공학 연구논문집 제29권 제1호 pp. 83~97, 울산대학교.
Journal of Engineering Research Vol. 29, No. 1, pp. 83~97, 1998
University of Ulsan.

# 공학설계 데이타베이스의 버전 모델링 및 관리에 관한 연구*

고 재 진
컴퓨터정보통신공학부

**<요 약>**

CAD 데이타와 같은 공학설계 데이타의 응용과 관리를 위한 지원 메카니즘은 데이타베이스 연구자의 중요한 관심사가 되어 있다. 본 논문에서는 그러한 지원 메카니즘의 하나로 공학설계 데이타베이스의 버전 모델링 및 관리에 중점을 둔다. 이것은 다양한 표현 형태를 갖고 시차적으로 발전하는 복잡한 공학제품 데이타베이스를 구성하고 조직하는 개념을 정립하고, 그것을 통해서 그러한 제품에 관한 서술을 만들고, 수정하는 연산을 정립하는 것이다. 공학 데이타 모델 분야에서 그러한 것을 위한 새로운 개념과 메카니즘들이 많이 제안되어 있다. 본 논문에서는 그러한 제안들을 통합하고 공통의 용어를 정립하고, 주어진 버전 환경의 요구에 맞는 새로운 틀을 제안하고자 한다.

# A Study on the Version Modelling and Management for the Engineering Design Database

Jae Jin Koh
Division of Computer Engineering & Information Technology

**<Abstract>**

The support mechanism to apply and manage the engineering design data such as CAD data has been the great interests to the database researchers. In this paper a modelling and management of the engineering database versions as the support mechanism are mainly

* 본 연구는 1997년도 울산대학교 연구비 지원에 의해 수행되었음

considered. The concepts to construct and organize the complex engineering artifacts databases which evolve with the various representations and time sequences are constructed, and the operations to describe the artifacts and modifiy them are established by the concepts. Several new concepts and proposals for them in engineering data model area were proposed. In this paper by unifying the proposals and by establishing the common terminologies, the new framework which meets the given version environments is proposed.

# 1. 서 론

기존의 데이타베이스 관리시스템은 규칙적으로 조직된 데이타를 저장하고 접근하는 데는 효율적으로 대처해 왔다. 이와 대조적으로 수백만 개의 트랜지스터 LSI 프로세서 칩의 서술에 대해서 생각해보자.

상호 연결된 트랜지스터의 서술뿐만 아니라, 수십만개의 논리 게이트, 수천줄의 하드웨어 서술 언어, 수백만 개의 집적회로 마스크 배치도, 수 많은 시험 데이타, 수백 개의 문서철등이 있다. 그런 복잡성을 체계적으로 이해하기 위해서, 설계자는 계층도를 사용해서 서술들을 의미있는 단위로 구분해서 집체시킨다. 그 서술들은 여러 단계로 나누어 지고, 동일한 실체를 여러 표현 형태로 서술하는 부분들은 상호 연관시킨다. 설계부분이 변경되고, 새로운 설계버전이 만들어지더라도 일관된 설계 관점을 유지하여야 한다. 설계환경은 넓은 범위의 기법들을 포함하고 있지만, 본 논문에서는 공학설계 데이타베이스 환경에서 고찰하도록 한다. 공학설계 환경은 다음의 사항들에 관심을 둔다. 그것은 기존의 DBMS의 확장으로 필요한 응용분야를 표현해야 하고, 효율성과 편리성을 제공하고, 공학설계 DB를 위한 동시성 제어기능, 일관성 유지 기능, 고장 회복 기능을 제공하고, 모든 데이타들은 기록 데이타로서 시차적으로 유지되어야 하고, 다양한 형태의 표현방법과 계층적 데이타 집체를 제공하여야 한다.

버전관리란 시차적으로 변하는 공학설계 데이타를 계층적인 집합체로 배열하는데 사용되는 조직 구성상의 개념들과 연산 메카니즘들의 집합을 말한다. 버전 모델은 기초적인 데이타베이스 설비위에서 공학적 데이타 조직을 지원하기 위해서 의미상의 확장을 제공하는 현행 데이타 조직에 접목된다.

설계 객체는 설계자에 의해서 응집된 단위로 취급되는 설계 데이타들의 집합체를 말하고, 그것들이 시차적으로 의미있는 단면을 버전이라 한다. 버전 모델링은 개개의 버전들을 버전 이력표로 조직하기 위한 데이타 조직 개념들을 제공하고, 그들의 구성요소 버전들로 부터 복합 객체를 구성하고, 표현 형태에 따른 동등 버전들을 추적할 수 있도록 한다. 버전 관련 연산의 하나인 상속이란 새로운 버전이 만들어 질 때 전신으로 부터 데이터를 얻을 수 있는 메카니즘을 말하고, 변경 전파란 복잡하게 상호 연관된 설계 객체 구조가 데이타 변경에 어떻게 반응하는지를 서술하는 것을 말하고, 작업 공간 모델은 새로운 모델들이 설계자들의 공동체에 어떻게 보여지는지를 서술하는 메카니즘을 말한다.

디자인 시스템은 소프트웨어의 집합체로서 설계객체를 만들고 합성하고, 정확성을 검증하기 위해서 분석하고 설계 데이타의 저장과 조직을 관리하고, 정확히 설계된 제품을 만들기 위해서 설계 응용들의 조절된 순차과정인 설계 흐름의 과정들을 관리한다. 여기서 설계 데이타를 만들

고 사용하는 설계 응용과 설계 데이타를 조직하고 설계 응용이 필요에 따라서 설계 데이타를 접근하도록 인터페이스를 제공하는 설계 관리의 차이점을 분명히 해둘 필요가 있다. 설계 데이타베이스 시스템은 설계 관리 소프트웨어의 한 부분으로서 설계 데이타의 저장, 검색, 그리고 일관된 수정을 다룬다. 설계 관리 소프트웨어는 설계 DB에 있는 데이타가 어떻게 설계를 표현하는지를 해석하는 일을 한다.

