웹 서비스를 이용한 자동화된 버전관리 시스템 구현

김남호° 박용범
단국 대학교 정보아키텍처 연구실
firstoend°@gmail.com, ybpark@dankook.ac.kr

An Implementation of Automation Version Management System Using Web Services

Nam-ho Kim° Young B. Park
Information Architecture Lab. Dankook Univ.

요 약

소프트웨어가 발전함에 따라 프로젝트의 규모가 커지게 되고, 참여 인원이 늘어나게 된다. 대규모의 프로젝트를 관리하는 것은 매우 복잡하고 어려운 작업이다. 버전관리 시스템은 이러한 대규모의 프로젝트를 관리하는데 있어 매우 유용한 도구로써 사용되고 있다. RSC, SCCS, 클리어케이스, CVS등 많은 버전관리 시스템이 있지만, 그 중 CVS는 대표적인 버전관리 시스템으로써 대부분의 오픈 소스 프로젝트에서 사용되고 있을 정도로 그 성능을 인정받고 있다. 본 연구에서는 버전관리 시스템의 가장 완성적인 형태라고도 볼리는 CVS에 대한 특징을 연구하고 CVS가 가지는 장·단점을 살펴보고 CVS의 성능을 개선시킬 수 있는 자동화된 버전관리 시스템을 제안한다.

1. 서론

소프트웨어가 발전함에 따라 현재의 소프트웨어 개발은 한·두명의 프로그래머가 아닌 팀 단위의 다수의 개발자들에 의해 복합적으로 개발이 이루어지게 된다 [3]. 복잡한 소프트웨어를 다수의 개발자들이 공동으로 개발하는 과정에서 상당한 수의 소스코드나 문서와 같은 파일들이 생성되고. 이 과정에서 소프트웨어 개발 프로세스 컨트롤, 변경 관리, 의사소통 지식전달 문제 등을 발생시킨다. 버전관리 시스템은 이러한 문제점들을 해결함에 있어 매우 유용한 도구라 할 수 있다 [10].

버전관리 시스템은 동일한 파일들의 여러 버전을 관리해 주는 시스템이다. 버전관리 시스템은 하나의 파일에 대한 변경사항들을 저장하므로 파일에 대한 변경의 이력을 확인하고 언제라도 과거 특정 시점의 파일을 확인할 수가 있다는 장점이 있다. 또한 팀원들이 하나의 소스 트리 상에서 효율적을 작업 할 수 있도록 도와준다 [6].

가장 오래되고 대표적인 버전관리 시스템의 하나인 CVS(Concurrent Versions System)는 현재 대부분의 오픈 소스 프로젝트에서는 CVS 를 이용하여 공동작업을 진행하고 있다. CVS 가 매우 유용한 버전관리 도구이지만 사용자가 직접 개발중인 소스코드를 저장하고 받아와야 하고, 사용자가 임의로 파일을 변경하거나, 비효율적인 네트워크 대역폭 활용 등 몇 가지 단점들이 존재한다. 본 연구에서는 CVS 가 가진 단점들을 해결하고 사용자에게 보다 편리한 개발환경을 제공하기 위해 웹 서비스 기반의 자동화된 버전관리 기능을 가진 시스템을 제안한다.

이를 위해 2 장에서는 CVS 에 대해 소개하고 CVS 의 단점과 장점을 알아본다. 그리고 웹 서비스에 대한 기술을 소개하며 버전관리 시스템을 위한 웹 서비스 아키텍처를 고려한다. 3 장에서는 자동화된 웹 서비스 기반의 버전관리 시스템을 제안한다.

2. 관련연구

2.1 버전 관리 시스템(Version Control System)

소프트웨어를 개발함에 있어서 버전관리란 소프트웨어에의 개발에 따른 변경을 기록으로 남기는 것이다. 이것은 각각의 소스코드나, 문서와 같은 파일들에 대한 변경일 수도 있고, 시스템 전체에 대한 변경일 수도 있다. 이러한 버전관리 시스템으로는 RCS(Revision Control System) [9], SCCS(Source Code Control System) [7], CVS 와 같은 버전관리 시스템들이 있다.

