論文
2-8-10

웹 기반의 분산 객체 지향 소프트웨어 개발 환경을 위한 버전 관리 모델 (Version Management Model for Distributed Object Oriented Software Development Environment Based on Web)

김 수 용* 최 동 운**
(Soo-Yong Kim) (Dong-Oun Choi)

요 약

본 논문에서는 웹 기반의 분산 소프트웨어 개발 환경에서 원시 코드 중심의 버전뿐만 아니라 원시 코드 이전 단계의 UML(Unified Modeling Language) 기반의 소프트웨어 개발 환경에서 발생하는 다양한 설계 객체들을 일정한 형태로 구축하여 효율적으로 관리하는 방법론을 제시하였다. 또한, 웹을 기반으로 한 분산 소프트웨어 개발 과정에서 발생하는 버전들을 일관성 있게 관리하기 위해 버전 규칙에 기초한 웹 기반의 규칙 버전 관리 모델을 설계 하였다.

ABSTRACT

In this paper, we proposed an efficient method of management in which various design objects in the environments for distributed software development. Those design objects are created in the environments for distributed software development based on Unified Modeling Language as well as versions of source codes. In this research, a version control technique has been specially focused that is based on our proposed version rules to consistently control versions from web-based distributed software development which enables developers to independently manage distributed objects from the software development platform. Based on the version control technique, we have designed a web-based rule-control version management model that has been proposed here.

1. 서 론

소프트웨어 개발 모델링 언어는 사전에 잘 정의된 기법들과 표현법들을 이용하여 소프트웨어를 조직적으로 생산하는 과정인데, 이러한 소프트웨어 개발 환경은 소프트웨어 개발의 각 단계를 지원하는 자동화된 도구들을 통해 구축된다[14, 15].

그러나, 도구들간에 서로 독립된 형태는 상호간의 연관성이 적어 연속적이고 통합적인 작업을 하기가 어렵다. 따라서, 각 도구들 간의 유기적인 관계를 통해 이들을 통합화된 환경으로 구축하기 위해 통합 정보 저장소에 대한 연구가 진행중이다[2, 8, 10].

* 정회원 : 서남대학교 전자계산소 조교
** 정회원 : 서남대학교 컴퓨터 정보통신학과 교수

논문접수 : 2001. 8. 3.
심사완료 : 2001. 8. 21.

최근 대형 소프트웨어 개발 방법론으로 실세계를 자연스럽게 모델링할 수 있고[3], 인터넷 기반의 네트워킹, 멀티미디어화, 분산화 등과 같이 복잡한 소프트웨어 개발 환경을 통합적으로 지원할 수 있는 객체 지향 소프트웨어 개발 방법론이 개발되고 있다[3]. 객체 기술에 관한 국제 표준화 기구인 OMG (Object Modeling Group)에서 UML(Unified Modeling Language)[13, 17, 18, 20]을 표준 객체지향 모델링 언어로 채택함에 따라 여러 부문에서 이 방법론을 적용하여 소프트웨어가 개발되고 있다. 현재까지 많은 객체 지향 CASE 도구들이 사용되고 있지만 이들은 모두 지리적으로 인접한 지역에서 독립적으로 소프트웨어 시스템을 개발하는 도구만을 제공하고 있다[1]. 그런데, 최근에 분산 환경에서 협동적인 소프트웨어 개발을 위해 많은 연구가 진행되고 있다[16]. 그러나, 분산 환경에서 객체 지향 소프트웨어를 개발하는 과정에서 발생하는 버전 관리 기법에 대한 연구가 미비한 상태이다. 본 연구는 웹을 기반으로 한 분산 소프트웨어 환경에서 하나의 프로젝트를 수행하는 과정에서 발생되는 다양한 버전들을 효율적으로 관리할 수 있는 새로운 버전 관리 방법에 관한 연구이다. 분산 환경에서의 객체지향 소프트웨어 개발과 유지 보수의 어려움으로 많은 시간과 노력이 필요하다. 이를 해결하기 위해서 분산 응용을 위한 개발 표준인 웹이 널리 이용되고 있다. 웹 환경은 기존에 존재하는 모든 다른 형태의 버전 제어 시스템을 포함할 수 있고, 사용자에게 시스템의 분산과 이질성에 대한 투명성을 보장하며, 응용 개발에 필요한 여러 기능들을 서비스로 정의하고 있다. 웹 환경의 분산 객체지향 소프트웨어 개발은 개발자들간의 공동 작업을 원활하게 하기 위한 다양한 기능들을 제공한다. 따라서 연구의 최종 목적은 기존의 원시 코드 중심의 버전 관리 시스템에 원시 코드 이전 단계인 UML을 기반으로 한 소프트웨어 개발 환경의 전체 개발 주기에서 발생하는 다양한 설계 객체들을 관리하는 웹 기반의 규칙 버전 관리 모델을 개발하는 것이다.

