□ 기술해설 □

표준화 안

객체지향 데이타베이스 표준화-ODMG 모델

서울대학교 조은선* · 이강우 · 이상원

목 차

1. 도 입

컴퓨터를 이용하여 대량의 데이타를 처리하고자 할 때 어떻게 데이타를 체계적으로 조직하고 효율적으로 관리하는가는 오래전부터 중요한 관심 대상이 되어 왔다. 현재 가장 널리 쓰이고있는 관계형 데이타베이스 시스템은, 계층형이나 CODASYL 데이타베이스 시스템에서 부담스럽게 강요된 네비게이션(navigation)문제를 깨끗이 해결하였음은 주지의 사실이다. 그러나 사회가 발전함에 따라 데이타베이스의 응용분야가 넓어진 후로 관계형 데이타베이스 시스템이 여러가지 한계에 부딪히게 되었다. 따라서, 최근에는 이러한 한계들을 극복하고자 하는 노력이 많이 시도되고 있으며, 이것을 데이타베이스 분야의 제 5 세대라고 부른다[2].

5세대 데이타베이스 시스템의 여러 연구 중에서도 가장 활발히 진행되고 있는 것은 확장된 관계형 데이타베이스 시스템(extended relational database system)과 논리 데이타베이스 시스템(logic database system), 그리고 객체 지향 데이타베이스 시스템(object-oriented database system)에 관한 연구이다. 이 중 실세계와 가장 자연스럽게 부합되어 많은 주목을 받고있는 것은 객체지향 데이타베이스 시스템이다. 그러나 관계형 데이타베이스의 사용자들은 쉽게 객체지향 데이타베이스로 전환하지는 못한다. 그 가장 큰 이유는 객체지향 데이타베이스의 의미(semantics)가 각 객체지향 데이타베이스 시스템 마다 달라진다는데 있다. 예를 들면, 객체지향 데이타베이스의 클래스와 그에 속하는 객체와의 관계는 시스템마다 약간씩 달라서 특히 데이타베이스 프로그램을 이식할 때에는 어려움이 많다. 따라서 모든 관계형 데이타베이스의 표준 질의어인 SQL을 사용하고있는 사용자 측에서는, 간단한 수정만으로 프로그램 이식이 가능한, 관계형 데이타베이스를 고수하게 된다.

객체지향 데이타베이스를 개발하는 측에서는 이러한 점을 인식하고, 표준화를 위한 노력을 해왔다. 실제로 이미 객체지향 모델 자체의 표준화는 OMG[4] 등에서 되어있었다. 따라서, 91

* 준회원

년에는 이를 지원하는 객체지향 데이타베이스 모델이 나오게 되었는데, 가장 대표적인 객체 지향 데이타베이스 시스템 회사들-Versant Object Technology, Object Design, Sun Soft, ONTOS, O_2 Technology, Objectivity- 주축이 되어 ODMG Group이라는 연합을 발족시켰으며, 93년 여름에는 그 결실로 객체지향 데이타베이스의 표준이 나오게 되었다. 이 표준안은 결성자들간의 투표와 DEC, HP, ITASCA, Servio, POET, Matisse 등의 회사의 검토를 받아 만들어졌다. 그리고, 빠른 시일안에 OMG나 ANSI 등의 다른 표준화 기구에서 이를 토대로 표준안을 공식화할 전망이며, 아직 협의되지 않은 세부 사항이나 새로 필요성이 발견되는 사항들에 대해서 첨가가 계속될 것이다[3]. 본 글에서는, 'ODMG-93 Document[1]'의 중요부분을 주로 발췌하여, 이 표준화 모델에 대한 간단한 소개를 하고자 한다. 2절에서는 ODMG모델에서 지원되는 객체의 의미(semantics)를 설명하고, 3절에서는 객체 정의어 ODL, 4절에서는 객체 질의어인 OQL에 대해 각각 언급한다. 5절에서는 기존의 프로그래밍 언어인 C++과 Smalltalk에 바인딩되는 방식을 다루며, 끝으로 6절에서 결론을 맺는다.