버전관리 시스템들의 기본적인 개념은 개발중인 프로그램의 파일들을 마스터 복사본(master copies)과 작업용 복사본(working copies)으로 나누는 것이다. 마스터 복사본은 저장소(repository)에 저장하게 되고, 개발자들은 저장소로부터 자신들의 작업용 복사본을 가져와(check out) 변경한다. 변경된 파일들은 다시 저장소에 저장하게 된다(check in). 매번 사용자들이 check in 할 때 마다 버전관리 시스템은 저장소에 새로운 버전의 파일들을 만들어 저장하게 된다[8].

버전관리 시스템에 의해 관리되는 버전들은 보통 델타(delta) 방식을 사용한다[4]. 델타방식이란 소스코드의 변경된 부분만을 저장하는 방식이다. 이 방식을 사용하는 대부분의 버전관리 시스템들은 그림 1 과 같은 구조의 버전 트리를 형성하게 된다[9].

이러한 기법은 파일의 변경에 대한 히스토리를 저장할

수 있도록 한다. 변경에 대한 히스토리를 저장한다는 것은 프로젝트를 개발하는데 있어 매우 유용한 도구가 된다. 이것은 사용자로 하여금 파일의 변경에 대한 추적(tracking)을 가능하게 하여, 개발자는 언제든지 자신이 과거에 작업했던 시점의 파일을 가져와 볼 수 있게 되고, 실수로 지워버린 코드들을 언제든 원하는 시점으로 돌려 놓을 수가 있다[5].

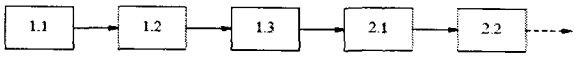

그림 1. 버전 트리

버전관리의 또 다른 중요한 개념의 하나는 트렁크(trunk)와 브랜치(branch) 이다. 트렁크란 개발의 중심이 되는 부분을 의미하며 브랜치란 트렁크로부터 분리된 작은 줄기를 의미한다[1].

다수의 개발자들이 동일한 버전의 파일에 대하여 작업을 할 경우에는, 하나의 파일에 대해 동일한 부분을 수정할 경우가 발생하고 이 경우 충돌(conflict)이 발생했다고 한다. 충돌을 해결하기 위한 일반적인 방법으로는 두 가지가 있다. 하나는 잠금 수정(lock-modify-unlock) 방식이고, 다른 하나는 수정 병합(copy-modify-merge) 방식이다[8].

잠금 수정 방식은 말 그대로 개발자가 수정하고자 하는 파일에 대해 잠금(lock)한 상태로 수정하고 수정이 끝나면 잠금 해제(unlock)하는 방식이다. 다른 사람들은 잠금이 걸린 파일에 대해서는 읽기는 가능하지만 수정하거나 저장은 불가능하게 된다. 이 방식은 잠재적인 충돌 위험을 최소화 할 수 있다는 장점이 있으나 반면에 한번 잠금이 걸린 파일에 대해서는 잠금 해제가 될 때까지 어느 누구도 수정할 수 없기 때문에 대규모의 공동작업에서는 작업진행 속도에 영향을 줄 수가 있다는 단점도 있다.

수정 병합 방식은 저장소에 있는 파일을 복사하여 수정한 후에 다시 병합하는 방식이다. 이 방식의 특징은 잠금 수정 방식과는 다르게 한 사람이 하나의 파일에 대해 작업을 하고 있는 동안 다른 사람이 그 파일에 수정을 가할 수 있다는 것이다. 이 방식은 개발자들이 하나의 파일에 대해 다른 서로 다른 부분을 수정하면 아무런 문제없이 파일을 병합(merge) 할 수가 있다 그러나 충돌이 발생할 경우 개발자들간의 의견교환을 통해서 충돌이 발생한 부분을 수정해 주어야 한다.

