# 객체지향적 스키마의 버전제어에 관한 연구

김영실* · 김창화** · 백두권*

* 고려대학교 전산과학과

** 강릉대학교 전자계산학과

## 요 약

스키마 변화처리시 기존의 스키마와 새로운 스키마의 관계는 대부분의 경우 서로 공통적으로 가지는 부분들과 그렇지 않은 부분들로 나눌 수 있다. 새로운 스키마는 서로 다른 부분의 구조에 대한 내용만을 기존의 구조에 추가하면 될 것이다. 이러한 성질을 객체지향 개념을 이용하여 표현함으로써 스키마의 변화 관리를 더욱 효율적으로 하는 방법을 제시하고 시스템을 설계했다. 본 연구에서 정의된 스키마 구조는 atomic type들로 이루어진 클래스들로 구성되어 있어서 변경된 내용은 스키마 전체의 변경을 요구하지 않고 이러한 타입이나 클래스의 변경에 의해서 이루어져 스키마 버전의 관리가 용이하며 효율적이다.

## 1. 서론

컴퓨터의 발전과 더불어 많은 양의 데이타를 처리하기 위한 데이타 베이스 시스템이 등장했다. 이러한 시스템들은 많은 기관들이나 조직체에서 요구하는 정보를 처리하는데에는 한계점에 이르렀다[1]. 사용자(일반 사용자, 분석가, 시스템 관리자)들은 따른 시스템내의 데이타뿐 아니라 스키마의 변화도 요구한다. 시간이 흐름에 따라 더욱 다양해 지는 사용자의 요구를 만족시키기 위해서 스키마 버전과 그 정보들에 대한 관리가 중요하게 되었다. 그러나 기존의 데이타 베이스 시스템에서 스키마의 변경은 그 변경내용이 부분적이라도 전체 스키마 구조를 변경해야 한다. 부분적인 스키마 변화에 대한 처리를 위해서 전체가 아닌 부분적인 변경에 의해 변경된 스키마 구조를 지원하는 방법이 필요하다.

객체지향 개념은 데이타 추상화, 계승(generalization / specialization), complex object 등을 좀 더 강력하게 표현하는 방법이며, 프로그래밍의 생산성을 높이고 유지(maintenance)하는데 용이하다[6,8,10]. 스키마 변화는 대부분이 기존 스키마의 일부분이 수정되는 것이므로 이전의 스키마와 변경된 스키마간에는 서로 상속(inheritance 또는 같은 구성요소)되는 부분이 많기

때문에 객체지향 개념을 사용하여 시간의 흐름에 따라 변화하는 스키마의 버전을 관리할 수 있다.

본 연구에서는 객체지향 개념을 이용하여 스키마 구조를 정의하고, 이 구조에 따른 버전의 종류, 버전정보를 위한 저장구조를 제안한다. 이러한 저장구조를 이용하여 버전을 제어하는 시스템 VECS를 설계한다. 스키마는 정의한 atomic type들로 이루어진 클래스들로 구성하며, 스키마의 변경요구가 들어오면 전체 구조를 바꾸는 것이 아니라 변경에 해당되는 클래스의 내용이나 구조 또는 atomic type들만을 변경함므로써 스키마 변경을 효율적으로 한다. 또한 이 스키마 제어 방법을 한조직체의 정보를 효율적으로 관리하고자 하는 정보자원 사전 시스템(Information Resource Dictionary System : IRDS)의 스키마 버전제어에 응용한다. 이 외에도 VECS 시스템은 경영정보 시스템, 사무정보 시스템, CAD/CAM 시스템등에도 응용할 수 있다.

2장에서는 IRDS의 기본개념과 구조에 대해서 다루고, 3장에서는 객체지향 개념을 이용한 스키마 정의를 다룬다.

