스키마 진화를 지원하는 인스턴스 접근처리시스템의 설계 및 구현

진 민[†] · 김 봉 진[††]

요 약

스키마 진화 환경에서는 여러 버전의 스키마와 각각의 스키마버전에서 생성된 인스턴스가 데이타베이스에 존재한다. 본 논문은 객체관계형 데이타베이스에서 개별 클래스를 버전화하는 방법을 이용한 스키마 진화 환경에서, 인스턴스가 생성된 스키마 버전에 관계없이 인스턴스에 접근하는 방법을 제안한다. 본 논문에서 설계된 접근처리 루틴은 인스턴스를 생성한 버전에 정의되지 않은 속성에 대한 접근을 지원한다. 또한, 연속적인 갱신/역갱신 함수의 적용이나 모든 버전 쌍에 대한 처리 함수를 정의할 필요 없이 의미정보의 쌍에 대해서 접근처리 루틴을 제공하여 속성의 의미정보 변화를 지원한다. 그리고 기존의 SQL에 VERSION 조건절을 추가하여 버전에 대한 질의를 지원하고 WHERE 조건절에 의미정보를 명시함으로써 속성의 의미정보를 이용한 질의를 지원한다.

Design and Implementation of an Instance Access Handling System for Supporting Schema Evolution

Min Jin[†] · Bong-Jin Kim[††]

ABSTRACT

There exist several versions of schema and instances created under different versions of the schema concurrently in the database in schema evolution environments. This paper proposes a mechanism for accessing instances regardless of versions in schema evolution environments in object-relational databases. Access handling routines are used in accessing to the instances through the undefined attributes in the versions in which they were created. The change of semantic information of attributes is supported. Access handling routines are defined for each pair of semantics rather than defining backdate/update functions for each pair of versions of a class. A query language that accommodates version and semantic information is defined through the extension of the primitive SQL format.

1. 서 론

데이타베이스 기술이 급속히 발전하고 멀티미디어, 사무자동화, CAD/CAM, 공학응용 등에서 새로운 데이터베이스 기술이 요구됨에 따라 객체지향 데이타베이스(OODB) 또는 객체관계형 데이타베이스(ORDB)에 대한 관심히 제고되고 있다. 특히 구체적인 응용이나 환경에 따라 객체지향 또는 객체관계형 데이타베이스가 가지는 기본적인 특성인 객체, 캡슐화, 상속, 병형성 외에 추가적인 특성이 요구된다[8, 12]. 스키마 진화는 CAD/CAM, 공학응용, 소프트웨어공학, 사무자동화 등에서 필요로 하는 기능 중 하나이다[5, 8]. 스키마 진화

※ 이 연구는 1998년도 경남대학교 학술연구조성비 지원에 의하여 수행되었음.
† 종신회원 : 경남대학교 정보통신공학부 교수
†† 정 회 원 : KNnet 대표
논문접수 : 2000년 5월 31일, 심사완료 : 2000년 12월 8일

를 지원하는 시스템은 이전의 스키마를 저장, 관리하여 필요에 따라 접근할 수 있어야 한다. 그리고 이전의 스키마에 의해 생성된 개체도 스키마와 같이 보존되어 접근할 수 있어야 한다. 이러한 스키마 진화는 이력, 수정, 대체 등을 표현할 수 있어 응용에 따라 유용하게 사용되어질 수 있다. 그러나, 스키마 진화 환경에서는 스키마 일관성, 버전 일관성, 개체 일관성, 개체 접근성, 버전 공유성과 같은 문제가 해결 되어야 한다[2]. 객체지향 데이타베이스에서는 스키마 전체를 버전화하는 방법, 개별 클래스를 버전화하는 방법, 뷰를 이용하는 방법 등의 형태로 스키마 진화를 지원할 수 있다[13].

본 논문은 객체관계형 데이터베이스에서 개별적 클래스를 버전화하는 방법을 이용한 스키마 진화환경에서, 개체가 생성된 버전에 관계없이 접근하는 방법을 제안하고 이를 지원하는 질의어를 설계한다.

본 논문의 구성은 다음과 같다. 2장에서는 기존의 개체 접근 방법들을 살펴보고 이들의 문제점들을 고찰한다. 3장에서는 본 논문에서 제시하는 인스턴스 접근처리시스템(Instance Access Handling System)에 대해 기술한다. 그리고, 4장에서는 버전을 지원하는 질의처리기(Query Processor)에 대해 설명하고, 5장에서는 3장의 접근처리기와 4장의 질의처리기를 위한 사용자 인터페이스에 대하여 설명한다. 마지막 6장에서는 결론과 향후 연구 과제를 제시한다.