잠금 예약 방식은 간단하지만 엄격하게 충돌을 제어해 준다. 그러나 다른 지역에 있는 다른 사람들과의 공동작업을 하는데 있어서는 불편한 점을 가지고 있다. 수정병합방식은 잠금 예약 방식에 비해 시간과 장소에 구애 받지 않고 원활한 개발을 진행 할 수 있다는 장점이 있다. 현재 가장 대표적인 버전관리 시스템인 CVS 에서 사용하고 있는 방식이기도 하다.

2.2 CVS (Concurrent Version System)

CVS 는 대표적인 버전관리 시스템으로써 오픈 소스 프로젝트로 이루어지는 대부분의 프로젝트 개발에서 사용되고 있으며 앞서 설명한 버전관리의 대부분의 기능을 지원한다. CVS 는 클라이언트/서버 구조로 공동 작업하는 프로젝트의 소스코드를 저장소라 불리는 CVS 서버에 저장하고 개발자들은 서버로부터 필요한 소스 코드를 다운받아 사용할 수 있어 공동 프로젝트를 지원해 준다는 장점을 가지고 있다[2].

CVS 는 다음과 같은 체크아웃(check out), 커밋(commit) 기능, 업데이트(update), 히스토리(history) 기능, 변경사항의 저장, 병합기능, 트렁크와 브랜치, 태그(tag) 기능, 비교(diff) 기능, 기능을 지원한다[10].

- 체크아웃: 저장소로부터 모든 소스와 속성 파일들을 받아 오는 과정.
- 커밋: 체크아웃 한 소스를 변경한 후 변경된 소스를 저장소에 갱신, 충돌 여부가 검사 된다.
- 업데이트: 저장소에 있는 소스들의 최신 버전을 가져오는 기능
- 히스토리 기능: 파일의 변경에 대한 히스토리를 CVS 서버에 기록하는 기능.
- 병합 기능: 여러 명의 개발자가 하나의 소스를 동시에 수정 시 CVS 가 병합하는 기능.
- 트렁크: 개발의 메인이 되는 주요 부분.
- 브랜치: 프로젝트에서 중요한 부분에 대한 변경이나 소스 트리의 대규모의 변경 시 트렁크에서 분리 됨.
- 태그 기능: 개발이 빌드 되거나 릴리즈 되는 특정 시점 등의 버전을 표시 함.
- 비교 기능: 버전과 버전, 태그와 태그 간의 소스의 차이점을 비교 함.

이와 같은 특징들은 CVS 를 프로젝트 개발에 있어 매우 유용한 버전관리 시스템으로 만들어 준다.

다음 그림 2 는 CVS 의 기본적인 동작과정을 보여준다. CVS 의 작업사이클은 크게 프로젝트의 생성 단계와 진행 단계로 구분할 수 있다. 프로젝트 생성 단계에서는 그림 2 의 ①, ② 와 같이 프로젝트의 리드 개발자가 CVS 에 프로젝트 저장소를 만들고(init), 프로젝트의 초기 버전에 대한 소스, 문서 등의 파일들을 서버에 저장하게 된다(import). 프로젝트 진행 단계에서는 개발자들은 우선 서버로부터 자신의 로컬 컴퓨터로 작업환경을 가져오고(그림 2-③), 각 개발자들은 서버로부터 받은 파일을 추가하거나 삭제하고, 커밋한다(그림 2-④). 소스파일의 최신버전을 유지하기 위해서 서버에서 최신 버전의 파일들을 가져온다(그림 2-⑤). Update 중에 충돌이 발생한 것은 병합하거나 수정하여 다시 커밋한다(그림 2-⑥). 개발자들은 ④~⑥의 과정을 반복하여 작업하게 된다. CVS 를 사용하는 개발자들은 충돌을 줄이기 위해서 이와 같은

과정을 통해서 항상 파일들의 최신 버전을 유지해야 한다.