본 논문에서 전개될 내용은 다음과 같다. 먼저 2장에서는 버전이 발생되는 UML 소프트웨어 개발 환경과 기존의 버전 제어에 대한 특징을 살펴보고, 3장에서는 UML을 기반으로 한 소프트웨어 개발 환경에서 버전들을 관리하기 위해 버전 관리자 모델을 설계하고, 버전들을 일관성 있게 관리하기 위해 버전 규칙을 정의하였다. 4장에서는 중앙버전 관리자를 설명하였다. 마지막으로 5장에서는 결론 및 앞으로의 연구과제를 논의한다.

2. 관련 연구

2.1 UML 소프트웨어 개발 환경

통합 모델링 언어인 UML은 기존의 Booch 방법론, Rumbaugh 등의 OMT(Object Modeling Technique), 그리고 Jacobson의 OOSE(Object Oriented Software Engineering) 방법론 등을 연합하여 만든 새로운 객체 지향 소프트웨어 모델링 언어이다[6, 15]. 이 모델링 언어는 먼저 요구명세를 정의하고, 이를 분석하여 설계하고, 구현한 다음 태스트하는 소프트웨어 개발 주기의 전 과정에 단계 정의 및 단계별 업무들을 정의한다. 단계 및 업무들 간의 효율적인 상호 관계를 식별하기 위한 단계별 업무들 간의 업무흐름을 정의한다[13, 20, 23]. 기본적으로 UML에서는 도메인을 기능적으로 분할 할 수 있는 장치인 사용사례도와 사용사례 정의서를 제공하고 있으며, 시스템의 정적, 동적, 기능적인 부분을 모델링할 수 있는 기법으로 클래스 다이어그램, 순차도 또는 협력도, 상태 전이도, 그리고 활동도를 제공하고 있다. 관련있는 클래스들을 결합할 수 있는 장치로 패키지도를 제공하고 시스템에 대한 물리적 장치를 표현할 수 있는 기법으로 다형도를 제시하고 있다. 이 모델링 언어는 문제를 표현하는 문장 형식인 요구 명세로부터 시작한다. 이를 위해 요구 명세에서는 사용사례도와 간단한 클래스도 그리고 활동도로 표현이 되어진다. 그런데 대개 문제 문장은 불완전하고 규정에 맞지 않기 때문에, 분석 단계에서 요구 사항의 모호함과 불일치성을 해결하여 실세계에 대한 모델을 명확히 해주어야 한다. 이를 위해 분석 단계에서는 클래스도와 순차도 그리고 협력도 그리고 상태도 그리고 활동도가 만들어진다. 설계 단계에서 분석 단계의 결과물에 기술적인 부분을 첨가하여 확장하는 것이다. 기술적인 확장이란 시스템을 어떻게 구현할 것인지에 초점을 두고 어떻게 동작하

고 어떤 제약이 있어야 하는지에 관하여 생각하는 것이다. 설계 단계에서 클래스도, 순차도 그리고 협력도 그리고 상태도 그리고 활동도, 구성요소도, 배치도가 만들어진다. 마지막으로 설계의 결과에 따라 구현 및 테스트 활동이 수행된다. 그리고 필요에 따라 뒷 단계에서 앞 단계로 피드백(feedback)이 일어날 수 있다.

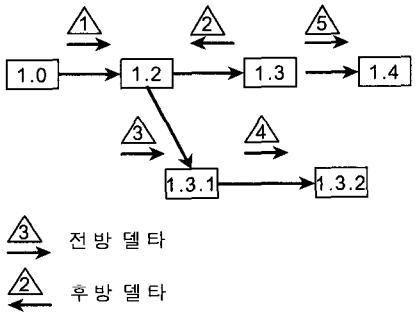
2.2 기존의 버전 제어

소프트웨어 개발 중에 발생하는 식별 가능하고 기계가 인식할 수 있는 문서 종류들을 소프트웨어 객체(software object)라 정의한다[19]. 예를 들어 설계 객체, 설계 문서, 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 코드 생성기, 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 다양한 버전의 소프트웨어 객체를 관리하기 위한 기법이다. 버전 제어는 같은 구성 요소의 많은 버전들을 효과적으로 저장·검색하는데 목적이 있다.

버전에 고전적인 논문[12]의 버전번호 부여 방법을 살펴보면, 같은 구성요소의 두 개의 계승하는 버전들의 차이를 델타(delta)로 표현한다. SCCS[12], RCS[19]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. 이는 가장 최근의 버전을 경제적으로 저장할 수 있으며, 후방 델타를 이용하여 과거의 버전을 성공적으로 검색할 수 있다. 이러한 소프트웨어 객체의 버전들을 데이터베이스, 보통의 파일(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 준다. 더욱이 보통의 파일 시스템에 비하여 쉽게 변경이 가능하다.