설계 관리의 중요한 요소로서 작업 공간과 설계 객체가 있다. 작업 공간은 설계 객체들의 저장소로서 사용자와 설계 응용이 그 객체들을 접근할 수 있고, 사용자들 사이에 설계 관리 시스템을 공유하는 단위가 된다. 설계 객체는 설계 기본 요소들의 집합체이다. 버전은 일정한 시점에서 설계 객체의 의미있는 단면을 의미한다. 따라서 버전은 현존하는 버전의 후속자일 수도 있고, 다른 버전의 전자일 수도 있다. 모든 변경이 신 버전을 만드는 것은 아니다. 신 버전은 설계 과정의 한 부분으로서 만들어지는 것이고, 변경이 수행되고, 변경된 객체가 검증되고, 이 변경이 수용된다면 신 버전이 만들어지고 출시되는 것이다. 설계 객체는 구성요소 객체들의 계층적 집합체이고, 상호 구성 배치(configuration)란 복합 객체의 한 버전과 그것의 각 구성 요소들의 버전들 사이의 바인딩(binding)을 의미한다.

그림 1 은 설계 관리[1]의 계층적 구조를 나타내고 있다.

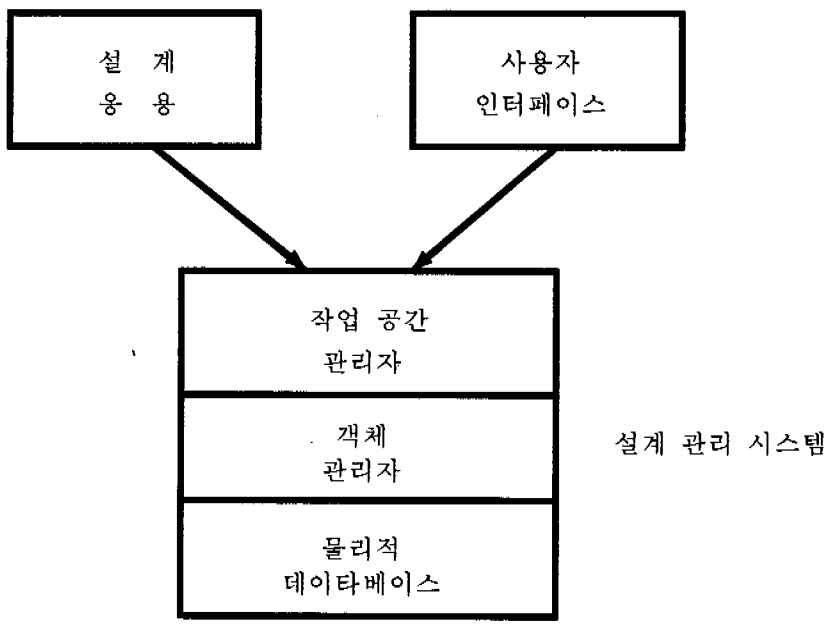


그림 1. 설계 관리의 구조도

그림 1 에서 물리적 데이타베이스는 데이타를 저장하고, 데이타를 원자적으로 수정하며, 시스템 장애시에도 일관된 상태를 유지하도록 한다. 그 위에 객체 관리자가 있는데, 그것은 설계 객체들의 고도로 상호 연결된 네트워크를 구현한다. 그 위에 작업 공간 관리자가 있는데, 그것은 설계 객체들을 수용하는 공유된 작업 영역을 제공한다.

공학설계 데이타베이스 버전 모델이 갖추어야 할 기본적인 모델링 개념과 연산은 다음과 같다.

- 설계 기본자들은 설계 객체라 불리는 의미있고 이름을 갖는 단위로 집체되어야 하고, 설계 객체는 설계 기본자들의 집합체이다.
- 설계 객체들은 보다 기본적인 설계 객체들로 부터 계층적으로 구성된다.
- 설계 객체의 내용물은 설계 기본자들의 모임이고, 인터페이스는 다른 설계 객체와의 입출력 및 작용을 기술하는 것이다.
- 상호 작용과 인터페이스로 기술되는 설계 객체의 상호 구성 배치를 표현하고, 구성하고, 관리하는 메카니즘이 제공되어야 하고, 상호 구성 배치란 설계 객체의 버전이 보다 기본적인 구성 객체들의 버전들로 부터 구성된다는 것을 의미한다.
- 다중 버전 설계 객체를 규정하고 관리하는 메카니즘이 제공된다.
- 설계 객체의 특정 버전을 선택해서 다른 상호 구성 배치에 참여시키는 기능이 있다.
- 전에 정의된 설계 객체가 새로운 설계 객체를 정의하는데 재사용될 수 있고, 객체의 정의와 실재사이에는 분명한 구분이 있다.
- 미래에 리뷰하기 위해서 설계 객체를 기록하는 이력 관리 메카니즘이 제공된다.

본 논문은 다음과 같이 구성되어 있다. 2절에서 버전 모델의 실례를 제시해서 버전 모델의 이해를 촉구했고, 3절에서 새로운 버전 모델 및 관리 개념을 제시해서 버전 모델 연구의 기초를 제공했다.

## 2. 버전 모델의 실례

여기서는 CAD 데이타를 위한 버전 서버[2,3,4]에서 구현된 특정한 버전 모델에 대해서 상세한 서술을 하겠다.

### 2.1 모델링 기본자 (Modelling Primitives)

#### 2.1.1 배경

버전 서버 모델의 기본사항은 설계 객체와 그들 사이의 관계이다. 이 모델은 is-a-kind-of 관계와 is-a-part-of 관계를 포함하고 있고, 노드는 객체를 표현하고, 아크는 관계를 표현하도록 해서 도식적으로 개념을 설명한다. 설계 객체들의 공간은 설계 응용에 의해서 만들어 지고

사용되는 설계 데이타들의 파일들에 해당된다. 설계 모델의 목적은 이 공간에 유용한 계층적 구조를 부과하는 것이다. 이것은 특수한 조직 객체와 관계를 도입함으로써 이루어지고, 이런 객체를 구조 객체라 한다. 구조 객체는 설계 객체들을 버전 이력도와 상호 구성 배치로 조직하는데 사용된다. 이 모델에는 설계 객체들을 계층적 그룹핑으로 조직하는 세가지의 구조적 관계가 있는데, 구성요소 계층, 버전 이력도, 동등관계가 그것들이다.