2. 관련연구

스키마 진화에서 버전에 무관한 인스턴스 객체에 대한 접근을 지원을 위해 많은 연구가 진행되어 왔다[3, 4, 9-11]. Skarra와 Zdonik은 ENCORE에서 타입진화스킴을 제안하였다[4]. 여기서는 모든 타입은 버전화될 수 있고 한 타입의 모든 버전은 그 타입의 버전집합으로 정의된다. 그리고 버전집합 인터페이스가 그 타입의 가장 일반적인 인터페이스로 정의된다. 타입의 새로운 버전 하나가 만들어질 때 이전의 어떠한 버전에도 없는 속성이 새로운 버전에 정의되었다면 이 속성은 버전집합 인터페이스에 포함된다. 이 버전집합 인터페이스는 그 타입의 모든 버전에 대해 하나의 인터페이스가 된다. 버전집합 인터페이스에는 정의되어 있지만 해당 버전에 없는 속성에 대해서는 예외처리기가 정의되어 다른 버전에서 생성된 인스턴스를 접근하는데 사용된다. 처리기는 전처리기(prehandlers)와 후처리기(posthandlers)가 있으며 전처리기는 읽기 처리기와 쓰기 처리기가 있다. 이 방법에서는 정의되어 있지 않는 속성이 새로운 버전에 추가될 때에 고정된 읽기 전용의 주어진 값만이 이전 버전에 제공된다. 각 개체는 정의되지 않은 속성을 위한 추가의 저장공간이 없기 때문에 진화연산에 제약이 있다. 또한 이 방법에서는 의미변화나 이름변화가 지원되지 않는다.

Monk와 Sommerville는 클래스의 버전을 위한 모델을 제안하였다[10, 11]. 이들은 또한 전진 조화성(forward compatibility)과 후진 조화성(backward compatibility)개념을 소개하였는데 전진 조화성이란 이전의 버전에서 생성된 인스턴스를 접근하도록 작성된 프로그램이 새로운 버전에서 생성된 인스턴스를 접근할 수 있는 능력을 의미한다. 후진 조화성이란 새로운 버전에서 생성된 인스턴스를 사용하도록 작성된 프로그램이 이전 버전에서 생성된 인스턴스를 접근할 수 있는 능력을 의미한다. 이 모델은 데이터의 의미 변화 없이 클래스 구조를 변경하는 것은 충분하지 못하다는 원칙에서 개발되었다.

이 모델은 의미정보가 버전 사이에서 변경되는 것을 허용하기 위해 현재 버전과 새로운 버전 사이에 변경연산이 정의된다. 새로운 버전이 생성되었을 때, 역갱신(backdate) 함수가 현재의 버전에서 생성된 인스턴스를 접근하기 위해 새로운 버전에 정의된다. 아울러 새로운 버전에서 생성될 인스턴스를 현재의 버전에서 접근하기 위하여 현재버전에 갱신(update) 함수가 정의된다. 따라서 모든 인스턴스 객체는 생성된 버전에 관계없이 어떠한 버전에서 접근될 수 있으며 의미 변화도 지원한다. 그러나 이 방법 역시 몇가지 문제점을 지니고 있다. 첫째로 사용자가 모든 갱신/역갱신 함수를 작성하여야 한다. 둘째로 멀리 떨어진 버전사이의 접근일 경우 연속적인 갱신 또는 역갱신 함수의 적용이 발생하는데, 이를 피하려면 버전의 모든 쌍에 대해 갱신/역갱신 함수가 정의되어야 하는 문제점이 있다.

Clamen은 스키마진화와 인스턴스의 적응을 지원하는 스킴을 제안하였다[9]. 인스턴스가 별도의 저장공간을 가져 동시에 여러 표현을 지원한다. 클래스의 새로운 버전이 생성되었을 때 그 클래스의 모든 인스턴스의 버전이 생성된다. 이러한 인스턴스 버전은 면(facets)으로 불린다. 여러 면에 사용되는 저장공간은 서로 공유될 수 있다. 이 방법은 각 인스턴스에 할당해야 할 저장공간이 너무 많고 각 버전에 적용하는 명세를 사용자가 작성해야 하는 부담이 있다.

3. 인스턴스 접근 처리 시스템의 설계

버전에 무관한 개체 접근을 지원하기 위하여 본 시스템에서는 모든 클래스와 버전에 관한 정보를 저장하고 관리한다. 이를 위해 ClassInfo와 VersionInfo라는 시스템 클래스가 정의되어 있다. 또한 스키마 진화에서 인스턴스가 생성된 버전에 무관한 개체접근을 위해서 접근 처리기를 설계하였다. 접근 처리기는 정의되어 있지 않는 속성에 대한 정보와 비록 속성은 존재하나 의미가 다른 속성에 대한 정보를 관리함으로써 버전에 무관한 개체접근을 지원한다.

3.1 클래스와 버전

버전은 클래스로부터 유도되거나 다른 버전으로부터 유도되어 만들어진다. 그런데 버전 유도는 객체지향 개념에서의 상위 클래스/하위 클래스의 유도와는 그 성격이 다르다. 상위 클래스/하위 클래스의 유도는 상위 클래스에서 정의된 모든 속성들을 상속받게되고 하위 클래스에서 필요로 하는 속성들을 추가함으로써 정의된다. 그러나 버전 유도에서는 시간의 흐름에 따른 변경의 기록이므로 상위 버전의 속성이 선택적으로 하위 버전에게로 상속된다.