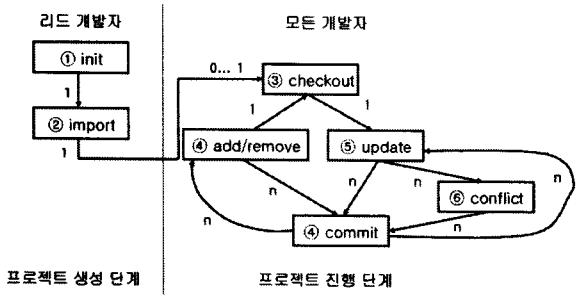

그림 2. CVS 작업 사이클

2.3 웹 서비스 (Web Services)

웹 서비스는 서비스 지향 아키텍처(SOA: Service Oriented Architecture)의 근본적인 개념을 구체화한 기술의 집합체이다. 웹 서비스는 매우 느슨한 결합력을 가진 XML(Extensible Markup Language) 기반의 비즈니스 분산 객체이다. 웹 서비스는 표준 RPC(Remote Procedure Call)을 통해 프로토콜에 의존적이지 않도록 배치되어 바인딩 될 수 있고, 표준 HTTP(Hyper Text Transfer Protocol)을 사용하여 통신하므로 제반 비용이 적고, 어느 곳에서든 HTTP 프로토콜을 사용하여 접근할 수가 있다는 장점이 있다. 웹 서비스가 가진 느슨한 결합력은 웹 서비스가 확장성과 유연성을 가지게 함으로써, 웹 서비스를 컴포넌트화 하여 재조립하고 새로운 웹 서비스로의 구성이 가능하게 한다[12].

웹 서비스는 표준화된 기술을 사용하여 비즈니스 어플리케이션을 통합하고 통합된 어플리케이션에 대하여 투명한(transparent) 인터페이스를 제공한다. 다음은 웹 서비스의 표준 기술이다[11].

- SOAP(Simple Object Access Protocol): XML 기반의 메세징 프로토콜, 특정 벤더에 종속되지 않아 웹 서비스에서 사용되는 모든 메시지는 SOAP 을 사용하여 통신함[13].
- WSDL(Web Service Description Language): XML 기반의 웹 서비스 기술 언어, 웹 서비스에 접근하고 웹 서비스를 사용하기 위한 메시지 스키마를 정의함[15].
- UDDI(Universal Description, Discovery and Integration): 웹 서비스 등록을 위한 레지스터(Register), WSDL 을 UDDI 에 등록하고 웹 서비스를 공표(publish)함[14].

웹 서비스는 위와 같은 표준을 사용하여 공개적으로 서비스를 등록하고 등록된 서비스를 사용할 수 있도록 해주는 새로운 웹 어플리케이션이다. 웹 서비스 사용자는 UDDI 를 통해 웹 서비스를 검색할 수 있으며, WSDL 을 이용하여 클라이언트를 생성하고 웹 서비스를 사용할 수 있게 된다. 다음 그림 3 은 웹 서비스의 기본적인 구성과 동작방식을 보여주고 있다.

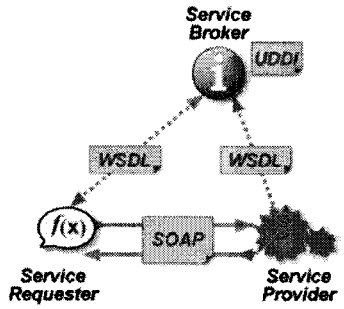

그림 3. 웹 서비스 구성 및 동작

3. 웹 서비스를 이용한 자동화된 버전관리 시스템

프로젝트를 개발할 때 소스파일들은 매번 변경이 되고 저장소에 저장된다. 개발자들은 소스의 커밋과 업데이트 시 발생할 수 있는 충돌을 최소화 하기 위해서 로컬 컴퓨터에 있는 파일들을 항상 최신 버전으로 유지하는 작업이 필요하게 된다. CVS 를 사용하는 경우 이러한 충돌을 막기 위해선 매번 commit 과 update 를 해주어야 하는데 이것은 개발자들에게 매우 불편한 부분이다. 이를 위해선 개발자들 사이에 충분한 커뮤니케이션이 이루어져야 하고 항상 다른 개발자의 진행상태를 확인 할 수 있어야 하지만 현재 CVS 에서는 이러한 부분에 대해서는 아무런 도움도 주지 않는다. 또한 CVS 를 사용할 경우 여러 명의 개발자들이 하나의 서버에 집중되어 네트워크 트래픽(network traffic)이 많이 발생한다.