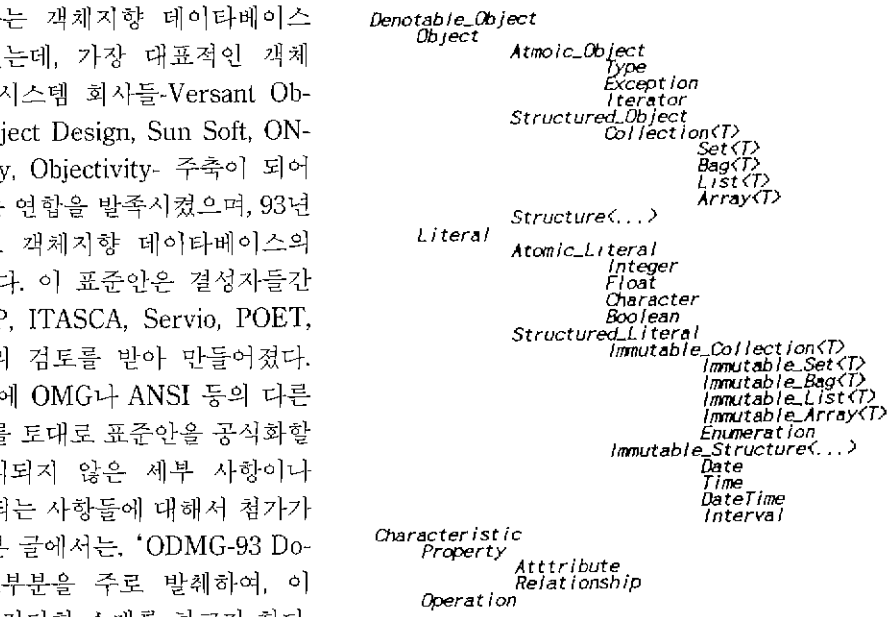


그림 1 ODMG 모델의 기본적인 타입 계층 구조

2. 객체 모델

ODMG 모델의 가장 기본이 되는 것은 '객체'이다. 객체는 그것의 상태와 행위로 이루어지는데, 객체의 행위는 연산(operation) 집합으로 정의되며, 객체의 상태는 성질(property) 집합으로 정의 된다.

2.1 객체와 타입

객체는 타입(type)으로 무리진다. 타입에는 하나의 인터페이스(interface)와 하나 이상의 구현(implementation)이 존재하는데, 타입 자체도 하나의 객체이다. 이러한 다중 구현(multiple implementation)은 하나 이상의 기계에 연결된 네트워크에 유용하며, 여러 컴파일러나 환경에 적응하기 쉽고, 프로그래머에게도 여러가지 상황별로 최적의 구현들을 구비할 수 있게 하는 장점이 있다.

타입들 간에는 계승 관계가 가능하며, 이 관계들은 계층 구조(hierarchy)를 이루는데, 기본적인 계층 구조는 그림 1과 같이 설정되었다. 응용 프로그래머는 이러한 타입들의 서브 타입을 생성함으로써 새로운 클래스를 정의할 수 있다. 계승시 이름이 충돌하는 경우에는 재정의(redefine)로 해결한다.

타입이 정의될 때는 익스턴트(extent)라는 것을 명시적으로 정의하여 그 타입에 속하는 인스턴스(instance)들의 집합을 나타낼 수 있다. 이 집합은 객체의 생성/삭제시 자동 관리된다. 서브타입은 이러한 익스턴스들의 부분집합 관계로 정의된다. 다음은 그 예이다.

```
interface Car
{
    extent: cars;
   …
}
```

타입의 구현은 표현부(representation)와 메쏘드(methods)로 이루어 진다. 표현부는 자료 구조이고, 메쏘드는 프로시듀어 내용이다. 이들은 인터페이스에서 기술되지 않은 내부적인 것일

수도 있다. ODMG 모델에서는 하나의 인터페이스와 하나의 구현이 결합한 것을 클래스(class)라 부른다.