2.1.2 구성요소 계층(IS-A-PART-OF)

설계 객체는 기본자일 수도 있고, 복합자일 수도 있다. 기본자 설계 객체는 구성 요소 계층 목구조의 잎부분에 위치하고, 복합 객체는 부 그래프의 근에 위치한다. 그림 2 는 구성요소 계층의 한 부분을 나타내고 있다.

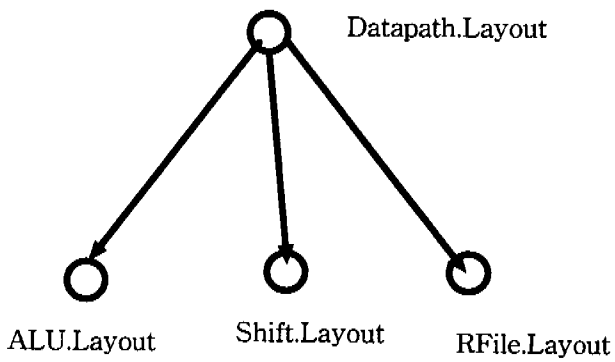


그림 2. 구성요소 계층도

그림에서 구성요소 객체와 복합 객체와의 관계는 is-a-part-of 관계이다. 이 그림은 구성요소 계층도가 목구조로 되어 있는데, 유향 비순환 그래프(directed acyclic graphs)로 표현하는 것이 더 일반적이다.

2.1.3 버전 이력도(IS-DERIVED-FROM)

객체는 이름[버전번호].타입 형태로 표시할 수 있다. 그러나 이것으로 어떤 버전이 어떤 버전으로 부터 전수 받았는지는 알 수가 없다. 따라서 버전 이력도라는 개념이 필요하다. 버전 이력도는 is-derived-from이라는 관계를 통해서 전수자/전수받는자의 상호 관계를 외양적으로 기록할 수가 있다. 그림 3 에 버전 이력도의 예가 있다.

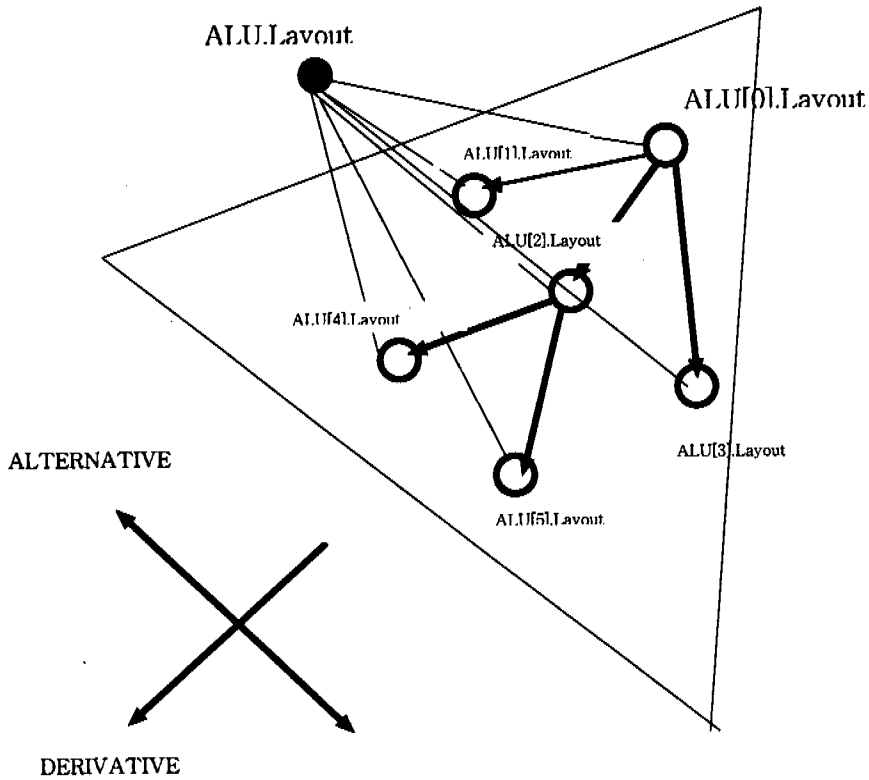


그림 3. 버전 이력도

그림 3 에서 ALU[2].Layout는 ALU[0].Layout로 부터 유도되었다. ALU[1], ALU[2], ALU[3]는 선택사항으로서 평행 버전이라고 한다. 버전 이력도는 목구조 형태를 하고 있다. ALU.Layout는 구조 객체로서, ALU[i].Layout는 ALU.Layout와 is-a-kind-of 의 관계를 갖고 있고, 타입-실재의 관계로 볼 수 있다. 다른 연구자는 버전은 타입에 대한 실재라고 제안하였다.[5]

2.1.4 상호 구성 배치(configurations)

버전 이력도와 구성요소 계층도가 결합되면, 상호 구성 배치라는 개념이 생긴다. 이것은 복합 객체의 버전은 그 구성요소 객체들의 특정한 버전들로 부터 구성된다는 사실에서 알 수 있다. 이 모델은 계층적 상호 구성 배치라는 개념을 지원한다. 그림 4 에 상호 구성 배치의 예가 있다.

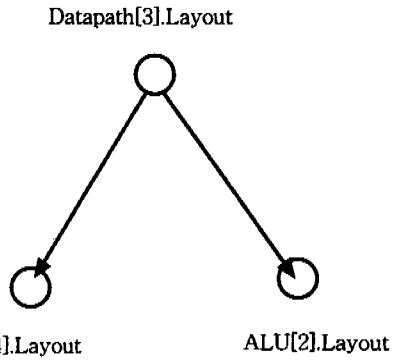


그림 4. 상호 구성 배치의 예

그림 4에서 Datapath[3].Layout는 RFile[4].Layout과 ALU[2].Layout로 구성되었음을 보여 주고 있다.

2.1.5 동등 관계(Equivalencies)

설계 제품을 완전히 기술하기 위해서는 다양한 표현이 필요하기 때문에, 설계 데이타는 다양한 면모를 갖는다고 말한다. 각각의 표현 형태나 뷰는 그 자체가 설계 객체이긴 하나, 이러한 것을 묶어줄 기본자가 필요하게 된다. 이 모델에서는 구조 객체인 동등 관계를 통해서 구현하고 있다. 그림 5에 동등 관계의 예가 표시되어 있다.