## 2. IRDS 모델과 버전개념

아직까지 우리에게 정보를 자원으로 생각한다는 것이 낯설기만하나 더 많은 컴퓨터와 소프트웨어의 보급, 정보들의 관리와 유지는 더욱 그 중요성이 높아지고 있다. 데이타 사전이란 데이타에 관한 정보, 데이타의 정의와 통제, 그리고 데이타의 구조 또는 표현이 변경될 때 어떤 프로그램이 영향을 받는가를 나타내 준다. 데이타 사전의 가장 중요한 목적은 데이타 표현을 표준화하고 정의하여 다 같이 사용하도록 하는 것이다.

IRDS는 데이타 표현의 공통성, 데이타 정의에 관한 합의 유도, 신뢰할 수 있는 문서화를 위한 기반, 시스템 개발 비용과 준비시간의 절감, 데이타 표준에 준거, 중복(redundancy)의 제거, 데이타 분석 작업의 경감, 동의어의 식별 등의 데이타 사전의 장점들을 갖게 될 것이다[18].

1980년대 NBS(National Bureau of Standrds)와 ANSI(American National Standards Institutes)에서 사전시스템(dictionary system)에 대한 표준화 작업을 시작했으며, 1983년에 이르러서 그 노력이 나타나기 시작했다. ISO에서는 ANSI IRDS를 받아들이지 않고 나름대로 개발하고 있다. 따라서 현재 IRDS는 ISO와 ANSI의 두 표준기관을 중심으로 이루어져가고 있다. 제안된 IRDS 표준은 공통의 인터페이스를 갖는다[11,16,17].

한 조직체의 정보자원을 효율적으로 관리하는 시스템인 IRDS는 ISO에서 제안한 relational model based 모델과 ANSI에서 제안한 ER model based 모델이 있다.

IRD(Information Resource Dictionary)는 조직체에 관련된 정보자원을 정의하기 위한 공유가능한 저장소이며, IRD 정의(혹은 사전정의(dictionary definition))는 IRD 스키마의 부분이 아닌 데이타 또는 과거나 미래에 IRD 스키마와 관련있는 데이타를 포함해야 한다. 따라서 IRDS는 IRD 및 IRD 정의를 생성, 관리, 접근하기위한 기능들을 제공해주는 시스템이다[3,7,12,16].

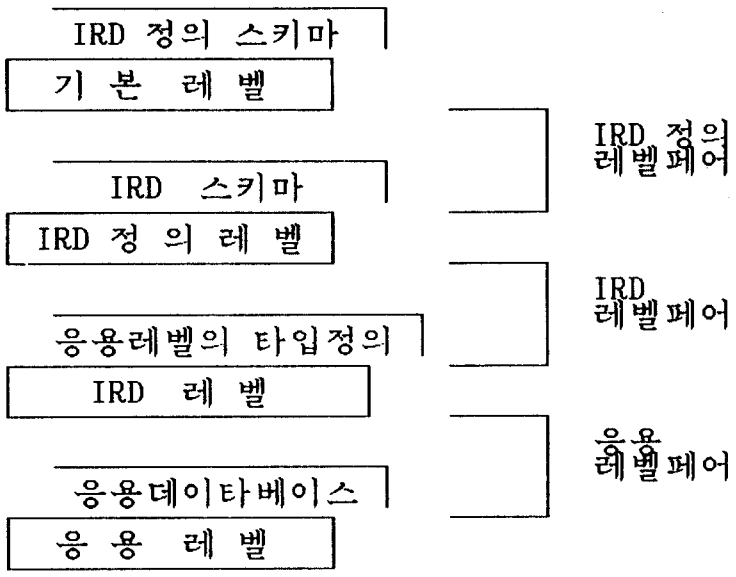
### 2.1 ISO IRDS 모델구조

IRDS 기본구조는 4개의 레벨과 각 레벨사이의 레벨페어로 구성된다.

(1) 기본레벨(fundamental level)

이 레벨은 IRD 정의 레벨에 저장될 데이타 타입을 기술한다.

