C++ 객체 영속성 부여를 위한 이음새 없는 인터페이스의 설계 및 구현

(Design and Implementation of Seamless Interface Providing Persistence to C++ Object)

이 미 영 † 김 명 준 ††
(Mi Young Lee) (Myung Joon Kim)

요 약 객체지향 언어 C++와 데이타베이스 관리 시스템의 결합은 C++ 응용 프로그램의 객체들이 프로그램이 종료되어도 지속되도록 데이타베이스의 저장 기능을 이용함과 동시에 데이타베이스 시스템에서 제공하는 다양한 기능을 사용하면서, 객체에 대한 조작은 C++의 일반 임시 객체처럼 C++에서 제공하는 기능들을 이용할 수 있도록 해 준다.

본 논문에서는 C++ 응용 프로그램의 객체에 영속성을 부여하는 방법으로 객체지향 데이타베이스 시스템의 상용 표준안인 ODMG-97의 C++ 바인딩에서 제시한 인터페이스를 보완하여 영속성 부여 인터페이스를 제시하고 이를 지원하기 위한 시스템의 설계 및 구현 내용을 서술한다. 제시한 인터페이스는 영속 가능한 클래스의 모든 객체는 동일한 인터페이스를 이용하여 사용자가 원하는 대로 임시 객체, 영속 객체를 모두 생성할 수 있고, 생성되는 객체의 클래스명을 추가로 명시하지 않고도 영속 객체를 생성할 수 있게 함으로써 이음새 없는 영속성 부여 인터페이스를 제공하고, 또한 객체 생성시 데이타베이스 객체와 메모리 객체간의 타입 호환성을 보장한다.

Abstract Binding the object-oriented programming language C++ with a database management system provides a persistency to C++ programming objects so that objects can persist after program termination. In such a binding system, we can manage a persistent object same as that we use a transient object and also use database management facilities such as transaction management and concurrency control.

This paper presents a method providing the persistency to C++ programming objects in the binding system. We propose an improved interface based on C++ binding of ODMG-97 and present the design and implementation technique of it. The proposed interface provides a seamless interface for creating objects of the persistent capable class. We can create a persistent object without its class name as we do not give a class name when creating a transient object. Also, we guarantee the type compatibilty between the object created in database and the object created in main memory.

1. 서 론

객체지향 프로그래밍 시스템의 객체에 영속성을 부여하는 방법으로 가장 초보적인 방법은 영속성을 부여하고자 하는 프로그램 객체에 대해 사용자가 파일 시스템이나 데이타베이스에 저장 요구를 호출함으로써 영속성을 부여하는 방법이 있다. 이 방법은 사용자 입장에서 파일 시스템이나 데이타베이스에 객체를 저장할 때와 저장된 객체를 사용하기 위해 적재하는 책임을 지고 이에 대해 조작을 해 주어야 하므로 자료의 일관성 및 사용에 부담을 준다. 그러므로 시스템에서 자동으로 객체를 저장 및 추출해 주는 시스템에 대한 요구가 제기되게 된다.

데이타베이스 시스템에서는 멀티미디어의 발달과 함

† 정 회 원 : 한국전자통신연구원 컴퓨터소프트웨어기술연구소 연구원
mylee@etri.re.kr
†† 종신회원 : 한국전자통신연구원 컴퓨터소프트웨어기술연구소 소장
joonkim@etri.re.kr
논문접수 : 1998년 12월 3일
심사완료 : 2000년 3월 7일

께 새로운 응용 분야가 생김에 따라 기존의 관계 자료형 모델의 한계를 느끼고 새로운 모델, 즉 객체지향 자료 모델에 대한 요구가 증가하게 되었다. 객체지향 프로그래밍 시스템의 객체에 대한 영속성 부여 요구와 데이타베이스 시스템에서의 객체지향 자료 모델 요구가 통합되면서 새로운 데이타베이스 인터페이스인 프로그래밍 언어 통합 방식이 생기게 되었다[1]. 프로그래밍 언어 통합 방식은 기존의 관계형 데이타베이스 시스템에서 데이타베이스 언어 SQL[2]을 주언어 프로그램에 내장하여 사용하던 방식의 문제점인 의미상의 불일치 문제를 해결해 주고 있다.

객체지향 언어 C++와 데이타베이스 관리 시스템의 결합은 C++ 응용 프로그램의 객체들이 프로그램이 종료되어도 지속되도록 데이타베이스의 저장 기능을 이용함과 동시에 데이타베이스 시스템에서 제공하는 다양한 기능(동시성 제어 기능, 회복 기능, 질의 기능 등)을 사용하면서, 객체에 대한 조작은 C++의 일반 임시 객체처럼 C++에서 제공하는 기능들을 이용할 수 있도록 해준다. C++ 응용 프로그램의 객체를 자동으로 저장 및 추출하기 위해서는 시스템에 이와 같은 작업이 필요한 객체라는 사실을 알려 주어야 하며 이를 알려 주는 방법에 따라 사용의 용이성 및 시스템의 이식성에 차이가 난다[3,4].