그림 1에서 보듯이 ODMG 객체 모델은 가지칭-객체(Denotable Object)와 특성(property)으로 구성되는데, 모든 가지칭-객체는 식별자(DID)와 값(Value)을 가진다. 식별자는 가지칭-객체을 식별하는데 사용되며, 값은 각 가지칭-객체의 상태를 표현한다. 각 가지칭-객체는 그것의 식별자의 부여방법에 따라 가변형 객체(mutable object)와 리터럴(literal)로 구별된다. 리터럴은 그것이 갖는 값이 자체가 식별자로 사용되며, 예로는 '1', '4.5'와 같은 정수 또는 부동소수를 들 수 있다. 가변형 객체는 그것이 갖는 값과는 독립적으로 고유의 식별자를 갖게 된다. ODMG에서는 가변형 객체를 줄여 주로 '객체'라고 부른다.

모든 식별자는 각 객체 생성시 고유하게 하나가 부여되지만, 객체의 이름은 여러 개를 가질 수 있다. 그리고, 데이타베이스의 질의를 위해 객체를 조건문으로 식별하기도 할 수 있다. 다음은 타입 section의 원소 중에서 section_number가 1993인 원소를 찾는 질의이다.

```
sections.select_element("section_number==1993")
```

객체의 삭제시 해당 객체와 관련있는 다른 객체들이 한꺼번에 자동으로 지워지지는 않는다. 그리고, 객체의 지속 기간은 해당 프로시듀어나, 프로세스 또는 데이타베이스의 지속 시간 중 하나로 제한된다.

객체의 성질은 속성(attribute)과 관계(replationship)로 구성된다. 속성은 타입과 리터럴의 쌍으로 나타내어진다. 속성에 작용하는 연산중 시스템이 제공하는 것으로는 set_value( ), get_value( ) 등이 있다. 이들 연산을 오버로딩하면, 트리거(trigger) 기능이나 제약 조건(constraint) 기능을 지원할 수 있다.

관계는 두개의 가변형-객체들 간에 정의된다. 이것은 관계형 데이타베이스 사용자의 편의를 위한 것이다. 이러한 두 객체 간에는 따라갈 수 있는 경로(traversal paths)가 존재하게 된다. 다음 예는 Student가 Course를 이수하는 상황을 관계로 모델링한 것이다. 의미상으로 Student의 takes의 변동 상황이 Course의 is_taken_by에도 반영되야하는 경우이다.

```
interface Student
{ …
  takes: Set<Course> inverse
         Course::is_taken_by
};
interface Course
{ …
  is_taken_by:
              Set<Student> inverse
              Student:takes
};
```

이러한 관계는 inverse 키워드에 의해 역 방향 연산을 자동적으로 보장받게 된다. 즉, 한쪽 방향에서 해당 관계에 삭제, 변경등을 하면, 이것이 반대편 타입의 객체들의 관계에도 반영되어, 참조적 무결성(referential integrity)이 보장될 수 있게 된다. 관계에 적용할 수 있는 연산들은 delete( ), add_one_to_one( ), remove_one_to_one( ), create( ), traverse( ) 등이 있다.

연산은 타입과 독립적으로 정의 되거나, 두 가지 이상의 타입에 정의될 수 없다. 그러나, 오버로딩은 가능하다. 클래스 operation에 가할 수 있는 연산들은 invoke( ), return( ), return_abnormally( ) 등이 있다.

2.2 예외처리(exception)

ODMG에서는 예외 상황 자체도 하나의 객체로 간주한다. 따라서 C++에서와 같이 예외 상황끼리 타입 계층구조를 이룰 수 있다. 이러한 계층 구조로써 타입 A의 예외처리기가 A의 서브타입 객체를 예외 처리할 수가 있다. 그리고, 중첩된 예외처리기도 지원한다고 되어있다.

2.3 구조 객체(Structured objects)

구조 객체에는 구조(structure)와 모임(collec-

tion)이 있다. 구조는 이름 붙은 슬롯(slot)이 여러 개 있어 이것을 원소로 하는 것이고, 모임은 같은 타입의 원소를 모은 것이다.

구조는 관계형 데이타베이스의 레코드와 비슷하다. 각 필드는 또 다른 타입의 객체를 값으로 가질 수 있기 때문에 구조의 중첩(nesting)을 지원한다. 복사(copy)는 원소간의 복사(shallow copy)를 기본으로 한다.