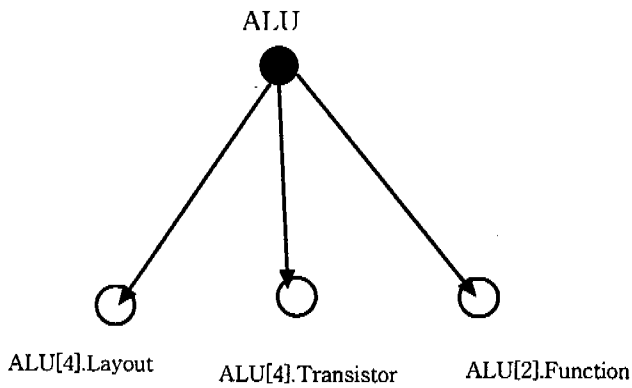


그림 5. 동등 관계의 예

그림 5에서 동등 관계 객체 ALU는 ALU에 대한 각각의 표현을 묶어 주고 있다. 동등 관계

란 표현 형태는 다르지만 동일한 현실 세계의 사물을 나타내는 것들 간의 관계를 의미한다.

## 2.2 연산(Operations)

모든 설계 객체는 읽어지고 쓰여져야 한다. 여기서는 설계 관리에 관한 연산에 대해서 서술한다. 이런 연산에는 버전 이력도내에서 현재 버전을 확인하는 연산, 유동적 상호 구성 배치를 서술하는 연산, 작업 공간들 사이에 객체들의 이동을 관리하는 연산, 변경 전파를 지원하고, 관련된 설계 객체들로 부터 속성을 상속받는 연산등이 있다.

### 2.2.1 현행 버전 확인 연산

가장 최근에 만들어진 버전을 현행 버전이라고 말할 수는 없다. 버전 서버 모델에서 신 버전은 항상 현행 버전의 전수받는자로 제한했다. 현행 버전의 예가 그림 6에 나타나 있다.

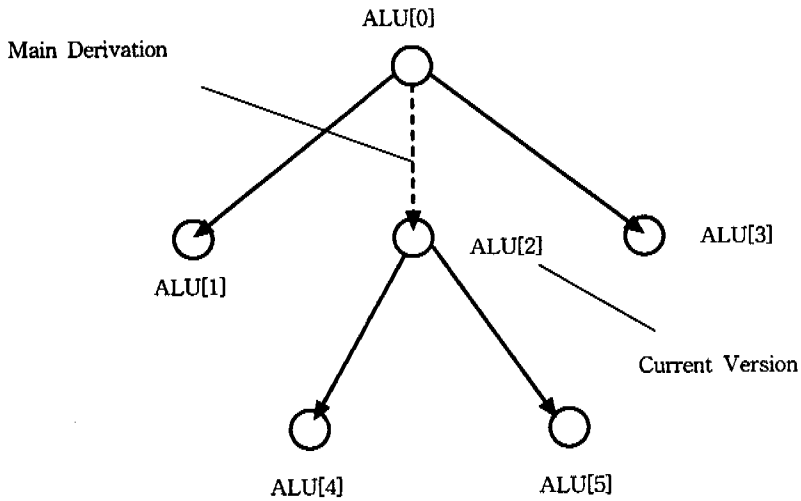


그림 6. 현행 버전의 예

그림 6 에서 ALU[2]가 현행 버전으로 구별되었다. 주 유도경로는 버전 이력도의 근으로 부터 현행 버전에 이르는 경로를 말한다. 현행 버전 규칙에 의하면 ALU[0], ALU[1], ALU[3]으로 부터 현행 버전의 재지정 없이 새로운 유도자가 나올 수 없다. 물론 ALU[4], ALU[5], ALU[2]로 부터는 새로운 유도자가 나올 수 있다. 버전 이력도내에서 어디에 있던지 현행 버전을 외양적으로 위치시키는 연산이 제공된다. 현행 버전과 관련하여 특정 프로젝트 상황에서 현행 버전 지정을 어떻게 운영할 것인지를 결정하는 정책이 있다. 그런 정책에는 누가 현행 버전 재지정을 수행할 것인가, 얼마나 많은 현행 버전이 동시에 활동할 것인가, 현행 버전이 이전 버전으로 역행할 수 있는지 또는 앞으로만 나아가야 하는지 등에 관한 것들이 있다.

2.2.2 유동적 상호 구성 배치(Dynamic Configurations)

is-a-part-of 관계가 실제적으로 섭렵될 때 까지 상호 구성 배치에서 구성 요소에 대한 참조가 결정되지 않도록 하는 유동적 상호 구성 배치를 만드는 것이 필요하다. 버전 서버 모델에서는 Xerox PARC [6]에서 개발된 PIE 시스템에서 사용한 계층과 상황 접근 방법에 기초한 메카니즘을 제안하고 있다.

기본 생각은 버전 이력도에 있는 관련된 객체들을 같은 계층에 넣고, 계층들의 순서를 지정하는 상황을 제공하는 것이다. 서로 다른 객체들의 두 버전이 같이 사용된다면 같은 계층에 넣는다. 상황은 계층들에 대한 탐색 순서를 정의한다. 유동적 상호 구성 배치에 관한 예가 그림 7에 있다.

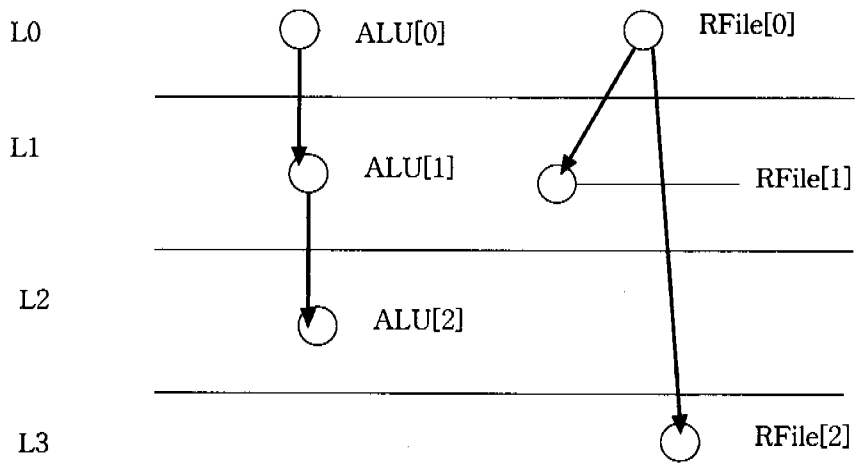


그림 7. 유동적 상호 구성 배치의 예

그림 7 에서 ALU[0]와 RFile[0]가 같이 사용되고, ALU[1]과 RFile[1]이 같이 사용된다.