3.2 버전 관리 정보

버전 정보의 유지 관리를 위하여 버전관리 시스템은 ClassInfo, VersionInfo라는 두개의 시스템 클래스를 이용하여 버전을 관리한다.

3.2.1 ClassInfo 클래스

클래스에 대한 정보를 관리하기 위해 다음과 같은 구조를 가지는 시스템 클래스가 정의된다.

```
ClassInfo
{
        class_name              string,
        version_count           integer,
        default_ver_number      integer,
        default_ver_flag        integer,
        latest_ver_number       integer
}
```

(1) 클래스 이름(class_name) : 해당 클래스의 이름

(2) 버전의 갯수(version_count) : 해당 클래스로부터 유도된 버전의 총 수

(3) 묵시 버전 번호(default_ver_number) : 해당 클래스의 버전을 명시하지 않았을 때 사용되는 버전

(4) 묵시 버전 플래그(default_ver_flag) : 사용자에 의해 묵시 버전이 지정 되었는지의 여부를 나타내는 플래그

0 : 묵시 버전이 설정되어 있지 않으므로 최근 버전 번호를 묵시 버전으로 사용한다.

1 : 묵시 버전이 설정되어 있으므로 묵시 버전 번호를 묵시 버전으로 사용한다.

(5) 최근 버전 번호(latest_ver_number) : 가장 최근에 유도된 버전 번호

새로운 버전이 추가될 때는 버전의 갯수가 증가하고, 묵시 버전 플래그를 참조하여 묵시 버전 플래그가 설정되어 있을 경우 최근 버전 번호만 갱신하게 된다. 만약 묵시 버전 플래그가 설정되어 있지 않다면 최근 버전 번호와 묵시 버전 번호가 동일하기 때문에 두 번호가 모두 갱신되게 된다.

3.2.2 VersionInfo 클래스

버전에 대한 정보를 관리하기 위해 다음과 같은 구조를 가지는 시스템 클래스가 정의된다.

```
VersionInfo
{
        class_name              string,
        version_name            string,
        version_number          integer,
        composite_att_list      set of string,
        parent_ver_list         set of integer,
        child_ver_list          set of integer
}
```

(1) 클래스 이름(class_name) : 해당 클래스의 이름

(2) 버전 이름(version_name) : 해당 버전의 이름을 나타낸다. 버전의 이름은 클래스의 이름과 버전 번호를 결합하여 만들게 되는데 클래스의 이름과 버전 번호를 구별하기 위해 '#'을 삽입하게 된다. 즉 computer의 네 번째 버전 이름은 'computer#4'라고 표기한다.

(3) 버전 번호(version_number) : 해당 버전의 번호

(4) 복합 속성 리스트(composite_att_list) : 버전을 구성하는 사용자가 정의한 클래스(버전) 정보를 저장한다. 만약 computer#1이 body#1, monitor#1, accessory#1을 속성으로 가진다면 복합 속성 리

스트는 {body#1, monitor#1, accessory#1}와 같은 집합의 형태로 정보를 저장한다.

(5) 부모 버전 리스트(parent_ver_list) : 해당 버전이 속성을 상속받은 버전들의 리스트

(6) 자식 버전 리스트(child_ver_list) : 해당 버전으로부터 유도된 버전들의 리스트

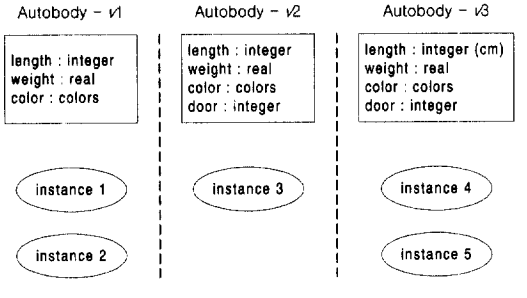
새로운 버전이 유도될 때 해당 버전의 복합 속성들은 복합 속성 리스트에 추가되고, 상속할 속성을 제공한 버전의 번호가 부모 버전 리스트에 추가된다. 그리고 부모 버전의 자식 버전 리스트 정보에 부모 버전의 속성을 참조하는 버전의 번호가 추가된다.

3.3 접근 처리기

스키마 진화에서의 접근 처리는 인스턴스가 생성된 버전에 관계없이 접근 가능해야 한다. 호출하는 버전과 인스턴스를 생성한 버전 모두에 정의된 속성을 통한 접근은 아무런 문제없이 접근이 가능하다. 하지만 다음의 두가지 경우에는 문제가 발생한다. 하나는 인스턴스를 생성한 버전에 정의되어 있지 않는 속성을 통하여 그 인스턴스를 접근하는 경우이고 또 다른 경우는 속성은 정의되어 있지만 서로 다른 의미를 가지는 속성을 통하여 인스턴스를 접근하는 경우이다.