모임은 Bag〈Part〉, List〈Course〉와 같은 타입 생성기(type generator)의 인스턴스이다. 데이타베이스는 이러한 모임에 대해 질의할 수 있다. 이러한 모임의 타입인 collection은 서브타입으로 조건제시 모임(predicate-defined collection)과 원소나열 모임(instance-defined collection)이 있다. 조건제시 모임(predicate-defined collection)은 타입과 조건식을 명시하여 이루어지는 모임으로써 현재는 타입의 익스턴스밖에 없다. 원소나열 모임은 일반적인 집합을 의미한다. 모임이 가지는 성질과 연산에는 insert_element( ), remove_element( ), replace_element_at( ), retrieve_element_at( ), select_element( ), select( ), create_iterator( ), create_index( ), drop_index( ) 등이 있다.

반복(iteration)은 반복연산자(iterator)를 정의함으로써 가능하다. 반복연산자 iterator의 타입 Iterator〈T〉는 next( ), first( ), last( ), more?( ), reset( ), delete( ) 등의 인터페이스를 가진다. 만일 다음과 같이 all_component가 타입 Part의 Bag으로 선언되었다면,

```
all_components: Bag〈Part〉;
i: Iterator〈Part〉
p: Part;
all_components: =engine;
```

all_components의 반복연산자는 Bag 타입의 메쏘드인 create_iterator( )를 써서 다음과 같이 생성되며,

```
i: =all_components.create_iterator( );
```

프로그램 상의 다음과 같이 while문과 반복연산자를 통해 생신된다.

```
while ((p: =i.next( )) !=nil) {
    all_components:=
        all_components union p.components
};
```

이러한 모임을 위해 Set〈T〉, Bag〈T〉, List〈T〉, Array〈T〉가 지원된다.

2.4 기타

연산, 성질, 모임, 속성, 타입, 예외처리 등은 그 자체가 하나의 객체이며, 각각 Operation 타입, Property 타입, Collection 타입, Attribute 타입, Type 타입, exception 타입 등의 인스턴스가 된다. 데이타베이스도 역시 객체이며, 타입 Database의 인스턴스가 된다. 데이타베이스는 지속적 객체의 저장 장소를 지칭하며, 타입 이름이나, 익스턴스 이름, 루트 객체 이름을 통해 접근된다. 트랜잭션도 타입 Transaction의 객체이며, 중첩 트랜잭션을 지원한다. 트랜잭션에는 begin( ), abort( ), commit( ) 등의 연산이 지원된다. 트랜잭션은 표준적인 잠금(lock) 기반 접근 기법으로 지원하는 것을 원칙으로하며, 비관적인 동시성 제어(pessimistic concurrency control)를 기본으로 하고있다.

3. 객체 정의어 ODL

ODL은 객체 타입의 인터페이스를 정의하는 언어이다. 이 언어를 통해 이식성과 상호 운용성을 살리려고 하며, ODMG 모델에 나타난 모든 의미(semantics) 구성을 가능하게 한다는 취지이다. 프로그래밍 언어가 아닌 명세(specification) 언어이기 때문에 사용하는 프로그래밍 언어에 독립적이다. OMG의 IDL(Interface Definition Language)[4]과 호환하고, 구현이 용이하게 하려는 원칙이 있다.

타입의 특성(characteristic)이라 불리우는 상위 타입, 명시적 익스턴트, 키(key) 등은 각각 키워드를 사용하여 명시한다. 익스턴트와 키 정의는 한 타입당 한 번 이상이 불가능하며, 익스턴트, 키, 슈퍼 타입의 선언은 불필요할 경우 생략이 가능하다.

속성 명세는 키워드 **attribute**에 일반 변수 정

의를 덧붙임으로써 가능하다. 키워드 **attribute**는 생략 가능하다. 관계 명세는 관계의 경로(traversal path)를 정의하는 것이다. 목표로 하는 타입의 경로와 순서화 정보 및 역 연산 정의 등이 가능하다. ODL의 연산 명세는 IDL[4]의 연산 명세와 호환이 가능하다. 익스텐트, 키, 성질 리스트 〈property_list〉와 연산 리스트 〈operation_list〉가 들어간 예는 다음과 같다. 이것은 타입 Person에서 계승받은 Professor타입의 선언이다.

```
interface Professor: Person {
  extent professors;
  keys faculty_id, soc_sec_no;
  attribute integer[6] faculty_id;
  Address address;
  attribute
        Set〈Struct〈 string degree_name,
        Year degree_year〉〉 degree;
  relationship Set〈Student〉 advisees
        inverse Student::advisor;
  relationship Department department
        inverse Department::faculty;
  fire(in Professor) raise(no_such_employee);
}
```