분산된 버전 제어 시스템(Distributed Version Control System : DVCS)[4]은 분산 환경에서 SCCS와 같은 소프트웨어 버전 제어 기법을 제공한다. 프로그래머가 버전들의 물리적인 위치를 알지 못해도 가능한 소프트웨어의 구성 요소의 버전 제어를 지원한다. DVCS는 다른 컴퓨터 사이트 상에서 버전의 반복으로 증가된 신뢰도와 적응성을 제공한다. 더욱이 다른 컴퓨터 사이트에 있는 원시 버전들을 동시에 컴파일 할 수 있고, 컴파일 된 목적 코드를 링크할 수 있다.

광역 분산 시스템(Global Distribution system : Gdist)[5]은 네트워크 상에서 어느 곳에서나



[그림 1] 전·후방 델타에 의한 버전

[Fig. 1] Version Tree Using of Forward · Reverse Delta

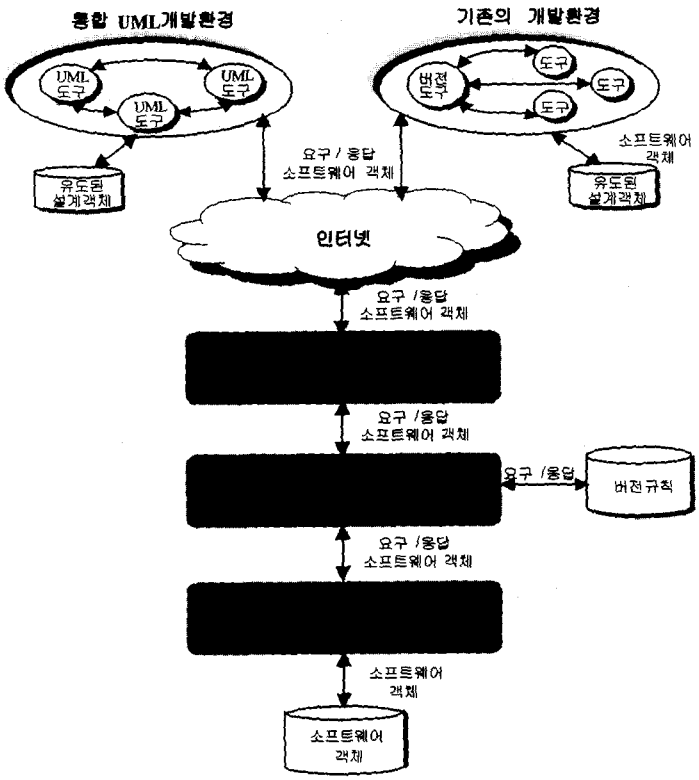
SCCS나 RCS와 같은 버전 제어 시스템을 접근할 수 있다. 모든 Gdist 명령어들은 스풀 기법을 이용하여 필요할 때 자동 분산 처리의 시작과 파일의 변경이 가능하다. Gdist는 에러를 체크하여 메일을 이용하여 조정자에게 통보하여 준다. 시스템 행위에 대한 로그를 유지하여 처리의 실패에 대한 회복을 도와줄 뿐만 아니라 버그의 추적이 가능하게 해준다.

Rational Software사의 ClearCase는 현대 소프트웨어 개발의 복잡성을 동적으로 해결해 주는 실용적인 소프트웨어 형상 관리 시스템으로 4가지 기능적 영역으로 나누어 설명하는 것이 유용하다[22]. 4가지 기능에는 작업공간 관리, 관리 구축, 프로세스 제어, 버전제어가 있다. 그 중에 Rational ClearCase의 Version Control는 모든 파일과 디렉토리에 변경을 추적하는 것에 의해 버전을 완전하게 유지한다. Rational CelarCase의 핵심 개념은 버전 트리이며, 그것은 그래픽으로 정보를 표현하고 구조화하며, 디렉토리 트리와 비슷한 계층적인 포맷으로 표현하는 특징을 갖는다. 그러나 Rational ClearCase는 파일과 디렉토리 기반으로 버전을 관리하므로써 웹상의 분산 개발팀들 간의 설계 정보를 공유할 수 있는 적절한 방법을 제공하고 있지 못하고 있다. 즉, 웹 기반의 분산 환경에서 소프트웨어 개발하는 팀들이 설계 정보를 공유하고, 관리하기에 많은 어려움을 느낀다. 본 논문에서 제안하는 웹 기반의 버전 관리자는 UML의 설계 정보를 관계형 데이터베이스에 저장 관리한다. 즉, 설계 정보의 의미 정보와 표기 정보를 관계형 테이블에 사상하여 웹 기반의 분산 환경에서 개발자들간의 공동 작업이 가능하도록 하였다.