본 논문에서는 첫 번째 문제의 해결을 위해 Skarra와 Zdonik[4]이 제시한 예외처리기 기법을 이용한 접근처리 루틴을 사용하고 두 번째 문제를 해결하기 위해 의미정보를 저장하고 이를 이용한 접근처리 루틴을 정의하여 사용한다.

(그림 1)과 같이 주어진 버전과 인스턴스에서 $v1$에 의해 생성된 인스턴스에 대해 다른 버전이 속성 door를 통하여 접근하는 경우 (그림 2)와 같이 주어진 접근처리 루틴이 작동된다.



(그림 1) 버전과 인스턴스

Class	Attribute	Set of versions	Access handling routine
Autobody	door	{1}	door : = 4 ;

(그림 2) 접근처리 루틴

의미 변화를 지원하기 위해서는 의미정보가 저장되어 있어야 된다. (그림 1)에서 $v3$의 속성 length는 의미가 inch에서 cm로 변경된 경우이다. 이때 속성 length를 통한 접근일 경우 (그림 3)과 같은 의미 변화 정보와 (그림 4)의 묵시적 의미 정보를 이용하게 된다.

Class	Attribute	Set of versions	Semantics
Autobody	length	{3}	cm

(그림 3) 의미 변화 정보

의미 변화 정보를 통하여 $v3$의 속성 length의 의미가 cm임을 알 수 있고 (그림 4)의 묵시적 의미 정보를 이용하여 명시적으로 속성 length에 대한 의미가 정의되어 있지 않는 버전에 있어서는 그 의미가 inch임을 알 수 있다.

Class	Attribute	Default semantics
Autobody	length	inch

(그림 4) 묵시적 의미 정보

이와 같은 정보를 이용하여 이름은 같지만 의미가 다른 속성을 통한 접근은 (그림 5)에서 보는 바와 같이 클래스, 속성, 인스턴스-의미, 접근-의미에 대해 정의된 접근처리루틴에 의해 처리된다

Class	Attribute	From semantics (instances)	To semantics (accessing)	Routine
Autobody	length	inch	cm	length:=2.54*length
Autobody	length	cm	inch	length:=length/2.54

(그림 5) 의미변화에 대한 접근처리 루틴

4 버전을 지원하는 질의 처리기

4.1 버전을 지원하기 위한 SQL확장

본 논문에서는 스키마 진화 환경에서 개체에 대한 접근을 지원하는 질의를 위하여 기존의 SQL을 확장하였다. (그림 6)과 같이 기존의 SQL문에 VERSION절을 추

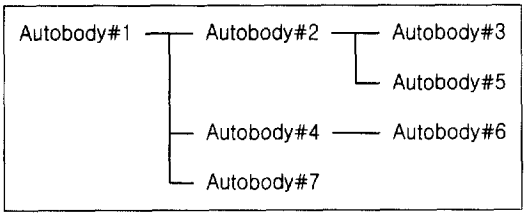
가하여 버전에 대한 질의를 가능하게 하고 WHERE 조건절에 의미정보를 표시함으로써 의미 변경을 지원한다.

```
SELECT attribute[, attribute]*
FROM class-name
[VERSION {all | version all | version[, version]*}]
WHERE attribute[semantic-information] op value

attribute : an attribute of the corresponding class
version : a version number of the corresponding class
op : comparison operator
value : a literal value or attribute specification
```

(그림 6) 버전을 지원하는 검색문 구조

다음은 (그림 7)과 같은 버전 유도 계층을 가진 환경에서 4가지 질의 유형 및 의미정보를 사용한 질의어를 보여주고 있다.



(그림 7) 버전 유도 계층

Example 1 : 모든 버전의 인스턴스에 대해 검색
SELECT door FROM autobody VERSION all WHERE color='yellow';

Example 2 : 버전 1, 4의 인스턴스에 대해서만 검색
SELECT door FROM autobody VERSION 1, 4 WHERE color='yellow';

특정 버전 이하의 모든 버전에 대한 질의는 특정 버전의 번호와 ALL 예약어를 동시에 사용함으로써 지원'된다.

Example 3 : 버전 2, 3, 5의 인스턴스에 대해 검색
SELECT door FROM autobody VERSION 2 all WHERE color='yellow';

다음과 같이 버전에 대한 명시가 없는 경우는 묵시 버전 정보를 이용하여 묵시 버전에 대해서만 질의가 실행된다. 여기서 묵시 버전은 버전을 명시하지 않았을 때 사용되는 버전으로서 사용자에 의해 지정된다. 만약 사용자에 의해 지정되지 않으면 가장 최근에 유도된 버전이 묵시 버전이 된다.