4. 객체 질의어 OQL

객체 질의어 OQL은 계산적으로 완전하지는 않으나, 선언적 접근이 가능하고, ODMG 모델을 기반으로, 추상적인 문법을 제공하는 것을 목표로 한다. 그리고, 집합이나 구조, 리스트 등의 복합 객체를 다루는 높은 수준의 구문을 제공하는 것을 원칙으로하며, 갱신을 위한 구문을 포함하지 않고, 기존의 프로그래밍 언어의 바인딩에 의존한다.

이러한 추상 언어는 문법을 형식화하기 간단하도록 되어있으며, ODMG에서도 이미 SQL과 비슷한 구체적인 언어를 하나 설계하여 추천하고있다. 본문에서는 이 언어를 중심으로 소개한다.

4.1 결과 타입

다음은 name이 "Pat"인 사람의 나이와 성별을 질의하여, 타입 set〈struct〉 의 리터럴을 리턴하는 질의이다. OQL의 질의는 select절에 명시한 것에 따라 리턴 타입을 반드시 가진다.

```
select distinct struct(a:x.age, s:x.sex)
from x in Persons
where x.name="Pat"
```

따라서 다음과 같은 중첩 질의의 타입도 타입 set〈 struct(name: string, hps: bag〈Employees〉)〉의 리터럴임을 쉽게 알 수 있다. salary가 100 000보다 큰 subordinate를 가진 고용인과 해당 subordinates집합을 질의한다.

```
select distinct
  struct(name:x.name, hps:
   (select y
    from y in x.subordinates where y.salary
    〉 100000))
 from x in Employees
```

그리고, 이와 같은 중첩된 select from where절은 select절 외에도, from, where절 어디든지 나타날 수 있다. 다음 예에서는 seniority가 '10'인 고용인중 name이 "Pat"인 사람을 질의한다.

```
select struct(a:x.age, s:x.sex)
from x in (select y
           from y in Employees
           where y.seniority="10")
where x.name="Pat"
```

객체 이름이나 '.'으로 연결된 경로는 하나의 객체나 객체 집합을 리턴한다.

4.2 객체 식별자

객체의 생성은 다음과 같은 해당 타입의 생성자를 써서 여러 형태의 객체를 생성할 수 있다.

```
Object: t(p_1:e_1,…, p_n:e_n)
Structure: struct(p_1:e_1,…, p_n:e_n)
Set: set(e_1,…, e_n)
Bag: bag(e_1,…, e_n)
```

List: list(e_1,…, e_n)
Array: array(e_1,…, e_n)

select-from-where절의 결과 값은 생성자의 초기값으로 들어갈 수도 있다. 기존의 객체를 뽑아내면 식별자를 가지는 객체의 모임이나 리터럴의 모임, 객체 등이 리턴된다. 질의는 적절한 연산을 호출하여 재귀적으로 수행될 수 있다. 질의문은 'define 〈var〉 〈query〉'문을 사용하여 원하는 변수에 할당될 수 있다. 객체에는 1진, 2진 산술식을 적용할 수 있으며, 각 모임에는 for all. exists, in(원소관계), select-from-where, sort-by, 그밖의 1진 연산 등을 적용하는 것을 원칙으로한다.

관계형 데이타베이스에서와 같이 group-by 절도 지원 하며, 아래의 예와 같이 모임의 i번째 원소를 직접 접근할 수 있다.

```
element(select x
  from x in course
  where x.name="math" and x.number=
  "101"). requires[2]
```

list(a. b, c, d)[1:3]과 같이 리스트의 일부분을 접근할 수 있으며, first(e), last(e)나 접합(concatenation)을 위한 '+' 등의 연산이 제공된다. 합집합 곱집합 차집합 등 2진 집합 연산도 SQL과 거의 같은 구문으로 가능하다.