3. 웹을 기반으로 한 UML 소프트웨어 개발환경을 위한 버전 모델

3.1 웹 기반의 규칙 버전 관리자 모델

본 연구는 웹을 기반으로 한 분산 소프트웨어 개발 과정에서 발생하는 버전들을 일관성 있게 관리하기 위해 버전 규칙에 기초한 버전 제어 기법을 제안



[그림 2] 웹 기반의 규칙 버전 관리자

[Fig. 2] Rule Version Manager based on Web

하였다. 분산되어 있는 소프트웨어 개발 환경에서는 설계 정보를 상호 교환할 수 있어야 하며, 더 나아가 이들을 보다 보편화된 웹 환경을 위해서 확장해야 한다. 그러기 위해서 우리 팀에서는 이러한 부분을 지원하기 위해서 웹 기반의 분산 객체 지향 소프트웨어 개발 환경을 위한 규칙 버전 관리자를 설계하였다. 전체 시스템 구조는 크게 형상자와 중앙 버전 관리자로 나누어진다. 전자는 주로 버전 제어 시스템과 개발자와의 인터페이스 역할과 웹 기반의 분산 설계 환경에서 발생되는 설계 객체를 효율적으로 관리하기 위해 설계 트랜잭션들의 관리와 버전 생성 시 버전 규칙과 제약조건 모두를 만족시키는지 평가한다. 후자는 버전 규칙에 기초한 형상자에서 두 가지 명령어 체크인(Checkin), 체크아웃(Checkout)를 거쳐 중앙버전 관리자에서 저장, 검색, 갱신, 삭제 등이 이루어진다. 즉, 형상자에서 요구하는 설계 객체에 대해 응답하는 역할을 한다. 이와 같이 두 레벨로 나눈 이유는 버전들의 일관성 제어와 기존의 소프트웨어 개발 환경과 통합 UML 소프트웨어 개발 환경에서 발생하는 설계 객체를 효과적으로 관리하고 통합할 수 있게 하기 위함이다.

3.2 버전관리자를 위한 규칙 버전 모델

본 연구팀에서 제안하였던 FONASSE 데이터 모델[21]은 개발 환경을 위해 4차원 관점으로 파악하는데, 이를 UML 기법의 특성을 효과적으로 지원하기 위해 UML 개발 환경에서 발생하는 한 설계 단위의 객체를 6차원 관점으로 파악하는 6차원 사용자 뷰를 제공하도록 변경하였다.

분산 환경에서 버전 모델은 소프트웨어 객체 식별자를 다음과 같이 6요소 튜플 <설계 객체명, 개발단계명, UML 관점명, 버전번호, 운영체제, 데이타베이스>로 나타낼 수 있다. 여기서, 설계 객체명은 설계 객체가 복합 계층(Composition Hierarchy)내에서 어느 한 UML 객체인가를, UML 관점명은 UML의 어떤 모델의 객체인가를, 개발단계명은 개발 주기에서 어느 단계의 객체인가를, 버전 번호는 버전 계층 내의 어떤 버전 객체인가를 명시한다. 운영체제는 응용프로그램이 실행되는 운영체제를 의미하며, 데이터베이스는 응용프로그램에서 사용되는 데이터베이스를 명시한다. 식별자를 데이터베이스 스키마의 형태로 표현하면 다음과 같다.

(정의 1) 소프트웨어 객체 스키마 S_D는 다음과 같이 6요소 튜플로 정의된다.

S_D = <설계객체명:S_1, 개발단계명:S_2,
UML 관점명:S_3, 버전번호:S_4,
운영시스템:S_2, 데이터베이스:S_6>

영역 S_1= {모든 가능한 세분화 단계의 설계 객체명}
영역 S_2= {요구문장(RS), 분석(A), 시스템설계(SD), 객체설계(OD), 구현(I)}
영역 S_3= {UML 모델중 하나, ω}
영역 S_4= {버전 번호}
영역 S_5= {사용 가능한 운영체제명, ω}
영역 S_6= {사용 가능한 데이터베이스명, ω}

여기서 S_3, S_5, S_6의 ω는 UML모델이 적용되지 않는 경우를 말한다. 예를 들어 시스템 설계 단계의 모든 설계 객체 중에 요구명세서도 ω에 속한다. 소프트웨어 객체를 식별하기 위해 surrogate와 구현 단계의 이형공간을 위해 운영시스템과 데이터베이스를 추가하여 스키마를 재구성하였다.

(정의 2) 소프트웨어 객체 스키마 S_D는 다음과 같이 7요소 튜플로 재정의 된다.

소프트웨어 객체 스키마(S_D) = <Surrogate:S_1, 설계객체명:S_2, 개발단계명:S_3, UML 관점명:S_4,버전번호:S_5, 운영시스템:S_6, 데이타베이스:S_7>

3.3 버전 구조

3.3.1 원시 버전번호 규칙

소프트웨어 객체는 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. 구축단계에서의 버전은 이형 공간에 대한 버전이다.

