimedia DB임.