(2) IRD 정의 레벨(IRD definition level)
이 레벨은 IRD 정의를 저장하기 위한 것이며 하나 이상의 IRD 스키마가 가능하다.
(3) IRD 레벨(IRD level)
이 레벨은 IRD를 저장하기 위한 것이며, IRD 내용의 일부는 응용레벨에 저장되는 데이타 타입을 저장할 수 있다.
(4) 응용 레벨(application level)
이 레벨은 상용데이타의 인스턴스들을 저장한다[12,15,16,17].




< 그림 1 > IRDS의 기본 구조

2.2 객체지향 개념과 버전
기존의 많은 데이타베이스 시스템이나 사전 시스템에서는 버전의 개념을 제공하지 않는다. 이러한 시스템들에서는 한 객체에 대한 정보는 snapshot 형태로 나타난다.
다양하게 변하는 사용자의 요구는 스키마의 변화를 요구하게 된다. 과거 스키마가 현재의 것과 전혀 관련이 없는 것이 아니라 새로이 생성된 현재의 스키마는 과거의 스키마와 공통적인 구조와 성질들을 가지면서(부분적인 경우도 있음) 만들어지는 것이 일반적이다. < 그림 2 >에서 그 간단한 예를 나타낸다.


(가) (나)

< 그림 2 > 스키마 변경의 예

버전은 시간의 흐름에 따른 정보의 변화를 나타내고자하는 의도에서 출발한 개념이다. < 그림 2 >에서 보면 알 수 있듯이 (나)의 경우 작업시간이라는 어트리뷰트가 새로이 첨가되면서 스키마가 변경된다. 기존의 방법에서는

(가)를 삭제하고 (나)를 만들어 준다. 따라서 (가)에서 적용하던 기존의 연산도 사라지고 다시 정의해서 사용해야 한다. 만약에 (가)를 그대로 가지면서 (나)의 작업시간이라는 부분만을 삽입할 수 있다면 연산도 그대로 가져다 쓸 수 있으므로 더 효율적이라고 할 수 있다. 이러한 것을 객체지향의 상속성 성질을 이용한다면 더욱 쉽게 표현할 수 있다.

따라서 제안된 IRDS 표준에도 객체지향 개념을 이용한다는 것은 실세계의 표현뿐만아니라 IRDS의 단점을 보완하면서 응용분야 개발에 생산성을 향상시키는 결과를 가져온다.

결국 객체지향 접근방법을 이용한 IRD 스키마 버전의 제어 및 관리는 조직체의 정보자원들을 효율적으로 관리하고자 하는 IRDS의 목적에 더욱 적합한 것이라고 할 수 있다.

## 3. 객체지향 방법을 이용한 스키마 정의와 버전종류 및 제어방법

관계모델의 유연성과 강한 이론적 측면을 가지면서 실세계의 보다 더 정확한 표현과 소프트웨어의 재생산성을 높여주기 위하여 객체지향 접근방법을 사용하며, 이러한 방법을 ROOM(Relation model using Object-Oriented Methodology)라고 하자.

### 3.1 ROOM에 의한 스키마표현

의미의 표현이 부족한 관계모델을 보완하면서 객체지향 방법과 관계모델을 결합하여 표현하는 방법을 [14]에서 볼 수 있다. 이러한 방법은 시스템 설계자에게 생산에 드는 비용과 시간을 절약할 수 있도록 소프트웨어 재생산성의 효과도 준다. 이 논문에서 객체지향 방법을 사용한 스키마 정의 언어의 구조를 보면 <그림 3>와 같다.