원시 버전들은 별개로도 사용되지만, 하나이상의 버전들이 결합되어 어떤 하나의 버전을 생성할 수도 있다. 이때 버전들은 결합과정에서 대부분 일관성을 잃게 되고, 단지 소수의 버전만이 일관성을 유지하게 된다. 이러한 문제를 해결하기 위해 우리는 결합 과정에서 버전 규칙을 적용하여 버전 데이터베이스로부터 일관된 버전을 구성하려 한다. 원시 객체에 버전 규칙을 적용하게 된 동기는 원시 객체 버전들의 결합 동안에 규칙을 적용하여 원시 버전의 일관성을 제어할 수 있기 때문이다. 일관성 제어 문제는 버전 공간과 프로덕트 공간 사이에서 다루게 되는데 버전 공간의 경우, 버전 규칙은 일치하지 않는 결합을 제거하는데 이용되며, 프로덕트 공간은 소프트웨어 객체에 관한 지식, 이것의 내용, 이것들의 상호연관성에 대해 프로덕트 제약조건을 체크하고 책임지는데 이용된다. 원시 버전의 경우 버전 규칙은 데이터베이스 내에 저장된 버전을 저장 검색하기 위해 사용된다.

(정의 3) 원시 버전 규칙은 다음과 같다.

수정공간
(1) t=max∧branch=main
(2) name=alpha
(3) merge(v1, v2)
(4) 원시 버전 번호
(5) 유도 버전 번호

이형공간
(6) os=unix∧ws=x11∧db=informix
(7) ¬(os=unix∧ws=x11)

변경공간
(8) c1 c2 c3
(9) c2 ⇒ c1
(10) c1⊗c2⊗c3

설계공간
(11) d1=설계객체명∧d2=UML관점명∧d3=개발단계명

(정의 3)의 규칙(1)과 (2)는 형상을 구성하기 위해 버전 그래프로부터 만들어진 버전을 선택한다. 규칙(1)은 메인 가지에 있는 가장 최근에 수정된 버전을 선택한다. 규칙(2)는 alpha에 속한 버전을 찾기 위한 상징적 이름을 사용했다. 규칙(3)은 2개의 버전 V_1과 V_2의 결합에 의해 생성된 새로운 버전을 구성하는데 사용된다. 이형 공간의 경우, 버전 규칙은 이형 애트리뷰트의 값으로 간주한다. 규칙(4)는 원시버전 번호를 유일하게 식별하는 번호이다. 규칙(5)는 원시버전과 유도버전간의 관계성을 정의한다. 유도버전이 어떤 원시 버전에서 유도되었는지를 파악할 수 있으며, 종속성과 설계 객체들간의 관계성과 상속성을 명확히 할 수 있다. 규칙(6)는 운영 시스템, 윈도우 시스템, 데이터베이스 시스템을 명시하여 이형을 정의한다. 규칙(7)는 이형공간에 대한 제약조건을 정의했다. 변경 공간의 경우 규칙(8)은 변경을 적용하기 위해서 버전을 분류하기 위해 사용된다. 규칙(8)과 (9)은 일관성 변경 결합에 사용된다. 규칙(9)은 변경 c_1이 c_2를 포함한 상태이다. 즉, c_2는 c_1을 기초로 한 변경 구조이다. c_2는 변경된 구조만 가지고 있다. 규칙(10)은 c_1, c_2, c_3가 변경 상태 임을 나타낸다. 설계 공간의 경우 규칙(11)는 설계객체명, 개발단계명, UML 관점명, 버전번호를 명시하여 설계 공간을 정의한다.

(정의 4) 원시 버전 규칙의 유효범위는 다음과 같다.
(1) status=checked_out
(2) *:os=unix∧ws=x11∧db=informix
(3) b.DependsOn*:name=alpha
(4) *:d_1∧설계객체명∧d_2=UML관점명∧d_3=개발단계명
(5) *:sv=원시버전명∧sn:원시버전번호∧dv=유도버전명∧dn:유도버전번호

(정의 4)의 규칙(1)은 단일 모듈 a에 적용하며, 현재 사용자에 의해 버전이 체크되며 선택된다. 규칙(2)의 *는 전체적인 이형공간을 나타내고, 동일한 이형과 모듈은 전체 프로덕트를 통해서 선택되어진다. 규칙(3)에서 b에 적용된 규칙은 b에 의존적인 모든

모듈에 적용된다. 규칙(4)의 *은 설계 공간내의 설계 객체를 나타낸다. 규칙(5)의 *은 유도버전이 어떤 원시 버전에 의해 유도되었는지 버전들의 이력을 나타낸다.