Example 4 : 버전 7의 인스턴스에 대해 검색
SELECT door FROM autobody WHERE color ='yellow';

그리고 속성의 의미 변화를 지원하기 위하여 WHERE에 다음과 같이 의미정보를 명시하게 함으로써 속성의 의미를 포함한 질의가 가능하게 하였다. 만약 의미정보가 없는 속성에 대하여 의미정보를 이용한 질의가 발생할 경우는 예외 처리를 이용하여 사용자에게 에러 메시지를 제공한다.

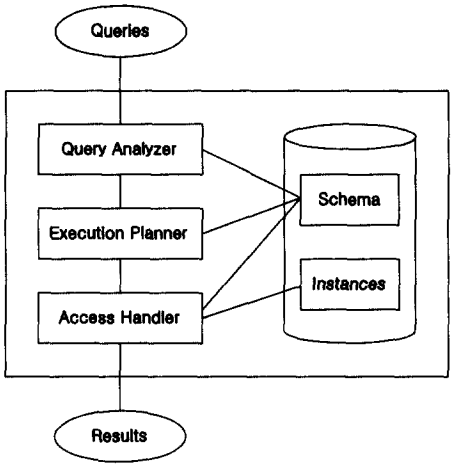
Example 5 :
SELECT door FROM autobody VERSION all WHERE length{inch} > 40;

위의 모든 질의는 3장에서 제시한 예외처리 기법을 이용한 접근처리 루틴과 묵시적 의미 정보, 의미 변화 정보를 이용한 접근처리 루틴에 의해 수행된다.

4.2 질의 처리기

사용자로부터 입력된 질의는 질의 분석기(Query Analyzer)에서 인스턴스 접근 처리 시스템이 제공하는 클래스 정보(ClassInfo), 버전 정보(VersionInfo), 접근처리 루틴, 의미 변화 정보, 묵시적 의미 정보, 의미변화에 대한 접근처리 루틴을 이용하여 버전 정보와 의미 정보를 분석하게 된다. 이때 질의에 대한 검증이 이루어지게 된다. 검증이 이루어진 질의는 실행계획자(Execution Planner)에 의해 각각의 버전에 해당하는 질의를 생성한다. 생성된 각각의 질의는 실행계획자에 의해 질의 실행계획(Execution Plan)이 수립된다. 실행계획이 만들어지면 접근처리기(Access Handler)가 실행 계획에 따라 질의를 수행한다. 이때 의미가 다른 속성이 발견되었을 경우 의미변화에 대한 접근처리 루틴을 수행함으로써 동일 의미를 부여하게 된다. 또한 해당 버전에 속성이 존재하지 않은 경우에는 예외가 발생하게 되고 발생된 예외는 예외처리기가 접근처리 루틴을 호출함으로써 시스템에 정의되어 있는 속성값을 반환하게 된다.

(그림 8)은 본 논문에서 설계한 질의 처리기의 구조를 나타내고 (그림 9)는 질의 처리기에서 질의를 처리하는 과정을 나타내고 있다.



(그림 8) 질의 처리기 구조

```
1. Accept a query
2. Analyze and verify the query
   Get the class and version information
   Get the attribute and its semantic information
   Verify the query
3. Generate the execution plan
   Check the version type
      If the version clause is left out, find the de-
      fault version number
      For each version of the version set,
         If the attribute exists,
         Then if the semantic is the same,
                then build basic-processor,
                else build semantic-processor
         else build undefined-processor
4. Execute the plan
```

(그림 9) 질의 처리 과정

5. 구현 및 사용자 인터페이스

5.1 구현 환경

본 논문의 인스턴스 접근처리시스템의 구현 플랫폼은 다음과 같다. 서버 환경으로는 Sun사의 Unix 운영체제인 Solaris 2.5이다. DBMS는 객체관계형 데이타베이스인 UniSQL/X Release 3.5.3을 사용하였다. 네트워크 연결은 TCP/IP 프로토콜을 사용하였다. 클라이언트의 운영체제는 Windows98 이다. 클라이언트의 그래픽 사용자 인터페이스를 위한 개발환경은 Delphi4.0을 사용하였다. 클라이언트와 서버의 연결을 위하여 UniSQL사에서 제공하는 ORDB for 32-bit Windows - Release 3.5.3와 UniSQL API를 이용하였다.

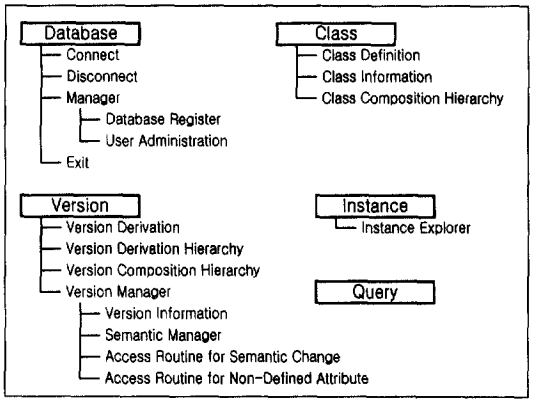
5.2 사용자 인터페이스의 설계 및 구현

메인화면은 5개의 주메뉴와 여러개의 부메뉴로 이루어져 있다. 전체적인 메뉴의 구성은 (그림 10)과 같다. Database 메뉴는 데이타베이스와 관련된 기능을 제공하는 것으로 서버와의 접속(Connect), 단절(Disconnect), 데이타베이스 환경 관리자(Manager), 프로그램 종료(Exit)의 기능을 가진다.