만약 현행 상황에서 계층 탐색 순서가 L3/L2/L1/L0로 되어 있다면, ALU에 대한 참조는 ALU[2]로 결정되고, RFile에 대한 참조는 RFile[2]로 결정된다.

2.2.3 작업 공간(Workspaces)

작업 공간은 설계 객체들의 저장소로서 이름을 갖는다. 작업 공간은 공적 보관소, 그룹 보관소, 사적 보관소로 나눌 수 있다. 공적 보관소는 모든 사람에 의해서 읽을 수 있고, 추가할 수 있다. 그러나 공증된 객체만이 공적 보관소에 저장할 수 있다. 사적 보관소는 특정한 설계자에게 속해 있고, 소유자만이 그 내용을 읽을 수 있고, 추가할 수 있다. 사적 보관소에는 미완성의 진행중인 설계 객체가 보관된다. 그룹 보관소는 두 명 이상의 설계자들이 진행중인 설계 객체들을 보관하고, 결합이 가능할 때 까지 보관한다.

그 그룹에 속하는 요원은 누구든지 그룹 보관소의 내용을 읽을 수 있고, 추가할 수 있다. 그림 8에 작업공간의 예가 있다.

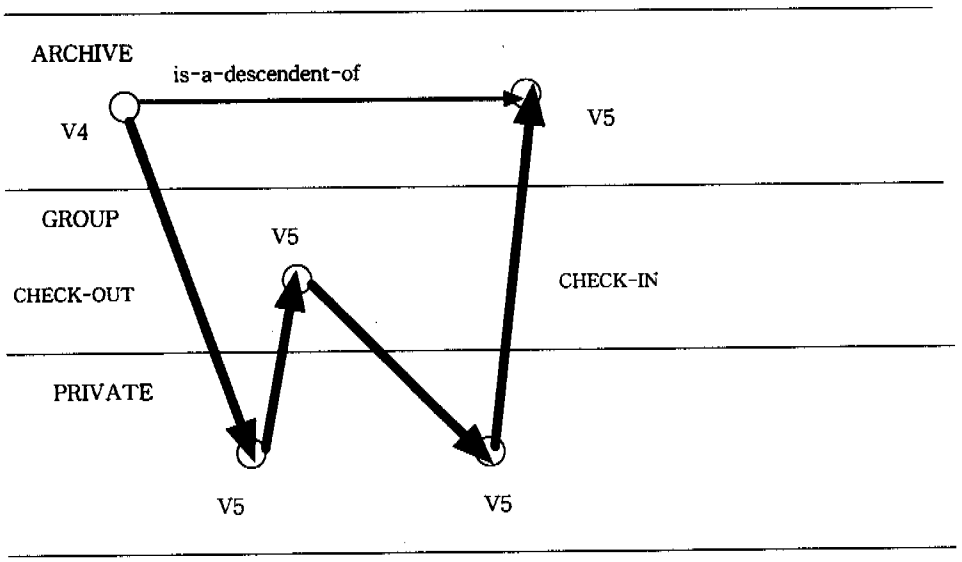


그림 8. 작업 공간의 예

그림 8에서 V4가 V5로 진화하는 과정을 도시하고 있다. 즉 V4는 사적 보관소로 체크아웃된 다음, 수정을 해서 V5가 되고, 그것을 그룹 보관소에서 다른 객체와의 결합을 체크한 다음 문제점이 있어서 다시 사적 보관소에 이송된 다음, 수정을 해서 공적 보관소로 체크인되는 과정을 도시하고 있다.

2.2.4 논리적 표현과 물리적 표현

버전 서버 모델이 설계 모델을 물리적으로 구현하는 기본 생각은 링크드 리스트 구조에 의해서 상호 관련 망을 표현하는 것이다. 이 모델의 표현 객체는 실제의 설계물을 갖고 있는 유닉스 파일들을 가르키는 레코드에 의해서 구현된다. 내부적으로 설계 객체는 상호 구성 배치 객체에 의해서 표현되고, 그것에서 설계 파일들을 참조하고, 버전 링크에 의해서 전수받는 자의 연관된 상호 구성 배치 객체와 구성 요소링크에 의해서 구성 요소들에 대한 포인터를 참조한다. 논리적 표현은 버전 이력도, 구성 요소 계층도, 동등 관계도로 구성된다. 물리적 표현은 설계 객체들간의 논리적 연결 관계를 링크드 리스트 구조로 구현한다.

2.2.5 변경과 제한사항의 전파(Change and Constraint Propagation)

변경 전파란 신 버전들을 자동적으로 상호 구성 배치에 도입하는 과정을 말한다. 제한사항 전파는 신 버전들을 절차적으로 재생성함으로써 동등 관계 제한사항을 이행하는 것을 말한다.[7]

제한사항 전파가 지원되기 위해선, 동등 관계에서 방향이 있어야 한다. 즉 한 표현형태의 객체는 다른 동등 관계 표현으로 부터 유도될 수 있어야 한다. 변경 전파에 관한 두가지 이슈가

있는데, 하나는 전파의 범위를 어떻게 제한할 것인가이고, 다른 하나는 변경 경로를 어떻게 분명히 할 것인가에 관한 것이다. 변경 경로의 분명성에 관한 예가 그림 9에 있다.