< 그림 3 >에서 필드는 속성들간의 관계나 다른 클래스와의 관계를 표현하며, 속성 타입은 시스템 정의나 사용자 정의를 사용할 수 있으며, 정수나 텍스트 타입도 가질 수 있다. 메소드는 라이브러리 형태로 시스템에서 지원한다. IS_A는 버전들간의 상속성을 나타내는 일반화(generalization) 관계의 표현이며 A_PART_OF는 집단화(aggregation) 관계의 표현이다. REL는 속성간의 관계 또는 다른 클래스와의 관계를 나타내며, METHOD에서는 정의된 클래스에서 가지는 제약(constraint)이나 관련된 연산을 수행하기 위한 것이다.

```
CLASS : 클라스 이름
    IS_A : 상위 클래스
  A_PART_OF : 통합 클래스
    REL  : 관계이름
          ( 필드1, 필드2 )
ATTRIBUTE :
    속성 이름1 :  타입1
        .          .
        .          .
    속성 이름n :  타입n
 METHODS
    메소드 이름1 ( 매개변수 리스트  )
        .                .
        .                .
    메소드 이름n ( 매개변수 리스트 )
ENDCLASS
```

< 그림 3 > ROOM에서 스키마 정의 구조

< 그림 4 >에서는 위의 스키마 구조를 이용하여 IRD 정의 레벨과 IRD레벨을 표현한 예를 나타낸다.

```
CLASS : IRD_col_def
   IS_A :
 A_PART_OF : IRD_com
    REL  :
INSTANCE 변수:
  IRD_col_name : character
  null : character
  data_type : character
  length : integer
METHODS
ENDCLASS
```

IRD_col_def

| IRD_col_name | null | data_type | legth |
|---|---|---|---|
| emp_name | N | character | 20 |
| emp_no | N | character | 8 |
| employee | N | character | 20 |
| activity | N | character | 15 |
| pay_code | N | character | 2 |
| work_time | N | integer | 3 |

```
CLASS : EMP_TABLE
   IS_A : IRD_col_def
 A_PART_OF :
   REL  : work_for
   (emp_name, employee)
INSTANCE 변수:
  emp_name : character
  emp_no : character
  employee : character
  activity : character
  pay_code : character
  work_time : integer
METHODS
  salsry(pay_code, worktime)

ENDCLASS
```

< 그림 4 > 스키마 정의를 이용한 예

3.2 버전의 종류

관계 데이타베이스 시스템이나 객체지향 데이타베이스 시스템에서는 타임스탬프(timestamp), 타임색인(time index), 앵커노드(anchor node) 등을 사용하여 시간의 흐름에 따른 변화정보를 표현, 유지하는데 사용하고 있다.

이 절에서는 3.1절에서와 같은 스키마 정의에서 버전을 생성할 수 있는 스키마 변화의 종류를 분류하면 아래와 같다.

(1) 클래스의 내용변경
  (1.1) 속성의 변화
    (1.1.1) 클래스에 새로운 속성의 삽입
    (1.1.2) 클래스로 부터 기존 속성의 삭제
    (1.1.3) 클래스에서 기존 속성의 이름 변경
    (1.1.4) 클래스에서 속성 도메인 변경
  (1.2) 메소드의 변화
    (1.2.1) 클래스에 새로운 메소드의 첨가
    (1.2.2) 클래스로부터 기존의 메소드 삭제
    (1.2.3) 클래스에 있는 메소드의 내용변경

(2) 클래스 변화
(2.1) 새로운 클래스의 첨가
(2.2) 기존의 클래스 삭제
(2.3) 클래스의 이름 변경

3.3 버전제어 방법

위 절에서 분류한 버젼들에 대한 제어를 위해서는 다음과 같은 기본적인 규칙이 필요하다

규칙 1. : 클래스 C에서 정의된 속성이 상위 클래스에 정의되어 있는 경우에는 지역정의(local definition)에 따라 현재 클래스에서 정의된 속성이 상위 클래스보다 우선순위를 갖는다(메소드나 관계의 경우도 동일).
규칙 2. : 클래스 C에서 속성이나 메소드의 변화는 이 클래스의 하위클래스(subclass)에도 이 성질들이 그대로 계승된다.
규칙 3. : 상위 클래스가 없는 새로운 클래스의 생성시에는 default root 클래스 OBJECT가 주어진다.
규칙 4. : 상위 클래스와 하위 클래스를 갖는 클래스 C의 삭제는 허용하지 않으나 사용자가 반드시 삭제를 요구하는 경우에는 명시적으로 계층성질이 그대로 유지되게 상위 클래스와 하위 클래스의내용을 변경해 주어야 한다.