본 논문에서는 C++ 응용 프로그램의 객체에 영속성을 부여하는 방법으로 객체지향 데이타베이스 시스템의 상용 표준안인 ODMG-97의 C++ 바인딩[5]에서 제시한 인터페이스를 보완하여 영속 가능한 클래스의 모든 객체는 동일한 인터페이스를 이용하여 사용자가 원하는 대로 임시 객체, 영속 객체를 모두 생성할 수 있고, 생성되는 객체의 클래스명을 추가로 명시하지 않고도 영속 객체를 생성할 수 있으며, 생성시 C++ 응용 프로그램 메모리 객체와 데이타베이스 객체간의 타입 호환성을 보장할 수 있는 인터페이스를 제시하고, 이를 지원하기 위한 시스템의 설계 및 구현 내용을 서술한다.

2. 기존 연구

2.1 프로그래밍 언어 객체와 영속 객체

C++ 객체지향 프로그래밍 시스템과 데이타베이스 시스템을 통합하여 C++ 객체에 영속성을 부여하는 방법은 크게 C++ 언어 자체를 확장하여 영속 객체임을 알려 주는 방법과 영속성을 제공하는 클래스를 정의하고, 그로부터 유도된 하위 클래스들의 객체에 한하여 영속성을 부여하도록 영속 클래스 라이브러리를 제공하는 방식이 있다.

C++ 언어 확장 방식으로 대표적인 시스템이 AT&T에서 개발한 ODE라는 시스템이 있다[6]. ODE에서는 C++ 를 확장한 O++라는 인터페이스를 제공하는데, O++에서는 다음의 예와 같이 persistent라는 키워드에 의해 저장 공간을 정의하며, pnew라는 새로운 연산자를 통해 영속 객체를 생성할 수 있도록 하며, pdelete라는 새로운 연산자에 의해 영속 객체를 삭제하도록 하고 있다[6].

```
persistent Item *pip;
.....
pip = pnew Item(initial-values);
.....
pdelete pip;
```

이와 같은 C++ 언어 확장 방식은 타입과 직교적으로 객체에 영속성을 부여할 수 있다는 장점이 있으나 일반 C++ 프로그래밍 환경에서 동작하기 위해서는 전처리기나 특수 컴파일러가 제공되어야 하므로 이식성에 문제가 있다[3,4].

영속 클래스 라이브러리 방식은 영속성을 제공하는 클래스를 정의하고, 그로부터 유도된 하위 클래스들의 객체에 한하여 영속성을 부여하는 것으로, 영속 클래스에 정의된 인터페이스를 이용하여 영속 객체를 사용할 수 있도록 한다. 이 방법은 프로그램 언어를 확장하지 않으므로 응용 프로그램들이 표준 C++ 언어를 그대로 사용하여 데이타베이스 기능을 활용할 수 있으므로 응용 프로그램의 이식성면에서 좋지만, 특정 클래스에 속하는 객체에 한하여 영속성을 부여하므로, 타입과 영속성이 직교적이지 못하다는 단점이 있다. 이와 같은 단점을 보완한 영속 객체 계승 방법도 제안되고 있다[7]. 그러나 실세계에서 데이타베이스 시스템내에 클래스가 미리 모형화되지 않은 상태에서 객체를 저장하는 일은 없으므로, 크게 문제가 되지 않는다. 그러므로 현재는 C++ 언어 확장 방식보다는 영속 클래스 라이브러리 방식을 선호하고 있어 많은 상용 객체지향 데이타베이스 시스템들이 이 방식에 따라 영속 객체를 지원하고 있다[8,9,10].

2.2 ODMG 인터페이스

영속 클래스 라이브러리 방식에서는 영속 객체와 임시 객체를 얼마나 인식 없이 사용할 수 있느냐에 따라 사용의 용이성이 달라진다. 현재 ODMG-97의 C++ 결합에서는 영속 가능 클래스를 정의하고 영속 가능 클래스에 속하는 객체로 임시 객체, 영속 객체 모두 생성 가능한 다음과 같은 인터페이스를 제안하고 있다[5].