구조의 각 원소의 접근을 위해서는 '.' 이나 '→' 둘다 동일한 의미로 쓰이는데, C++과는 다른 부분 중 하나이다. 타입이 엄격하므로, 하나의 원소를 가지는 모임을 하나의 객체로 변경하는 element( ), 리스트를 집합으로 변경하는 listtoset( ), 여러겹 구조로 되어있는 구조 객체를 단일 모임으로 만드는 flatten( ) 등의 연산이 존재한다. 이외에도 C++의 타입 변환 식과 같은 구문도 가진다.

연산은 원소의 접근과 마찬가지로 '.' 나 '→'로 호출되며, 그 리턴 값과 타입을 사용한다.

5. C++ 바인딩

ODMG C++ 바인딩의 원칙은 프로그래머에게 하나의 언어로 인식되도록 하는 것이다. 따라서, 타입 시스템, 구문, 의미론 등이 일치해야 하며, 기반 언어인 C++과 중복되는 기능을 가지지 않도록 하고, OML의 식과 C++의 식이 서로 섞여 피연산자에 자유롭게 나타날 수 있어야한다.

C++ 바인딩은 크게 C++ODL, C++OML, C++OQL로 구성된다. C++ODL은 클래스 라이브러리와 확장된 C++구문으로 정의되며, C++OML은 객체 추출과 갱신으로 이루어진다. C++OQL은 기존의 OQL을 그대로 내포하는 방식이다.

5.1 C++ODL

C++ODL에서는, 타입은 C++의 클래스로, 객체는 C++의 참조(reference)[6]로, 리터럴은 시스템 클래스의 객체나 다른 클래스에 내포된 클래스의 객체로 그대로 대응 시킨다. 그러나, 관계는 C++ 메쏘드, Ref, 모임 등 으로 시뮬레이션해야하고, 키(key)는 C++ 멤버, 코드 부분, 인덱스 등으로 시뮬레이션해야하며, 익스텐트는 언어에서 다루지 않고 프로그래머에게 그 구현을 맡긴다. 그리고, 다중 구현도 C++에서는 지원이 불가능하다.

C++바인딩을 위해 가데이타베이스화 클래스(database capable class)라는 것을 두는데, 이 부류에 속하는 클래스는 지속적이고, 비지속적인 인스턴스를 둘다 가질 수 있다. 따라서 데이타베이스에 속하는 객체는 이러한 클래스의 인스턴스여야 한다.

템플릿(template) 클래스인 'Ref'는 가데이타베이스화-클래스를 위한 포인터 클래스이다. ODL 전위처리기는 각 가데이타베이스화-클래스 X에 대하여, Ref〈X〉를 준비한다. 객체 식별자를 가지는 ODMG 객체들은 이러한 Ref 클래스를 통하여 다루어진다.

일부 시스템에서는 지속적인 객체(persistent object)의 일부가 되는 객체(component object)에 무조건적으로 지속성을 부여하지만, ODMG에서는 그렇게하지 않는다. 그리고, ODMG에서는 물리적인 기억장소에 대한 언급이 없으며, 데이타베이스를 생성하거나, 모임에 인덱스를 두는 것 등에 대해서도 다루지 않고 있다. 다음의 예는

Ref 타입의 관계와 Set 타입의 관계를 모두 가지고 있는 C++클래스 선언이다.

```
class Professor: public person
{
  public:
   //properties
   int id_number;
   char* name;
   Ref〈Department〉 dept inverse Depart-
   ment::professors;
   Set〈Student〉 advisees inverse Student::
   advisor;
  //operations:
   void assign_course(course);
  private:
   ...
   }
```

속성은 C++의 멤버와 같은 문법을 가지며, 관계는 inverse라는 키워드를 새로 도입한 것이 기존의 C++과 다르다. 1 : 1 관계를 위해 Ref〈T〉 타입을, 1:many 관계를 위해서는 Set〈T〉타입을 사용한다. 연산은 멤버 선언과 예외 처리 명세 등을 이용하여 가능하다.

5.2 C++OML

모든 지속적인 객체의 접근은 클래스 Ref〈T〉를 사용한다. 클래스 Ref〈T〉가 가리킬 수 있는 객체는 주로 클래스 Ref〈T〉의 객체보다 더 지속시간이 길거나 같은 객체이다. 더 지속시간이 짧은 객체는 오직 지속 시간 동안만 가리킬 수 있으며, 데이타베이스 상에서는 NULL로 표현된다. 만일 비존재 데이타 참조(dangling reference)문제가 발생하면, 예외 상황이 발생한다.