이러한 규칙들을 이용하여 3.2절에서 분류한 버젼종류에 따른 제어방법들은 다음과 같다.
(1.1.1)의 경우, 새로운 속성의 삽입은 이름모순(name conflict)이 발생하지 않으면 그대로 삽입되고 하위클래스에도 적용되며, 이름모순시에는 규칙 1을 따른다.
(1.1.2)의 경우, 삭제하고자하는 속성이 계승되는 경우에는 삭제할 수 없다. 이러한 경우의 삭제는 그 클래스의 모든 하위 클래스에서 이 속성의 삭제를 의미한다.
(1.1.3)의 경우, 이름의 변경은 이름모순이 발생하면 규칙 1에 따른다.
(1.1.4)의 경우, 도메인 변경은 규칙 2를 따른다.
메소드의 변화는 클래스의 속성 변화와 동일하다.
(2.1)의 경우, 상위 클래스가 없는 경우는 규칙 3를 따르며, 이름모순이 발생하면 규칙 1를 따른다.
(2.2)의 경우, 기존의 클래스를 계층구조에서 삭제하면 하위 클래스는 삭제된 클래스의 바로 위의 클래스를 자신의 바로 위의 상위 클래스로 갖게 된다. 이러한 클래스의 삭제는 규칙2를 따른다.
(2.3)은 (1.1.3)과 동일하다.

4. 스키마 버젼 제어 시스템의 설계

3장에서 분류한 버젼들을 제어하는 시스템 VECS(VErsion Control System)의 설계를 위해 데이타 베이스에 저장되는 구조와 시스템 구성에 대해서 다룬다.

4.1 VECS 시스템에서의 저장구조

클래스DB, 속성DB, 메소드DB, 버젼정보DB가 있으며, 클래스DB에서 클래스의 저장구조 형식은 다음과 같다.

<형식>

| 클래스 ID | 클래스 이름 | 버전 ID | 상위 클래스 | 통합 클래스 | 속성 ID | 메소드 | 시간변수 (시작,끝) | 작성자 | lock |
|---|---|---|---|---|---|---|---|---|---|

클래스 ID : 클래스를 식별하기 위함
버전ID : 클래스의 버전을 나타내며, 예를 들어 1.1이라는 것은 클래스 1번에서 스키마의 변화에 의해 만들어진 하위클래스를 의미한다.
상위 클래스 : 상속성을 받는 클래스를 의미
속성 ID : 속성들의 고유ID들의 집합으로 표현
(속성도 클래스 ID와 같은 고유 ID를 가지고 있다.)
메소드 : 메소드 ID들의 집합(메소드도 고유 ID를 갖고 있으면서 라이브러리 형태로 저장되어 있다.)
시간 변수 : 생성된 날짜와 사용이 끝난 날짜를 저장
lock : 다른 사용자가 사용하지 못하게 하기 위해서 사용
< 그림 4 >에서 본 예를 정의한 저장구조를 가지고 표현해 보면 다음과같다.

| 3 | IRD_col_def | 1 | 2 | 0 | 1, 2, 3, 4 | 0 | (91/3/20,0) | 321 | N |
|---|---|---|---|---|---|---|---|---|---|