본 연구에서 제안하고 있는 자동화된 버전 관리 시스템(Automation Control Version Management System: AVMS)은 이러한 CVS 의 단점에 대한 해결 방안을 보여준다.

3.1 시스템 설계

다음의 그림 4 는 AVSM 의 아키텍처를 보여준다.

AVMS 는 크게 클라이언트와 서버의 구조를 가지고 두 가지의 통신 방법을 사용한다. 첫째는 SOAP 메시지를 이용한 웹 서비스에 접근 방법이고 두 번째는 파일 전송을 위한 TCP/IP 를 사용하고 있다. 이는 클라이언트와 서버간의 네트워크 트래픽을 줄이기 위한 방법으로 AVMS 서버와 저장소를 서로 다른 곳으로 위치시킬 수가 있다.

클라이언트 부분은 로컬 에이전트를 통해서 사용자의 파일에 대한 변경사항을 모니터링 하고, 파일의 변경이 있을 경우 변경에 대한 부분을 트랜잭션 처리하여 서버에 저장하게 된다.

서버는 크게 버전관리와 파일관리에 대한 부분으로 나누어 준다. 버전관리 부분은 사용자가 명시하는 버전을

관리하는 부분이고 파일 관리는 자동적으로 백업되는 파일들을 관리하는 부분이다. 파일에 대한 히스토리 관리는 파일전송관리자에 의해 동작하며, 실제 파일이 전송될 때 로그를 작성하게 된다.

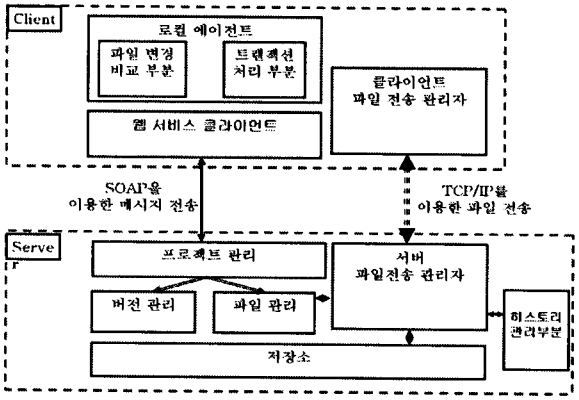

그림 4. AVMS 아키텍처

3.2 Automation Version Management System (AVMS)

3.2.1 자동화된 파일 백업

AVMS 의 기본적인 기능은 기존의 CVS 와 거의 유사하다. 그러나 CVS 에서 지원하지 않는 자동화된 파일백업 기능을 사용하여 CVS 의 단점을 해결하였다.

자동화된 파일 백업은 AVMS 의 기본적인 개념으로 기존의 대부분의 버전관리 시스템이 가지고 있는 버전관리의 기본개념과 같다. 그러나 AVMS 는 파일에 대한 변경이 있을 경우 이를 자동으로 서버에 저장하고, 저장된 내용을 다른 사용자에게 자동으로 업데이트 시키게 된다. 이러한 점은 기존의 CVS 에서 수정, 커밋 업데이트 세 단계에 걸쳐 동작하는 부분을 하나의 자동화된 트랜잭션으로 처리할 수 있다는 장점으로, 사용자들에게 큰 편의성을 제공해주는 부분이다. 자동화된 파일 백업은 파일에 대한 변경이 있을 때 마다 수동으로 커밋하고 업데이트 해야 한다는 CVS 의 번거로움을 해결한다.

3.2.2 AVMS 의 버전 관리

모든 버전관리 시스템이 그렇듯 기본적인 버전관리 방식은 사용자가 변경한 파일에 대한 기록을 남기고, 파일의 변경된 부분을 저장소에 저장하는 것이다. 그리고 저장된 파일에 대해 새로운 버전을 부여함으로써 새로운 버전의 파일이 생성된다. 기존의 CVS 에서는 이러한 부분을 사용자의 명령어에 의해 동작하던 부분이었지만 AVMS 는 자동으로 이러한 부분을 해결해 준다.