ODMG에서 C++바인딩시 지원하고자 하는 연산은 생성, 삭제, 갱신, 참조 등이다.

생성은 'new' 연산자에 선택사항으로 지속시간 인자를 덧붙임으로써 가능하다. 즉, 일반 C++에서는 다음과 같은 문법으로 포인터를 하나 생성하지만,

```
Ref〈Schedule〉 temp_sched1=new Sche-
dule; // trasient
```

다음과 같이 new의 괄호안에 생성 위치를 명시하기도 한다.

```
Ref〈Student〉 student1=
              new(myDB) Student //
              in myDB
Ref〈Student〉 student2=
              new(student1) Student //
              near student1
```

삭제는 Ref::destroy( ) 연산을 이용하며, 이것은 데이타베이스에서 제거됨을 의미한다. 삭제 후에는 인스턴스 값은 NULL이 되며, Ref의 값은 'fail'을 가리킨다.

지속적인 객체의 갱신은 실행시간에 ODBMS와 통신으로 이루어진다. 각 지속적 객체는 클래스 Pobject의 서브 클래스의 원소가 되며, 생성자, 파괴자나 다른 갱신용 함수에서는 반드시 Pobject::mark_modified( ) 함수 호출을 한다. 즉, 예를 들면, 다음과 같은 문장으로 나타내어진다.

```
obj_ref→mark_modified( );
```

실제로 데이타베이스가 갱신 되는 것은 트랜잭션이 정상 종료(commit)될 때이다.

객체 이름은 데이타베이스마다 단편적(flat)으로 사용되며, 클래스 데이타베이스에 정의되어 있는 연산들을 통해 다루어진다. 속성에서는 Ref〈T〉타입을 쓸 수 없다. 관계에서는 1 : 1 관계를 위해

```
Ref〈TargetClass〉 tPath.get(void) const;
void tPath.set(Ref〈TargetClass〉);
void tPath.delete(void);
```

등의 연산이 지원되며, 1: many 관계를 위해 다음과 같은 연산이 지원된다.

```
void tPath.insert(Ref〈TargetClass〉);
void tPath.remove(Ref〈TargetClass〉);
Iterator〈TargetClass〉
  tPath.create_iterator
  (boolean stable=FALSE);
void tPath.delete(void);
```

C++OML의 연산은 오버로딩, 재정의 등 C++멤버 함수에서 가능한 성질은 대부분 지원해야 한다.

C++OML 라이브러리에는 클래스 Ref〈T〉, Pobject 외에도 모임 타입을 지원하는 클래스들이 있다. 클래스 Collection은 템플릿 클래스로, 삽입/삭제/생성/원소 관계 검사/복사(shallow copy) 등이 지원된다. 그의 서브 클래스로, Set, List, Bag 등도 정의되어 있다. ODMG의 배열은 C++의 배열을 그대로 사용한다.

그 외에도, 템플릿 클래스 Iterator와, Transaction, Database 등의 클래스가 라이브러리에서 제공된다. 클래스 Transaction은 중첩 트랜잭션이 지원되야 하며, 클래스 Database는 가데이타베이스화-클래스가 아니며, 프로그래머가 인스턴스를 생성할 수 없다.

5.3 C++OQL

C++ 바인딩에서는 기존의 언어를 확장하지 않고도, 질의어 함수 Pobject::Query( ), oql( ) 등을 두어, 프로그램 내에서 OQL수행을 요구할 수 있도록 하고 있다. 다음은 그러한 함수의 예이다.

```
Ref〈Set〈Student〉〉 math;
Student->Query(math,
  "exists s in this.takes:s.section_of.name
  \"math\"");
oql(assisted_profs, "select t.assists.taught_
by—
  from t in TA where t.salary〉 $1r and t
  in $2k", x, mathmaticians);
```

향후에는 ODL에서 클래스 ref 대신 보통 C++의 포인터를 쓸 수 있도록 하고자 하며, object_collection=collection[predicate] 등과 같은 선언적인 직접 접근이 가능한 질의를 검토하고 있다.