Class 메뉴는 클래스와 관련된 기능을 제공하는 것으로 클래스 정의(Definition), 해당 클래스 정보(Information)보기, 클래스 구성 계층도(Composition Hierarchy) 보기 기능을 제공한다.

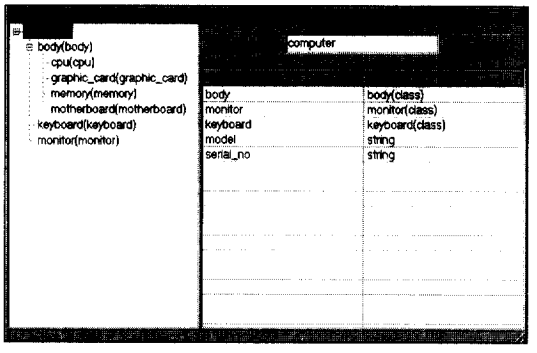
Version 메뉴는 버전을 관리하기 위한 기능을 제공하는 것으로 버전 유도(Derivation), 유도 계층도(Derivation Hierarchy), 구성 계층도(Composition Hierarchy) 등을 제공한다. 그리고 해당 버전의 정보를 볼 수 있는 기능을 제공하고 접근 처리기에서 사용되는 의미 정보를 관리하는 기능(Manager)을 제공한다.

Instance 메뉴는 인스턴스 정보를 탐색하는 Instance Explorer를 제공하고 Query 메뉴는 GUI를 이용한 질의 기능을 제공한다.



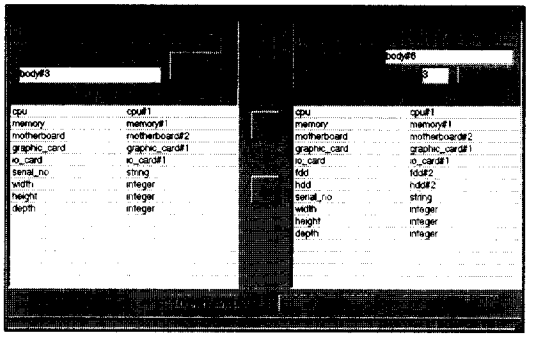
(그림 10) 전체 메뉴 구성

본 시스템이 제공하는 몇가지 부속메뉴를 구체적으로 살펴보기로 한다. 생성된 클래스의 구성 계층 구조를 보기위해 (그림 11)과 같은 클래스 구성 계층도가 제공된다. 클래스 구성 계층도는 왼쪽의 트리에서 복합속성에 해당하는 클래스를 표시하고, 오른쪽에는 트리에서 선택된 클래스의 속성들이 표시된다.



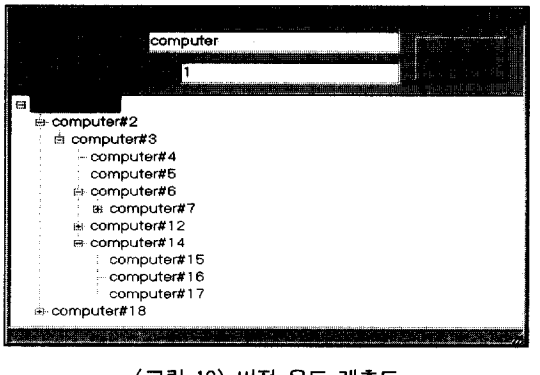
(그림 11) 클래스 구성 계층도

또한, 새로운 버전을 유도하기 위해서는 (그림 12)와 같은 버전 유도기를 사용한다. 버전 유도 과정은 상속 개념과 다른 선택적 상속을 이용하여 유도하게 된다. 왼쪽에 참조가 되는 버전의 속성 정보가 표시되고 왼쪽에는 선택 또는 추가된 속성들이 표시된다.