(정의 5) 원시 버전 규칙은 다음과 같은 순서로 적용되어 원시 버전을 선택하게 된다.

constraint
(1) *:os=unix∧ws=x11∧db=informix
(2) *:d_1=설계객체명∧d_2=UML관점명∧d_3=개발단계명∧d_4=버전번호

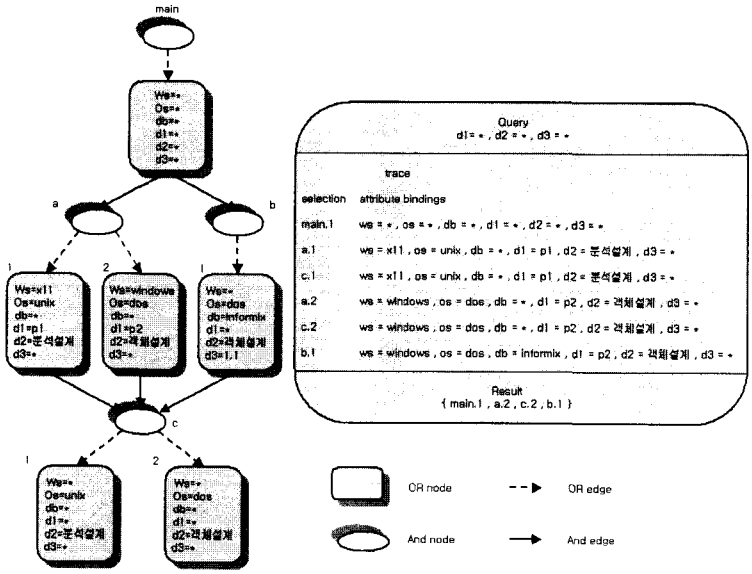
reference
(3) *:status=checked_out
default
(4) *:t=max

(정의 5)의 규칙(1)은 이형공간에 속한 버전을 모두 선택할 때 사용된다. 규칙(2)는 설계공간에 속한 설계 객체를 선택한다. 규칙(3)은 버전을 선택하여 이용 가능한 상태인지 체크한다. 규칙(4)는 가장 최근에 변경된 버전을 선택한다.

3.3.2 원시 버전 형상자

형상자는 중앙버전관리자의 모든 모듈에 대한 통합 인터페이스 역할을 하고, 4개의 형상자가 결합된 구조이다. 첫째, 상태 지향 기능 형상자는 버전 데이터베이스 상에서 작동하며, 이것은 델타 방법을 적용한 형태이다. 버전 규칙은 제어 표현 형태로 취한다. 제어 표현은 버전의 적절한 구성요소를 선택하기 위한 튜플을 평가한다. 규칙 기반 형상자는 일관성을 검사하거나 제한조건을 실현한다. 둘째, 상태 지향 규칙 기반 형상자는 적당한 검색 방법을 통해서도 선택하지 못할 때 지원하는 형상자이다. 버전 선택이 이루어짐에 따라, 제한조건은 더해진다. 운영 시스템내의 어떤 이형이 선택되었다면, 이러한 선택은 형상 프로세스를 통해서 전달되며, 모든 노드를 위해 일관되게 수행되어 질 것이다. 만약 이것이 불가능하다고 판단 된다면, 형상자는 잘못 선택된 이형을 역추적 해야 한다. [그림 3]에서 보여준다.

a.1를 선택한 후에 d2 애트리뷰트는 분석설계에 결합되어 있다. 하지만 b의 선택은 실패하고 트리거는 역추적을 한다. 결국은 a.2를 선택한 후에 d2 애트리뷰트는 객체설계와 결합되고, db의 애트리뷰트는 informix와 결합 된다. 셋째, 변경 지향 기능 형



[그림 3] 형상의 규칙 기반 구조
[Fig. 3] Configurator or Rule-Based Structure

상자는 변경 순서에 지원을 받는다. 내부적으로 데이터베이스는 전방 델타나 혹은 델타를 포함하여 사용한다. 개념적으로 변경은 어떤 기본적인 버전 순서 내에서 적용 받는다. 불일치한 변경 결합을 찾기 위해서 사용될 수도 있다. 불일치한 것은 변경 동작이 실패했을 때 찾게 된다. 넷째, 변경 지향 규칙 기반 형상자는 형상 프로세스 내에서 버전 규칙을 적용함으로써 선택하게 된다.

3.4 유도 버전의 구조

유도 객체의 생성 과정을 시스템 구축이라 설명한다. 유도 버전 구축을 위한 형상자를 구축자라 하며, 구축자는 구축 계획에 따라 실행되고, 버전 규칙에 맞게 실행된다.