1, 2, 3, 4가 의미하는 것은 IRD_col_def에서 정의한 속성들의 ID를 의미한다. 메소드 부분이 0이라는 것은 그 클래스에서 정의된 메소드가 없는 경우를 표시한다. 상위 클래스, 통합클래스, 속성인 경우에도 동일하다.
각 버전에 따른 변화정보를 저장하기 위한 자료구조는 3 가지 형식으로 분류된다.
클래스의 삽입시에는 스키마 정의 모듈을 이용하여 삽입한다.
1. 형식 1
<형식>

| 클래스 ID | 클래스 이름 | 버전종류 ID | 버전 ID | 상위 클래스 | 통합 클래스 | 필드 A | 필드 B | 필드 C | 변경자 | 변경날짜 (시작,끝) |
|---|---|---|---|---|---|---|---|---|---|---|

이 저장구조는 속성의 삽입, 이름변경, 도메인변경, 메소드 삽입, 메소드 삭제, 클래스 이름변경시에 사용된다. 이름변경시에는 반드시 가명테이블(alias name table)에도 저장되어야 한다. 버전종류에 따라 각 필드에 저장되는 내용이 다르며, 그 내용을 살펴보면 <표 1>과 같다.

| 버전종류 | 필 드 A | 필 드 B | 필 드 C |
|---|---|---|---|
| 1.1.1 | 속성 ID | 속성 이름 | 속성 타입 |
| 1.1.3 | 속성 ID | 기존의 이름 | 변경된 이름 |
| 1.1.4 | 속성 ID | 기존의 타입 | 변경된 타입 |
| 1.2.1 | 메소드 ID | 메소드 이름 | 메소드내용 |
| 1.2.2 | 메소드 ID | 메소드 이름 | 메소드내용 |
| 1.2.3 | 메소드 ID | 기존의 내용 | 변경된 내용 |
| 2.3 | | 기존의 이름 | 변경된 이름 |

< 표 1 > 버전종류에 따른 저장내용의 분류

2. 형식 2 : 속성 삭제
〈형식〉

| 클래스 ID | 클래스 이름 | 버전종류 ID | 버전 ID | 상위 클래스 | 속성 ID | 속성 이름 | 속성 타입 | 변경자 | 변경날짜 (시작,끝) | 화일 이름 |
|---|---|---|---|---|---|---|---|---|---|---|

여기에서 화일이름은 삭제된 속성에 저장된 데이타를 가지고 있는 데이타 화일이다.

3. 형식 3 : 클래스 삭제
〈형식〉

| 클래스 ID | 클래스 이름 | 버전종류 ID | 버전 ID | 상위 클래스 | 속성 ID | 구성 메소드 | 변경자 | 변경날짜 (시작,끝) | 화일 이름 |
|---|---|---|---|---|---|---|---|---|---|

여기에서 화일 이름은 속성에 저장된 데이타를 가지고 있는 데이타 화일이다.