Smalltalk 바인딩의 원칙도 C++바인딩과 비슷하다. 그러나, Smalltalk의 타입 시스템은 동적(dynamic)이기 때문에, 언어의 성질을 반영하여 결합하다보면, ODMG의 타입 관련 성질들을 포기해야만 한다. ODMG에서는 다른 언어로의 이식성을 위해서, 클래스 정의시 제약 조건(constraint)을 두고 컴파일러로 하여금 인식하게 한다.

Smalltalk 클래스 Class에는 키(key)를 추가/삭제하는 함수 add/remove가 첨가되고, 클래스 Object에는 지속적인 객체를 생성/변경하는 연산이 첨가된다. 서브클래스들의 이름은 광역(global)이며, 데이타베이스 기능을 가지는 클래스 Session이 첨가된다. Smalltalk OQL은 기존의 Smalltalk 바인딩 언어인 OPAL[5]의 영향을 많이 받았다. 기타 Smalltalk 바인딩에 관한 자세한 내용은 지면 관계상 생략한다.

6. 결 론

이상 살펴본 바에 따르면, ODMG 모델은 기본적인 데이타모델에 관한 사항으로부터, 스키마 정의 언어, 질의어, 프로그래밍 언어와의 바인딩에 이르기까지 많은 부분을 다루고 있다.

그리고, 키(key)나 관계(relationship) 등 관계형 데이타베이스의 성격을 살리려는 의도가 있었다. 이것은 기존의 관계형 데이타베이스 사용자의 친숙도를 높이려는 것이다.

ODMG 표준안은 적어도, 여러 모델들이 혼미한 현재 객체지향 데이타베이스 계에 지침이 될 수 있다는 점이 가장 중요하다 하겠다. 물론 지적되는 문제들도 여러가지 있지만, 이러한 점들은 앞으로 계속되는 연구로 극복될 수 있다고 본다. 궁극적으로 ODMG 표준안은 앞으로 더욱 개선되어 가면서, 객체지향 데이타베이스의 실용화에 커다란 기여를 할 것이라고 생각된다.

참고문헌

[1] R. G. G. Cattell Ed. *The Object Database Standards: ODMG-93,* Morgan Kaufmann Publishers, 1993.

[2] Won Kim. *Introduction to Object-Oriented Databases*, The MIT Press, 1990.

[3] M. E. S Loomis, *The ODMG object model*, JOOP, June, 1993.

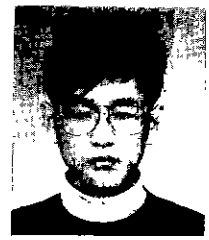
[4] Soley, R. M., Ed. *Object Management Architec-*

ture Guide, Rev. 2.0, 2nd Ed., OMG TC Document 92. 11. 1, Object Management Group, 1992.
[5] Robert Bretl *et. al, The Gemstone Data Management System*, Object-Oriented Concepts and Applications, ACM Press, Won Kim and Frederick H. Lochovsky 1989.
[6] Bjarne Stroustrup, The C^{++} *Programming Language , 2nd Ed.* Addison Wesley, 1991.

조 은 선

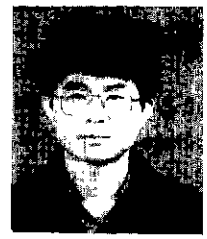
1991 서울대학교 계산통계학과(이학사)
1993 서울대학교 계산통계학과 전산과학전공(이학석사)
1993 ~현재 서울대학교 계산통계학과 전산과학전공 박사과정
관심 분야 : 데이타베이스, 프로그래밍 언어, 객체지향 시스템

이 강 우



1991 서울대학교 계산통계학과(이학사)
1993 서울대학교 계산통계학과 전산과학전공(이학석사)
1993 ~현재 서울대학교 계산통계학과 전산과학전공 박사과정
관심 분야 : 데이타베이스 트랜잭션, 객체지향 시스템

이 상 원



1991 서울대학교 컴퓨터공학과(공학사)
1994 서울대학교 컴퓨터공학과(공학석사)
1994 ~현재 서울대학교 컴퓨터공학과 박사과정
관심 분야 : 데이타베이스 보안, 객체지향 시스템