3.4.1 유도 버전 규칙

시스템 구축은 원시 객체와 일관된 유도 객체를 유지하는 것과 깊은 관련이 있다. 이러한 관계성에 따라 다음과 같은 요구사항이 따른다. 첫째, 구축 단계는 정확한 순서 내에서 실행되어야 한다. 둘째, 변경의 경우에 구축 단계에 영향을 미치는 모든 것은 재실행되어야 한다. 셋째, 구축은 최소 시간에 이루어져야 한다. 넷째, 구축은 정보에 종속되어 의존한다. 다섯째, 유도 객체를 보관하는 저장 공간을 최소화해야 한다. 이런 요구 사항은 원시 레벨에서 고려되어진 것으로서 서로 다르다. 이런 선택은 형상 프로세스를 극복함으로써 수행되어지며, 모든 버전 규칙이나 프로덕트 제한조건에 따라 움직인다. 유도 버전을 구축했을 때, 우리는 일차적으로 구축의 효율성과 정확성을 고려해야 한다. 대부분 구축 규칙은 구축 결과에 관해서 결정적이다. 유도 객체의 버전 규칙은 버전 규칙 상태에서 어떻게 하나의 유도 버전이 원시 버전에 의존하여 구축되어지는 과정을 보여준다. 구축 규칙이라는 것은 구축 단계가 수행됨을 의미한다. 다시 말하자면, 입력 객체에 동작을 주어 출력 객체가 행해진다. 일반적으로 동작은 출력 객체를 생산하기 위해 입력 객체 상에 어떤 도구로 자극한다. 시스템 규칙은 사용자에 의해 변경되지 않으며, 버전 시스템에 의해서 변경되어진다. 사용자 정의 규칙은 사용자에 의해 변경이 이루어진다.

(정의 7) 유도 버전의 구축 규칙은 다음과 같다.

1) instance-level rules

```
foo : {d1, d2, d3, d4}
foo # uml
foo : main.o a.o b.o c.o
cc main.o a.o b.o c.o -o foo # link
main.o : {d1, d2, d3, d4} main.h a.h b.h
cc -c main.c # compile
```

2) type-level rules

```
uml [mdl : design object]
conversion [m : design object]
compile [m : Module]:
m.invalid # precondition
{cc -c m.c} # tool activation
not m.invalid; # postcondition success
m.invalid; # postcondition failure
```

(정의 7)은 인스턴스와 타입 레벨 규칙에서의 구축 규칙에 대한 설명이다. 첫째, 타입 레벨 규칙은 어떤 규칙의 모든 인스턴스로 적용된다. 예를 들어, 하나의 타입 레벨 규칙은 각각의 ".o" 파일의 상태이며, ".c" 파일에서 생성된 것이고 ".h" 파일도 포함한다. 둘째, 인스턴스 레벨 규칙은 어떤 타입의 구체적인 인스턴스로 불려진다. 규칙은 정해진 순서에 의해서 적용된다. 제약조건은 강제적인 규칙이고 이것을 반드시 만족시켜야 한다. 규칙과 제약조건이 만족되어질 때만이 성능이 최적화 된다. 규칙은 우선순위에 의해 적용되는데 높은 우선순위의 규칙은 낮은 우선순위 규칙 전에 고려된다. 우선순위는 명시적으로 할당되거나 본래 순서에 의해 암시적으로 정의된다.

3.4.2 유도버전 형상자

형상자는 잘못 선택한 것으로부터 역추적 해야 하는 형상자로 유도 객체의 구축을 고려한다. 증가 애트리뷰트 평가에 밀접하며, 평가는 빠르게 수행되어진다. 형상자는 구축을 가속화 하는데 사용한다. 규칙 기반 형상자는 어떤 규칙 기반 내에서 저장된 구축 규칙에 의해 운영된다. 완전한 구축 기록은 기원 그래프에 나타난다. 각 파라메터는 구축 단계의

출력에 영향을 주며, 이것은 입력으로서 기록된다. 이 입력은 원시 객체나 유도 객체, 도구 옵션, 도구 버전, 운영 시스템, 호스트를 포함한다. 재구축은 다음에 의해 조작되는데 변경 입력이나 수정 기원 그래프의 경우 오래된 기원 그래프를 복사하여 구축하거나 이형과 함께 모든 구축 단계의 출력을 대신한다. 구축단계의 실행이나 오래된 출력의 재사용을 말한다. 각 구축 단계를 위해 이러한 결론은 선언 차이에 근거를 두고, 예전 것과 새로운 입력을 비교할 수 있다.

4. 결론 및 향후 연구과제

본 논문에서는 기존의 원시코드 중심의 버전제어 시스템에 원시 코드 이전 단계 UML을 기반으로한 소프트웨어 개발 환경에서 다양하게 발생되는 설계 객체들을 효율적으로 관리할 수 있는 버전제어 관리 규칙을 제안하였다. 우리의 모델은 웹 기반의 분산 설계 환경에서 효과적으로 설계 객체를 관리할 수 있도록 고안되었다. 따라서, 웹 기반의 분산 객체 지향 소프트웨어 환경에서는 보다 더 많은 개발자가 자유로운 장소에서 자유로운 시간에 데이터를 공유하며 서로의 의견에 대해 교환할 수 있다. 그렇지만, 아직 설계 객체 저장 구조를 완전히 설계하지 못하였으며, 앞으로 완전한 시스템으로의 구현 작업이 연구과제로 남아있다.