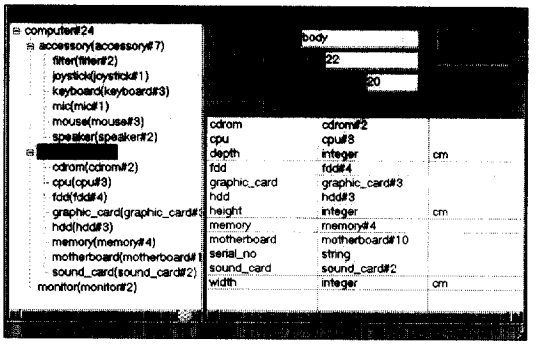


(그림 12) 버전 유도

생성된 버전들의 유도 구조를 보기 위해 (그림 13)과 같은 버전 유도 계층도가 제공된다. 그리고 각 버



(그림 13) 버전 유도 계층도

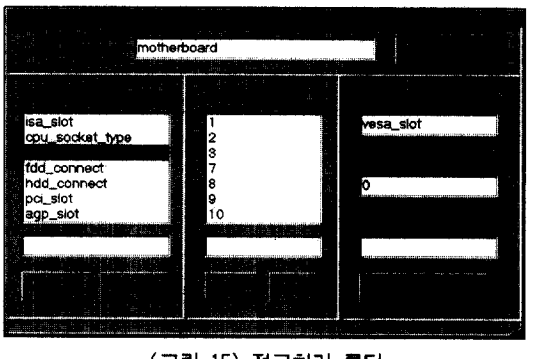


(그림 14) 버전 구성 계층도

전들의 구성 정보를 보기위해 (그림 14)와 같은 버전 구성 계층도를 제공한다. 버전 구성 계층도는 클래스 구성 계층도와 같은 형태로 표시되지만 클래스가 아닌 버전으로 구성된다는 차이점이 있다.

버전에 무관한 개체 접근을 위해 사용되는 접근처리 루틴, 의미변화정보, 의미변화에 대한 접근 처리 루틴을 관리하는 곳이 버전 관리자(Version Manager)이다.

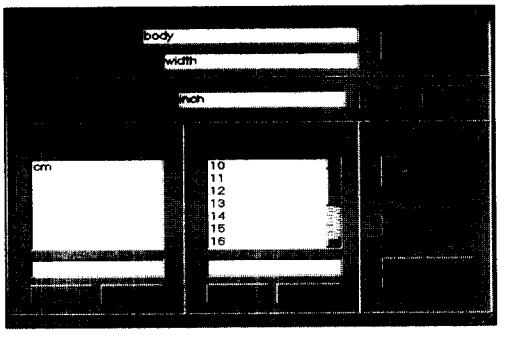
(그림 15)는 (그림 2)의 접근처리 루틴을 입력하는 사용자 인터페이스이다.



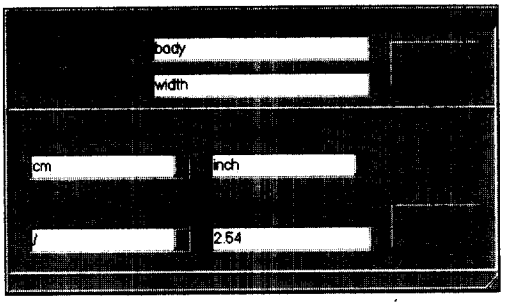
(그림 15) 접근처리 루틴

각 버전에 정의된 속성들의 의미 변화를 지원하기 위해 제공되는 의미 정보를 입력하기 위해 (그림 16)과 같은 의미 정보 관리자를 제공한다. 그리고 의미가 다른 속성을 통한 접근을 처리하기 위해 필요한 접근처리 루틴을 정의하기 위해 (그림 17)와 같은 사용자 인터페이스를 제공한다.

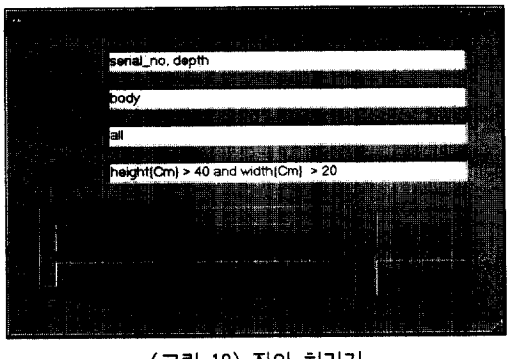
질의 과정에서 사용자 편의성 및 효율적인 질의처리를 가능하게 하기 위하여 (그림 18)와 같은 사용자 인터페이스를 제공하며, 실행된 질의의 결과는 (그림 19)과 같이 표시된다.



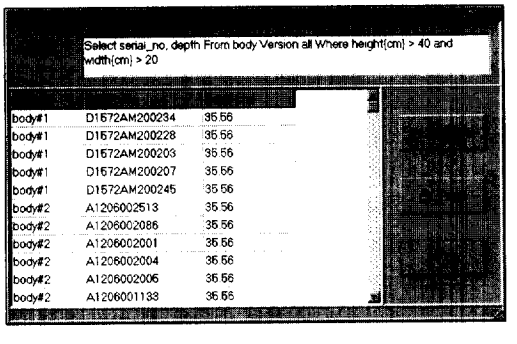
(그림 16) 의미정보 관리자



(그림 17) 의미변화를 지원하는 접근처리 루틴



(그림 18) 질의 처리기



(그림 19) 질의 결과

6. 결론 및 향후연구

본 논문에서는 클래스 버전을 기반으로 하는 스키마 진화 환경에서 클래스버전에 관계없이 인스턴스 객체를 접근하는 방법과 이를 지원하는 질의어를 설계하고 이를 지원하는 질의 처리기와 접근 처리기를 설계하고 구현하였다.