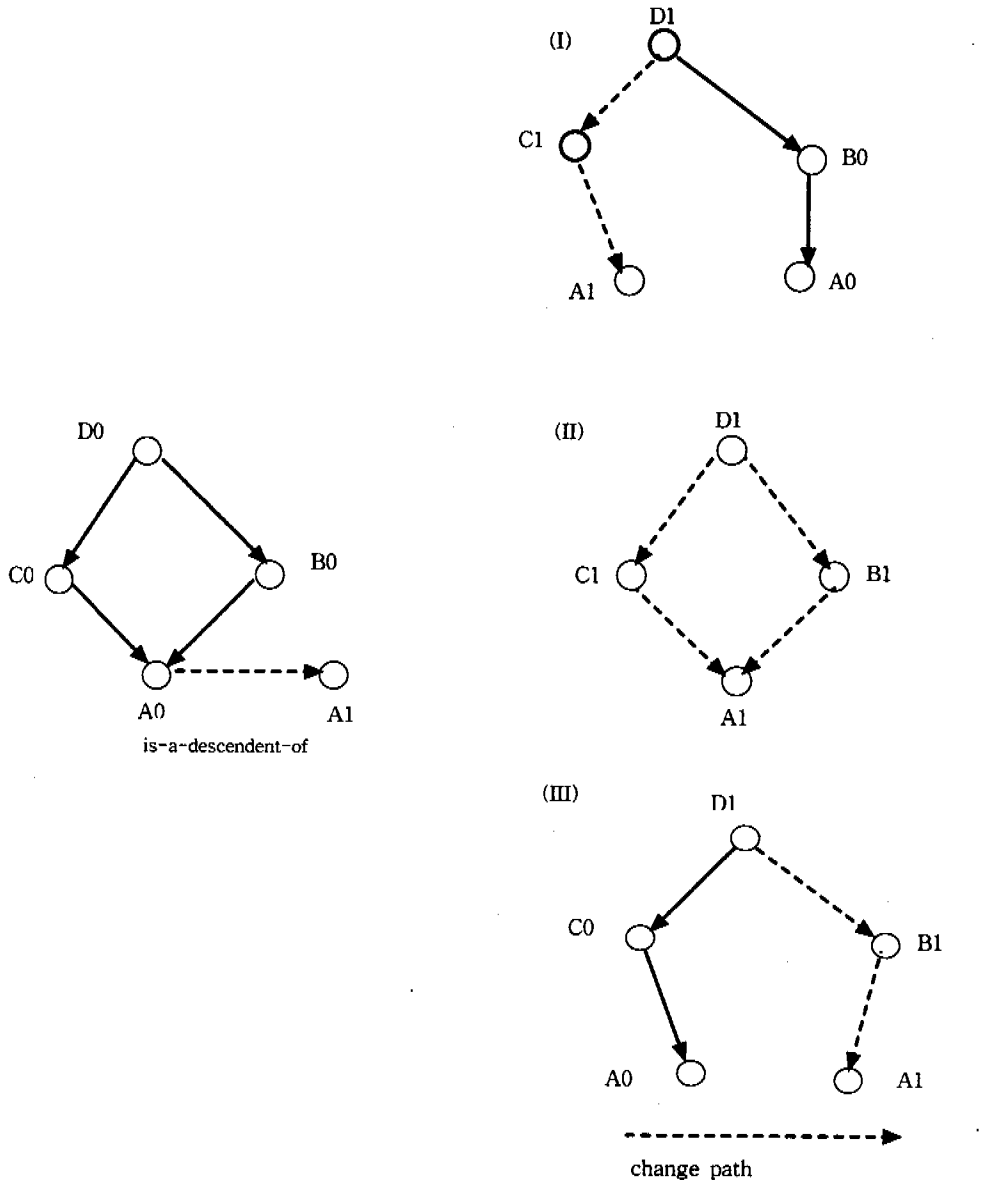


그림 9. 분명한 변경 전파의 예

그림 9에서 왼편의 상호 구성 배치도에 A1을 삽입함으로써 생기는 세가지의 가능한 새로운 상호 구성 배치도를 보여 주고 있다. (I) 은 변경 전파 경로가 A에서 C로, C에서 D로 가는 것이고, (III)은 변경 전파 경로가 A에서 B로, B에서 D로 가는 것이고, (II)는 전파 경로가 양쪽 경로에 전부 적용된 것이다.

2.2.6 상속 관계(Inheritance)

상속 관계는 어떤 객체 실재가 그 자체의 데이타나 연산을 정의하지 않았다면 그것의 상위 타잎에 따르는 것을 말한다. 그리고 상속 관계는 신 버전들에 대해서 디폴트 연산이나 데이타를 정의할 수 있는 메카니즘을 자연스럽게 제공한다. 바토리와 김[5]은 상속 관계를 버전을 모델링 하는데 주요한 메카니즘으로 채택했다. 버전들은 어떤 타잎의 실재들이고, 버전 생성시 인터페이스 기술같은 그 타잎의 양상들을 상속받게 된다. 이에 대한 예는 그림 10에 도시되어 있다.

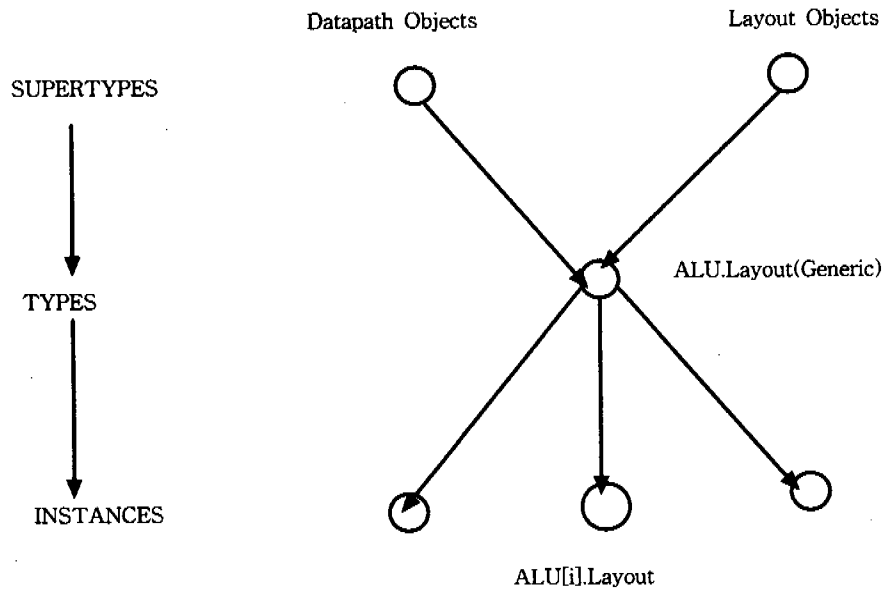


그림 10. 상속 관계의 예

# 3. 제시된 버전 모델

이 절에서는 선행 연구와 문헌 연구의 결과 및 기존의 버전 모델들을 기초로 해서 새로운 버전 모델을 제시하고자 한다.