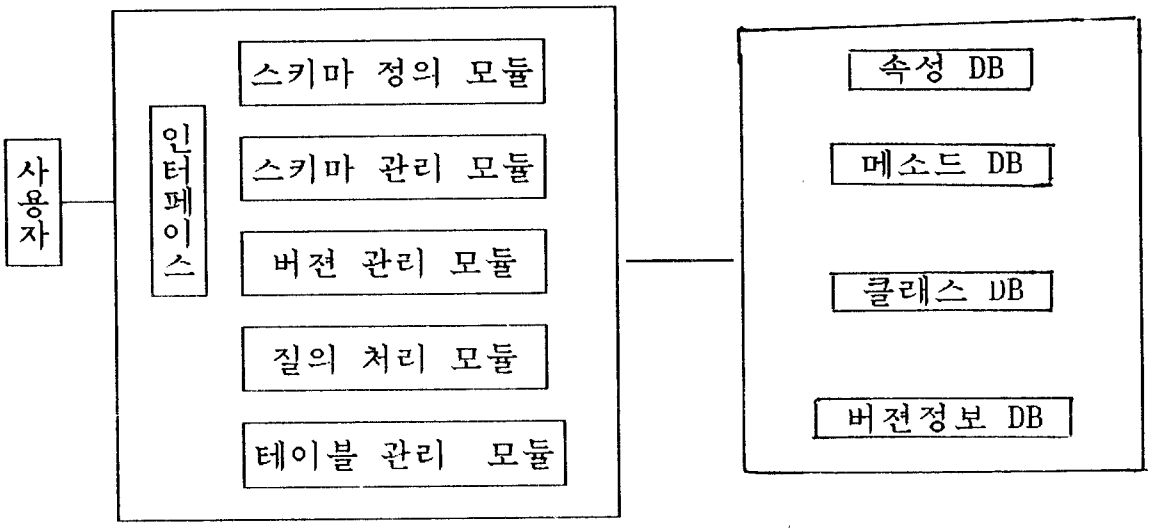
4.2 시스템의 구성

4.1절에서 정의한 저장구조 형식을 가지고서 3.1절에서 분류한 버전들을 제어하기 위한 시스템의 구성을 보면 < 그림 5 >와 같다.
사용자는 인터페이스를 통해 전체 시스템을 사용할 수 있으며, 스키마 정의 모듈은 새로운 스키마 생성을 지원해 주며, 생성된 스키마는 클래스 데이타 베이스에 등록이 된다. 또한 버전정보 데이타 베이스에서는 이 스키마로 부터 만들어지는 버전이 생성될 때 그 정보를 유지하며, 질의처리 모듈이나 스키마 관리 모듈에서 이러한 정보를 사용한다.

테이블 관리 모듈은 조직체의 여러 부서나 시스템 관리자, 사용자 또는 프로그램 등에서 동의어를 관리하기 위하여 가명 테이블이라고 하는 테이블의 사용과 관리를 지원한다. 이 테이블에서는 클래스 이름, 속성 이름, 메소드 이름에서 여러가지 이름을 가질 수 있으므로 이러한 경우에 해당 ID(클래스 ID, 속성 ID 또는 메소드 ID)와 관련 내용은 같지만 이름만이 틀린 경우로 하여 테이블에서는 이러한 가명들을 유지하고 있으면 실제 처리에 있어서는 서로 다른 이름이 같은 역할을 수행하도록 한다.

버전관리 모듈는 삽입모듈, 변경모듈, 삭제모듈로 구성되어 있다. 삽입 모듈에서는 클래스, 속성, 메소드의 삽입에 관한 변화정보의 저장과 함께 각각의 삽입에 따른 데이타를 입력하며, 클래스의 삽입시에는 스키마 정의 모듈을 이용하여 생성한다. 삽입 모듈에 의해 만들어진 정보들은 반드시 버전정보 데이타 베이스에 저장되어야 한다.

삭제 모듈에서는 클래스, 속성, 메소드를 실제로 삭제하는 것이 아니라 관련된 해당 스키마 정의에서 누락시키는 역할을 하며 각각에 저장되어 있던 데이타까지도 그대로 가지고 있다. 변경 모듈에서는 타입이나 이름, 도메인의 변경을 지원한다.




< 그림 5 > 시스템 구성도

## 5. 결론

지금까지 시간에 따른 스키마의 변화를 관리하는 방법에 객체지향 개념을 사용한 이유를 설명하고 이 방법을 이용하여 스키마의 구조를 정의했다. 이 스키마정의에서 가능한 버전의 종류를 제시하고, 버전들을 저장하기 위한 자료구조를 만들었다. 객체지향 개념의 특징인 상속성, 집단화를 이용하여 시간의 흐름에 따라 변하는 스키마 버전을 제어하는 시스템 VECS를 설계했다.

VECS 시스템에서는 스키마의 구성요소들을 atomic type으로 저장하므로 이러한 타입들의 모임에 의해 새로운 스키마를 정의하는 것이 용이하며, 스키마 일부분의 수정시에는 전체 스키마를 재정의하지 않고 수정되는 부분의 정보를 시스템에 제공하면 시스템이 자동적으로 기존의 스키마에 수정된 부분을 결합하여 새로운 스키마를 생성한다.