AVMS 가 변경된 파일에 대해 버전을 붙이는 것은 CVS 와 다른 방식을 취한다. AVMS 는 사용자의 명령에 의해서가 아니라 자동적으로 파일을 저장하는 경우에 파일에 버전번호 대신 파일이 변경된 시각의 기록을 남긴다. 이것은 파일에 대한 히스토리 태그(history tag: HT)로써 사용자가 파일을 저장할 때 마다 파일의 변경 사항을 체크하고 변경된 부분이 있을 경우 자동으로 파일을 저장한다. 저장된 파일은 히스토리 로그(history log)를 남기게 되고, 히스토리 로그가 남겨진 파일에 대해서는 현재 동일한 프로젝트를 진행하고 있는 다른 사용자에게 자동으로 업데이트 된다.

AVMS 가 자동으로 파일에 대한 변경 사항을 저장하기 위해 히스토리 로그를 사용하지만, 이것은 사용자의 의지와 상관 없이 파일의 변경 내용을 저장하므로 사용자가 원치 않는 불필요한 부분까지도 기록으로 남게 된다. 불필요한 부분을 저장함으로써 저장소의 공간을 소비할 수 있고, 각 파일의 변경에 대해 무분별하게 저장된다는 단점을 가지게 된다. 이러한 문제를 해결하기 위한 방법으로 사용자가 명시적으로 파일에 대한 변경의 백업을 요청할 수가 있다. 사용자가 명시적으로 파일의 백업을 요청하게 되면 AVMS 는 백업 태그(backup tag: BT)를 사용하여 현재까지 작업된 파일의 히스토리 로그를 하나의 변경으로 간주하고 백업 로그(backup log)에 저장하게 된다. 이때 기존에 있던 히스토리 태그는 모두 사라지고, 새로운 백업 태그 만이 남게 된다.

다음 그림 5 는 이러한 과정을 보여주고 있다. 그림 5 에서 히스토리 태그 HT200701051124004~5 와 HT200701051124006~8 는 사용자가 명시적으로 작성한 백업 태그 BT005~6 에 의해서 감추어 지게 되고, 이전 히스토리 태그에 이어 새로운 히스토리 태그인 HT200701051124009 로 이어진다.

각 파일에 대한 버전관리는 그림 5 와 같이 히스토리 태그와 백업 태그를 사용하여 관리하게 되고, 사용자는 차후 개발에 있어 중요한 시점을 하나의 프로젝트 버전으로써 관리 할 수 있다.

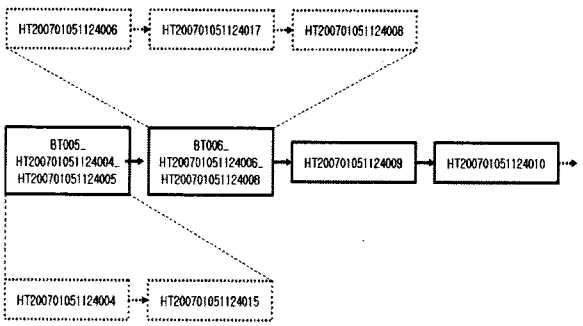

그림 5. HT와 BT를 이용한 버전관리 트리

AVMS 에서의 프로젝트 버전은 전체 프로젝트에서 큰 의미를 두거나 개발의 중심적인 부분이 있을 경우 프로젝트 리더에 의해서 명시되는 부분이다. 이 경우 그림 6 에서 보듯이 AVMS 는 현재까지 명시된 백업 태그들 중 최신의 태그들만을 모아서 하나의 버전으로 체크한다. 이

때 AVMS 는 기존 버전의 파일들을 자동으로 동결 시키고, 새로운 버전을 위한 저장소를 만들어 체크된 이후의 파일들을 복사한다. 이는 프로젝트의 버전관리에 있어 무결성을 보장하기 위한 방법으로 버전관리를 프로젝트 단위로 관리하여 개발자들이 버전을 나눌 때 좀 더 의미 있는 단위로 나눌 수 있도록 해준다. 그림 6 은 이러한 예를 보여주고 있다.