## 3.1 버전 집합체 구성(Organizing the Version Set)

여기서 제시된 주요한 개념은 버전 이력도, 총칭 객체, 전후자 관계, 주 유도, 분기, 현행 버전등이다. 버전 실재는 실재하는 객체로서 시스템에서 유일하게 식별되고, 시차 기록을 갖는 유일한 버전 번호를 갖는다.

버전 실재는 공통의 총칭 객체와 연관되고, 그들로 부터 속성과 기본값을 상속받을 수 있다. 버전 실재들은 그들간에 전후자 관계로 상호 연관을 갖는다. 전후자 관계는 버전 이력도에 순서를 주기 위해서 사용된다.

하나의 버전 실재가 현행 버전으로 인식되고, 최근에 만든 버전이 항상 현행 버전이 되는 것은 아니다. 목구조의 근으로 부터 현행 버전에 이르는 경로를 주 유도 경로라 부른다. 버전 이력도에서 대체 버전들을 분기라 부르고, 본 논문에서 대체 버전은 원칙적으로 하나만 존재하는 것으로 하고, 그것은 나중에 병합되는 것을 전제로 한다. 만약 병합이 안되면, 버전 이력도에서 삭제하는 것으로 한다. 따라서 버전 이력도는 목 구조라기 보다, 유향 그래프로 생각할 수 있다. 제시된 버전 이력도의 예가 그림 11에 있다.

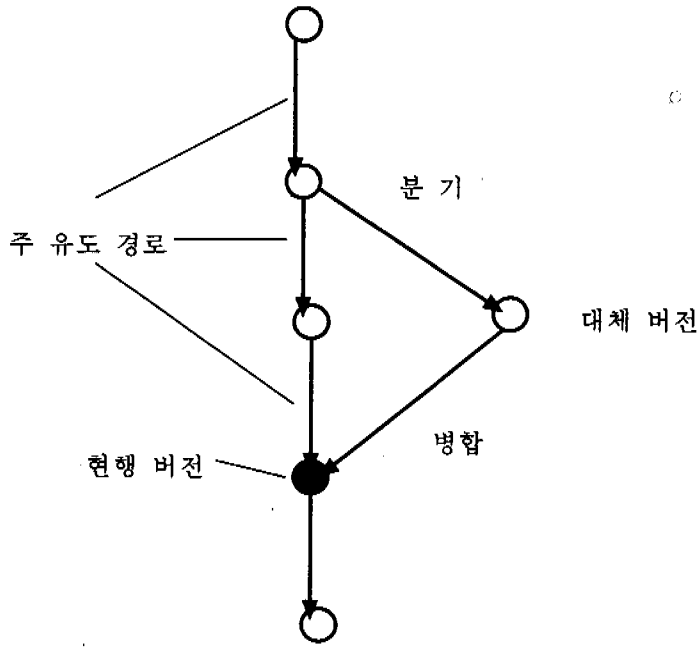


그림 11. 버전 이력도의 예

## 3.2 동적 상호 구성 배치 방법(Dynamic Configuration Mechanism)

정적 참조는 특정한 버전의 참조를 말하고, 동적 참조는 총칭 객체를 지시하며, 특정한 버전에의 참조는 연기되어서 나중에 바인드된다. 동적 상호 구성 배치 방법을 효율적으로 수행하기 위해서 동적 참조가 필요하며, 동적 참조를 원활히 수행하기 위해서 총칭 참조와 참조 환경이라는 개념을 제시한다. 총칭 참조는 사용되는 버전과는 독립적으로 설계 객체에의 참조를 말한다. 참조 환경은 이름을 가진 이진 관계로서 (객체 번호, 버전 번호) 쌍의 집합과 기본 옵션으로 되어 있다. (객체 번호, 버전 번호) 쌍의 뜻은 객체 번호에 해당하는 객체가 참조되었을 때, 해당하는 버전이 사용된다는 것이고, 그런 쌍이 존재하지 않을 때는 기본 옵션이 사용된다는 뜻이다. 참조 환경은 언제든지 생성될 수 있고, 데이타베이스에 저장된다. 참조 관계는 참조 환경이 기동될 때만 유효하게 된다. 참조 환경을 기동하는 명령이 제공되고, 특정 시점에서는 하나의 참조 환경이 기동될 수 있다. 참조 환경의 뷰에는 정의 뷰와 실행 뷰가 있는데, 앞에서 말한 (객체 번호, 버전 번호) 쌍의 집합과 기본 옵션은 실행 뷰에 해당하고, 정의 뷰는 사용자가 참조 환경의 생성 과정을 서술한 것으로서, 실행 뷰는 정의 뷰가 기동되어서 만들어 진다. 참조

환경의 정의 뷰는 이름, 기본 옵션, 직접 참조 엔트리의 집합, 간접 참조 엔트리의 집합으로 구성된다. 직접 참조 엔트리는 (객체 번호, 버전 번호) 쌍으로 구성되며, 객체 번호에 해당하는 객체가 참조되면 해당하는 버전 번호의 버전이 참조됨을 의미한다. 간접 참조 엔트리는 (객체 번호, 참조 환경 번호) 쌍으로 구성되며, 객체 번호에 해당하는 객체가 참조되었을 때 해당하는 참조 환경 번호의 참조 환경에서 재귀적으로 탐색하도록 한다.

### 3.3 버전 그룹핑 방법(Version-Grouping Mechanism)

그림 11에서 제시한 분기, 버전 이력도는 버전의 일반적인 전개도를 나타내고 있다. 설계 객체의 현실적인 면을 고려하면, 버전들의 모임에 의미나, 논리성을 부여하기 위한 버전 그룹핑 또는 버전 그룹핑의 그룹핑을 생각할 수 있다. 버전 그룹핑은 유향 그래프 관계로 형성되며, 버전들을 문제 해결에 적합한 구조로 구성한다.