IRDS는 한 조직체의 정보자원을 더욱 효율적으로 관리하고자 제안된 시스템으로 이 논문에서 제안한 스키마 버전제어 방법을 이용하여 IRD 스키마 버전제어에 이용할 수 있다.

VECS 시스템은 변화정보의 획득과 타입정의 변화문제에 응용하거나 동시성 제어, 신뢰성 검사, CAD/CAM System, 혼합된 문서 정보를 갖는 사무정보 시스템, 그리고 경영정보 시스템 등의 분야에서 응용될 수 있다. 또한 이러한 시스템의 설계는 소프트웨어 재생산성을 높이는데 기여할 수도 있다.

앞으로는 버전제어 시스템 VECS의 구현과 여러가지 다른 모델로 표현된 스키마 구조들을 정의한 스키마 구조를 이용하여 표현하는 방법론에 관한 연구, 분산된 IRDS 시스템에서의 응용, 그리고 사용자에게 편리한 그래픽 사용자 인터페이스의 연구가 필요하다.

## 6. 참고문헌

[1] Ariav G., J. Clifford, and M. Jarke, "Time and Databases", Proc. ACM SIGMOID Conference 1983.
[2] Lefkovits and Sibley, "Information Resource / Dictionary Systems", Wellesley, 1983.

[3] Geofine A., "The Information Resource Dictionary System", Proc. of the 15th International Entity-Relationship conference 1985.
[4] Hong-Tai Chou and Won Kim, "A Unifying Framework for Version Control in a CAD Enviroment", Conference 12th Very Large Data Bases, 1986.
[5] Klahold P., Schlageter G., and Wilkes W., "A General Model for Version Management in Databases", Conference 12th Very Large Data Bases, 1986.
[6] Skarra A. H. and S. Zdonik, "The Management of Changing Types in an Object-Oriented Database", Proceedings Object-Oriented Programming Systems, Languages, and Applications, October 1986.
[7] Daniel R. Dolk and Robert A. Kirsch II, "A Relational Information Resource Dictionary System", Communication of the ACM Vol. 30, NO.1, January 1987.
[8] Won Kim and Hong-Tai Chou, "Versions of Schema for Object-Oriented Databases", Proceedings of the 14th VLDB Conference, 1988.
[9] Daniel R. Dolk, "Model Management and Structured Modeling : The Role of an Information Resource Dictionary System", Communication of the ACM Vol. 31 NO. 6, June 1988.
[10] Chou H. T, and Won Kim, "Versions and Change Notification in an Object-Oriented Database System", Proceedings 25ths Design Automation Conference, June 1988.
[11] Sandra V. Anderson , "The Information Resource Dictionary System (IRDS) Standard Proposal", Data Base Newsletter.
[12] Mohan Prabandham, William J. Selfridge, Douglas D. Mann, "A View of the IRDS Reference Model", Database Programming & Design, March, 1990.
[13] Mohan Prabandham, William J. Selfridge, Douglas D. Mann, "The Role of the IRDS", Database Programming & Design, April, 1990.
[14] William J. Premeriani, Michael R. Blaha, and Thomas A. Varwing, "An Object-Oriented Relational Database", Communication of the ACM, Vol 33, No. 11, 1990.
[15] ISO / IEC JTC1 / SC21 / WG3 10027, Framework, January, 1990.
[16] ISO / IEC JTC1 / SC21 / WG3 N4895, IRDS Service Interface, June, 1990.
[17] 조 완섭, 김 명준, "Information Resource Dictionary System", 정보통신기술 제2권 제3호, 1988.
[18] 우 치수, 데이타 베이스 환경의 실현 및 관리, 교학사, 1989.