본 논문에서 제안한 접근 기법은 예외 처리기 방법[4]에서 지원하지 못하는 의미정보 변화를 지원할 뿐만 아니라 갱신/역갱신 함수 방법[10, 11]에서 처럼 모든 버전의 쌍에 대한 갱신/역갱신 함수를 정의할 필요 없이 의미 정보의 쌍에 대해서만 접근처리 루틴을 정의하였다.

그리고, 스키마 버전에 무관한 개체 접근을 위해 SQL을 확장하여 의미 변경을 지원하는 버전에 대한 질의가 가능한 질의어를 제시함으로써 기존의 SQL을 사용하는 이용자들로 하여금 보다 쉽게 버전을 사용한 질의를 가능하게 하였다. 뿐만 아니라 질의 과정에 필요한 사용자 인터페이스를 제공함으로써 사용자 편의성 및 효율적인 질의처리를 가능하게 하였다.

앞으로 본 연구를 기반으로 하여 다중 조건, 중첩 질의 등을 지원할 수 있도록 질의어를 확장하고 다양한 의미 변화 연산을 지원하기 위해 질의 해석기의 파서가 확장되어야 할 것이다.

참 고 문 헌

[1] 진 민, 김봉진, "스키마 진화에서 버전에 무관한 개체 접근", 한국정보과학회 '98 봄 학술발표논문집, 제25권 제1호, pp.190-192, 1998.

[2] 진 민, "객체지향 데이터베이스의 스키마 진화 환경에서 버전에 무관한 인스턴스 접근", 한국정보처리학회 논문지, 제6권 제3호, pp.553-561, 1999.

[3] A. Bjornerstedt and S. Britts, "AVANCE : An Object Management System," In Proceedings of OOPSLA'88, pp.206-221, 1988.

[4] A. H. Skarra and S. B. Zdonik, "Type Evolution in an Object-Oriented Database," In Research Directions in Object-Oriented Systems, pp.393-413, 1987.

[5] J. Banerjee, W. Kim and H. F. Kirth, "Semantics and Implementation of Schema Evolution in Ob-

ject-Oriented Databases," In Proceedings of the ACM SIGMOD International Corference on Management of Data, pp.311-322, May. 1987.

[6] M. Jin, "An Object-Oriented Database Approach for Supporting Product Evolution in Agile Manufacturing," University of Connecticut, Ph.D., 1997.

[7] Min Jin, Bing-Jin kim, Min-Soo Jung, Wu Woan Kim, T.C. Ting, "An Instance Handling System in Schema Evolution," Proceedings of the ISCA 12th International Conference on Computers and Their Applications in Industry and Engineering, Atlanta, pp.111-114, November. 1999.

[8] R. G. G. Cattell, "Object Data Management : Object-Oriented and Extended relational Database Systems," Revised Edition, Addion-Wesley, Reading, MA, 1994.

[9] S. M. Clamen, "Schema Evolution and Integration," Distributed and Parallel Databases, Vol.2, No.1, pp. 101-126, 1994.

[10] S. R. Monk and I. Sommerville, "A Model for Versioning of Classes in Object-Oriented Databases," In Proceedings of the 10th British National Conference on Databases, pp.42-58, July. 1992.

[11] S. R. Monk and I. Sommerville, "Schema Evolution in OODBs Using Class Versioning," SIGMOD Record, Vol.22, No.3, September. 1993.

[12] W. Kim, "Introduction to Object-Oriented Databases," The MIT-Press, Cambridge, MA, 1990.

[13] W. Kim and H. T. Chou, "Versions of Schema for Object-Oriented Databases," In Proceedings of the 14th VLDB Conference, pp.148-159, 1988.

진 민

e-mail : mjin@mail.com.kyungnam.ac.kr

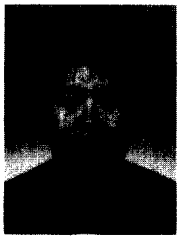
1982년 서울대학교 계산통계학과 졸업(이학사)

1984년 한국과학기술원 전산학과 졸업(공학석사)

1997년 코네티컷 주립대학교 컴퓨터공학과(공학박사)

1985년~현재 경남대학교 정보통신공학부 교수

관심분야 : 객체지향 데이타베이스, 멀티미디어 데이터베이스, 데이터 모델링, 분산처리



김 봉 진

e-mail : bjkim@knnet.co.kr

1998년 경남대학교 전자계산학과 졸업(공학사)

2000년 경남대학교 컴퓨터공학과 졸업(공학석사)

2000년~현재 KNnet 대표

관심분야 : 객체관계형 데이타베이스, 데이터모델링, 분산처리, 데이타베이스 마케팅