여기에서 버전 그룹핑 타입이 필요하고, 버전 그룹핑 실재는 그 타입에 맞추어서 생성된다. 버전 그룹핑에 관한 연산으로는 버전 그룹핑 타입을 정의하는 것, 버전 그룹핑 실재를 생성하고 삭제하는 것, 버전 그룹핑 구조를 탐색해서 버전 그룹핑과 버전들을 검색하는 것등이 있다. 그리고 버전 그룹핑과 총칭 참조 및 참조 환경을 결합하면 상승 효과를 얻을 수 있는데, 참조 환경의 정의에서 (객체 번호, 버전 번호) 쌍을 확장해서 (객체 번호, 버전 그룹핑.버전 번호)로 하면 참조 환경의 실행 뷰를 생성하는 데 효율적으로 할 수 있다.

### 3.4 변경 통지와 전파(Change Notification and Propagation)

객체 버전은 다른 객체들의 여러 버전들을 참조할 수 있다. 참조된 버전들이 변경되었을 때, 그 변경 사항을 참조하는 객체에게 통지할 필요가 있다. 이것을 변경 통지와 전파라 한다. 변경 통지 방법에는 메시지 방법과 플랙 방법이 있고, 메시지 방법은 시스템이 영향을 받는 버전의 사용자에게 통보하는 것으로서 즉시 통보와 지연 통보의 두가지 방법이 있다. 플랙 방법은 변경된 버전의 자료구조에 변경 여부를 기록하는 방법으로서 사용자는 그 버전을 접근했을 때 변경 여부를 알 수 있다. 버전 참조는 매우 복잡하게 연관되기 때문에 통보 범위를 제한하는 것이 필요하다. 그래서 통보 범위에 두가지 가능성이 있는데, 하나는 변경된 버전을 직접적으로 참조하는 버전들에만 통보하는 것이고, 다른 하나는 변경된 버전을 직접적 또는 간접적으로 참조하는 모든 버전들에게 통보하는 것이다. 여기서는 메시지 방법에 대한 메카니즘을 제시하고, 통보 범위는 직접적으로 참조하는 버전에게 통보하고, 그 통보받은 버전이 변경되면, 다시 직접적으로 참조하는 버전에 통보하는 방식으로 메카니즘을 제시하겠다. 변경 통보의 메시지 방법을 구현하기 위해서 버전 V를 참조하는 모든 버전들의 리스트인 역 참조 리스트를 유지해서, 버전 V가 변경되는 경우 통보해 주어야 한다. 역 참조 리스트에는 참조되는 버전 이름, 참조하는 버전 이름, 통보 방법, 변경 형태 등을 레코드로 기록 유지하여야 한다.

### 3.5 설계 버전 객체 공유 방법(Design Version Object-Sharing Mechanism)

버전들은 임시 버전, 작업 버전, 출시 버전등 세가지로 구분한다. 임시 버전은 사용 데이타베이스에 저장하고, 작업 버전은 프로젝트 데이타베이스에 저장하고, 출시 버전은 공용 데이타베이스에 저

장한다. 임시 버전은 설계자 개인별의 작업 대상이고, 작업 버전은 프로젝트 팀별의 작업 대상이고, 출시 버전은 검증된 버전을 말한다. 공용 데이타베이스에서 프로젝트 데이타베이스나 사용 데이타베이스로 버전을 보내는 것이나, 프로젝트 데이타베이스에서 사용 데이타베이스로 버전을 보내는 것을 체아웃이라 하고, 사용 데이타베이스에서 프로젝트 데이타베이스 또는 공용 데이타베이스로, 또는 프로젝트 데이타베이스에서 공용 데이타베이스로 버전을 보내는 것을 체인이라 한다.

## 4. 결 론

본 논문에서 버전 모델링과 관리에 관한 기초적인 사항을 서술했고, 버전 모델이 갖추어야 할 요구 조건에 대해서 기술했다. 버전 모델의 실례를 통하여 버전 모델의 여러 양상에 대해서 실감할 수 있도록 했으며, 버전 모델 설계 기법을 터득할 수 있도록 했다. 아울러 새로운 버전 모델의 프레임워크를 제시해서 버전 모델 및 관리에 관한 연구에 기초를 다졌다. 여러 사람이 버전 모델링과 관리에 관한 모델을 제시했는데, 그들의 제안이 표현면에서는 다소 상이하지만 내용 및 개념면에서는 동등한 내용들이 많기 때문에 그것들을 통합하고 용어를 통일하는 것은 연구의 중요한 요소라 생각한다. 앞으로의 연구 과제는 어떤 환경에서도 적용할 수 있는 일반적인 버전 모델을 정립하는 것이다. 이것은 공개된 연구 과제로서 많은 노력이 필요하리라 생각한다.

## 참 고 문 헌


1. Katz, R. H., Information Management for Engineering Design, Springer-Verlag Computer Science Survey Series, Heidelberg, West Germany, 1985.
2. Katz, R. H., Chang, E., and Bhateja, R., Version Modelling concepts for Computer-Aided Design Databases, In Proceedings of the ACM SIGMOD Conference, pp. 379-386, Washington, DC, May 1986.
3. Katz, R. H., Anwarrudin, M., and Chang, E., A Version Server for Computer-Aided Design Data, In Proceedings of the 23rd ACM/IEEE Design Automation Conference, Las Vegas, Nev., June 1986.
4. Katz, R. H., Bhateja, R., Chang, E., Gedye, D., and Trijanto, V., Design Version Management, IEEE Design and Test Vol. 4, No. 1, pp. 12-22, 1987.
5. Batory, D., and Kim, W., Modelling Concepts for LSI CAD objects, ACM Trans. Database Syst., Vol. 10, No. 3, pp. 322-346, 1985.
6. Goldstein, I., and Bobrow, D., Layered Networks as a Tool for Software Development, Proceedings of the 7th International Conference on Artificial Inteligence, pp. 913-919, Aug. 1981.
7. Katz, R. H., and Chang, E., Managing Change in a Computer-Aided Design Database, Proceedings of the 13th Conference on VLDB, pp. 455-462, Brighton, England, Sept. 1988.