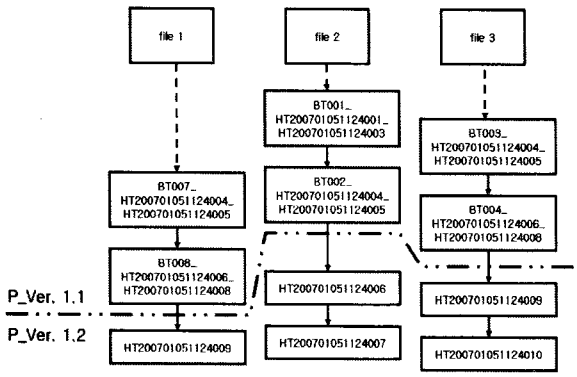

그림 6. 프로젝트 버전 관리

그림 6 은 현재 진행 중인 프로젝트에 버전 P_Ver 1.1 을 명시한 경우로 file1, file2, file3 의 가장 최신 버전의 백업 태그인 file1-BT008, file2-BT002, file3-BT004 를 개발자가 명시한 P_Ver 1.1 로 묶고, 그 이후에 변경된 히스토리 태그들은 새로운 버전인 P_Ver 1.2 로 묶이게 된다. 그리고 P_Ver 1.2 의 파일들을 새로운 저장소로 복사하고 프로젝트를 진행하게 된다. 이때, 개발자들은 기존 버전의 파일들은 볼 수 있지만 수정은 할 수 없고, 만약 파일에 수정을 원한다면 기존의 파일을 복사해 새로운 버전의 파일로 만들게 된다.

만약 현재 만들어진 버전이 릴리즈 되거나 프로젝트의 구조적으로 중요한 변경을 가지게 될 경우, 버전 태그(version tag)를 사용하여 체크할 수 있다. 이는 CVS 의 브랜칭과 태그에 해당하는 부분으로 프로젝트에 대한 중요변경에 대한 내용을 좀 더 손쉽게 관리 할 수 있도록 지원해 준다.

AVMS 에서의 충돌은 다음의 두 가지가 있다. 첫째는 둘 이상의 사용자로부터 동시에 같은 파일이 커밋되는 경우이다. 이 경우 AVMS 는 이들 파일에 대한 충돌을 감지하고, 파일들에 대해 각각의 HT 를 부여하고 동일한 충돌 태그를 사용해 체크한다. 체크된 파일들은 파일을 변경한 개발자들에게 통보가 되어 각 개발자들이 충돌에 대한 처리를 할 수 있도록 위임한다. 두 번째는 A 사용자가 파일에 대해 수정하고 있는 도중에 B 사용자가 동일한 파일에 대한 수정 작업을 마치고 커밋하는 경우이다. 이 경우 AVMS 는 자동적으로 커밋된 파일을 업데이트 하기 때문에 A 사용자는 B 사용자가 커밋한 파일에 대해서 확인하지 못하고 파일을 덮어쓰게 된다. 이 경우 사용자는 자신이 수정한 소스에 대한 무결성을 보장 받을 수가 없다. 이러한 문제점을 해결하기 위해 AVMS 는 사용자가 파일을 서버에 커밋하기 전에 수정된 파일에 대한 변경사항을 체크하고 현재 커밋하려는 파일이 마지막에 업데이트된 파일인지의 여부를 판단한다. 만약 현재 커밋하려는 파일이 마지막에 업데이트된 파일이라면 파일이 오버라이팅(overwriting) 되었는지를 판단하고 사용자에게 경고를 주게 된다.

3.3 시스템의 평가

다음 표 1 은 CVS 와 AVMS 를 비교 평가한 결과 이다.