※ 참고문헌

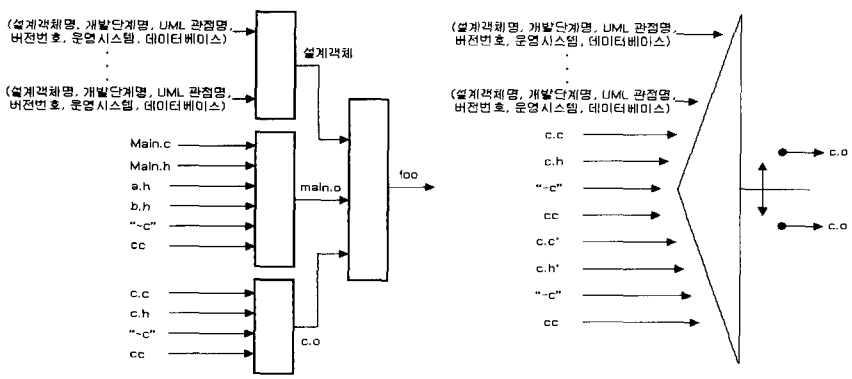
[1] A. Versey, A. P. Sravana, "CASE as Collaborative Support Technologies," Communication of the ACM, pp. 83-94, Jan. 1995.

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

[3] 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.

[4] B. Korel, Wedde, Horst, Nagaraj, Srinivas, Nawas, Kaliaque, 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.

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



(a) derivation (b) difference predicate

[그림 4] 시스템 구축을 위한 버전 모델

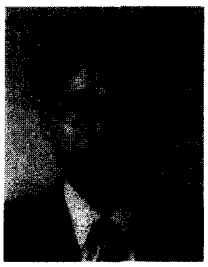
[Fig.4] Version Model for System Building

[6] Booch, Rumbaugh, Jacobson, The Unified Modeling Language user Guide, Feb., 1999.
[7] Hans-Erik Eriksson, Magnus Penker, UML Toolkit, John Wiley & Sons, 1998.
[8] I. Thomas, "Tool Integration in the pact Environment," Proc. of the 11th Int'l Conf. on Software Eng., pp. 16-18, May, 1989.
[9] J. Rumbaugh et al., Object-Oriented Modeling and Design, Prentice Hall, 1991.
[10] M. Chen and R. Norman, "A Framework for Integrated CASE," IEEE Software, pp.18-22, Mar, 1992.
[11] M. Gaedke, Hans-W. G., A. Schmidt, Ulf S., Wolfgang Kurr, "Object-oriented Web Engineering for Large-scale Web Service Management," Proc. of the 32nd Hawaii international conference on System Science, IEEE, 1999.
[12] M. Rochkind, "The source code control system," in IEEE transaction Software Engineering, vol. 1 (4), December, pp. 255-265, 1975.
[13] Martin Fowler, Kendall Cott, UML Distilled, Addison-Wesley Longman, 1997.
[14] P. Coad and E. Yourdon, Object-Oriented Analysis, Englewood Cliffs, New Jersey, Yourdon Press, 1990.
[15] P. Coad and E. Yourdon, Object-Oriented Design, Prentice Hall, Englewood Cliffs, New Jersey, 1991.
[16] Rational Software et. al, OA&D CORBA-facility, OMG document number: ad/97-08-09.
[17] Rational Software et. al, UML Extension for Business Modeling, OMG document number: ad/97-08-07.
[18] Rational Software et. al, UML Summary, OMG document number: ad/97-08-03.
[19] W. F. Tichy, "RCS - A System for Version Control," Software-Practice and Experience, Vol. 15, No. 7, 1986.
[20] 김형준, "UML을 이용한 컴포넌트 설계기술," 한국 내쇼날소프트웨어 Feb., 1999.
[21] 최동운, "객체 지향 설계 데이터 관리자를 위한 시각 질의 처리기," 박사학위논문, 전북대학교, 1997.
[22] http://www.rational.com/products/clearcase, Rational Rose Co.
[23] http://www.rational.com/uml, Rational Rose Co.

최 동 운

1984년 전북대학교 전산과 졸업(학사)
1986년 전북대학교 대학원 전산과 졸업(이학석사)
1997년 전북대학교 대학원 전산과 졸업(이학박사)
1994년 ~ 1998년 8 서남대학교 전자계산소 소장
1994년 ~ 현재 서남대학교 컴퓨터 정보통신학과 교수
관심 분야 : 지능형 에이전트, 객체 지향 시스템, 웹 공학, 웹 데이터베이스, XML 저장소

김 수 용



1996년 서남대학교 수학과 졸업(학사)
1998년 서남대학교 대학원 전자계산학과 졸(이학석사)
1998년 서남대학교 대학원 컴퓨터정보통신학과 (박사수료)
1999년 ~ 현재 서남대학교 전자계산소 조교
관심 분야 : 객체 지향 시스템, 웹 공학, 웹 데이터베이스, XML 저장소