표 1. CVS와 AVMS의 기능 평가

	CVS	AVMS
자동 파일백업 기능	지원 안함	지원
커밋	수동	자동
업데이트	수동	자동
트랜잭션 처리 (atomic commit-update)	지원 안함 (수동으로 작업)	자동화 지원
버전관리 단위	파일	프로젝트 단위
브랜칭	지원	버전 태그 사용
파일 변경 추적	버전번호, 로그파일사용	히스토리 태그, 히스토리 로그 사용

표 1 에서 보이듯이 AVMS 는 CVS 에서 지원하지 않는 자동 파일백업 기능과 자동 커밋, 자동 업데이트, 트랜잭션에 대한 처리를 지원하고 있다. 또한 버전관리의 단위를 파일이 아닌 프로젝트로 관리하여 보다 효율적인 버전관리 기능을 지원하고 있다. 또한 파일의 변경에 대한 추적을 히스토리 태그를 사용하여 지원하고 있으므로 변경에 대한 내용을 추적하는데 있어서도 많은 편의성을 제공해 준다.

4. 결론

프로젝트의 규모가 커지고, 참여 인원이 많아질수록 소스 코드와 문서 등의 파일을 관리하기가 어려워지는 대규모의 소프트웨어 개발에 있어서 버전관리 시스템은 개발자들에게 필수적인 도구가 되었다. 그 중 대표적인 버전관리 시스템이라 할 수 있는 CVS 는 현재 대부분의 오픈 프로젝트에서 사용하고 있을 정도로 그 성능이 입증되었고, 버전관리 시스템의 완성형이라고도 불린다. 그러나 CVS 를 사용할 겨우 발생 할 수 있는 충돌을 줄이기 위해서 사용자들은 파일들은 항상 최신 버전으로 유지해야 한다는 단점이 있다. 이것은 개발자들에게 무척 귀찮은 일이 될 수 있다. 본 연구에서 제안한 자동화된 버전관리 시스템은 자동화된 파일 백업 기능과 웹 서비스를 이용하여 이러한 문제점을 해결하고, 사용자들에게 좀 더 많은 편의성을 제공한다. 현재의 버전관리 시스템은 기본

적으로 CVS 의 동작방식에 따라 작동하고, 그 중 자동화 될 수 있는 부분들을 찾아 사용자들의 개입 없이 처리해 줌으로써 이러한 문제점들을 해결한다.

향후에는 제안된 시스템에서 취약한 충돌에 대한 부분을 보완할 수 있는 연구가 수행되어야 한다.

참고 문헌

[1] Alan J. Dix, "Version Control for Asynchronous Group Work," YCS 181, 1992

[2] Alexander Yip, "Pastwatch: a Distributed Version Control System " , http://ris.lcs.mit.edu/irisbib/papers/pastwatch:nsdi06/paper.pdf

[3] B R Gaines, "Version Control in Collaborative Writing", IPCC, 1994

[4] Brian O'Donovan, "A Distributed version control system for wide area networks," 1990

[5] Bryan O'Sullivan, "Distributed revision control with Mercurial," http://hgbook.red-bean.com/hgbook.html, 1996

[6] John Plaice, "A New Approach to Version Control", IEEE TRANSACTIOS ON SOFTWARE ENGINEERING, VOL. 19, NO. 3, 1993, 03

[7] Marc J. Rochkind, "The Source Code Control System, " IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, Vol. SE-1, No. 4, 1975

[8] Panagiotis, "Version Control," IEEE SFTWARE, 2006

[9] Walter F. Ticky, "RCS - A System for Version Control," Software-Practice & Experience 15, 1985

[10] 민진우, "이클립스 기반 필수 유틸리티," 한빛, 2004, 01

[11] 이강찬, "웹 서비스 표준화 현황", IE 매거진, 2004

[12] 이경하, "SOA(Service-Oriented Architecture)와 웹 서비스", 정보과학회지 제 22 권 제 10 호, 2004, 10

[13] "SOAP Version 1.2 Part 0: Primer (Second Edition), " http://www.w3c.org/TR/SOAP, W3C Recommendation, 2007, 04

[14] "UDDI Spec TC (UDDI Version 3.0.2), " http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm, OASIS, 2004, 10

[15] "Web Service Description Language (WSDL) 1.1," http://www.w3c.org/TR/wsdl, W3C Note, 2001, 03