d_Object이라는 영속 가능 클래스를 정의하고, 이 클

래스로부터 계승받은 클래스의 객체들만 영속 객체가 가능하며, 영속 객체가 되기 위해서는 객체 생성시 영속 객체가 생성되도록 요구하여야 한다. 영속 객체 생성 요구를 일반 C++ 객체 생성시 new()를 이용하는 것과 같은 방법으로 사용할 수 있도록 d_Object의 메소드로 연산자 new()를 다음과 같이 오버로드하여 제공하고, 또한 영속 객체의 삭제를 위해 연산자 delete()를 오버로드하여 제공한다.

```
class d_Object {
public :
   void * operator new(size_t size);                    // (1)
   void * operator new(size_t size, const d_Ref_Any
          &cluster, const char *typename);     // (2)
   void * operator new(size_t size, d_Database
          *database, const char *typename);  // (3)
   void   operator delete(void *);
};
```

new 인터페이스 (1)번은 일반 new 연산자로 임시 객체를 생성하는 인터페이스이고, new 인터페이스 (2)번은 매개 변수로 전달하는 cluster(실제 생성될 객체의 데이타베이스내 저장 위치에 대한 정보)의 존재 여부에 의해 객체의 영속성이 결정되고 있다. new 인터페이스 (3)번에서는 database 변수 타입에 transient_memory라는 static 변수를 정의하고 이 값에 의해 영속 객체, 임시 객체를 생성하고 있다.

객체 생성 인터페이스를 사용하는 예를 보면 다음과 같다.

```
d_Ref<employee>  *emp_pobj, *emp_tobj;
d_Ref<manager>   *man_pobj, *man_tobj;
d_Database *myDB;// 값이 정의되어 있다고 가정
emp_tobj = new employee;                          // (1)
man_tobj = new manager;                           // (2)
emp_pobj = new(myDB, "employee") employee;// (3)
man_pobj = new(myDB, "employee") manager; // (4)
```

상기 예에서 (1), (2) 번은 임시 객체를 생성하는 것으로 각각 employee와 manager 임시 객체를 생성하고, (3), (4)번은 영속 객체를 생성하는 것으로 (3)번에서는 데이타베이스에 employee 객체를 생성하고, 메모리에도 employee 객체를 생성하는 반면, (4)번 예에서는 데이타베이스에는 employee 객체를 생성하고 메모리에는 manager 객체를 생성한다.

ODMG-97 인터페이스에서는 임시 객체를 생성할 때는 new 연산자 다음에 나오는 클래스명을 이용하여 생성될 객체의 타입을 결정하는데 반해, 영속 객체 생성을 위한 new 연산자는 매개 변수로 typename을 요구하고 있으며, 이 typename에 의해 데이타베이스에 생성할 객체의 타입을 결정하고 있다. 생성될 객체의 타입 선택이 영속 객체와 임시 객체 생성시 다르게 적용되고 있어 사용시 불편하며, 사용자에게 개념적인 혼돈을 줄 수 있다. 이것은 데이타베이스에 객체를 생성하기 위해서는 클래스명을 알아야 하는데 C++에서는 객체가 속한 클래스에 대한 정보를 실행시에 얻을 수 있는 방법이 제공되고 있지 않기 때문이다.

또한 ODMG-97 인터페이스를 이용시에는 (4)번 예와 같이 typename과 new 연산자 다음에 나오는 클래스명이 다를 수 있으므로 메모리에 생성되는 객체와 데이타베이스에 생성되는 객체 타입간의 호환성에 대한 점검이 필요하다. (4)번 예에서 employee가 manager 클래스의 하위 클래스라면 실제 데이타베이스에 생성된 employee 객체를 C++ 응용 프로그램에서 manager 클래스의 모습으로 타입 변환하여 사용하여도 문제가 없지만 두 클래스간에 계승 관계가 없거나, 있어도 employee가 manager의 상위 클래스라면 C++ 응용 프로그램 동작시 문제가 발생할 수 있다. 그러므로 객체 생성시 두 타입간에 호환성 점검을 실시하여야 하는데 생성되는 시점에 메모리 객체의 클래스 정보를 알 수 없으므로 호환성 점검에 어려움이 있다.

그러므로 본 논문에서는 상기와 같은 문제점을 해결한 C++ 바인딩 시스템에 대하여 서술한다. ODMG-97의 C++ 바인딩에서 제시한 인터페이스를 보완하여 typename을 추가로 명시하지 않고도 영속 객체를 생성할 수 있는 인터페이스를 설계한다. 보완된 인터페이스에서는 임시 객체, 영속 객체 모두 연산자 new 다음에 나오는 클래스명에 의해 객체를 생성하므로 개념상의 혼돈이 없고, 데이타베이스 객체와 메모리 객체간의 호환성 점검이 필요하지 않다. 그리고 typename이 명시된 경우에도 생성되는 데이타베이스 객체와 메모리 객체간의 타입 호환성 점검을 실시하는 C++ 바인딩 시스템의 설계 및 구현 내용에 대하여 서술한다.

3. 바다-III 객체지향 데이타베이스 시스템

3.1 시스템 구성

바다-III[11]은 전문 정보 검색 기능이 통합된 객체지향 데이타베이스 관리 시스템으로 그림 1과 같이 구성된다.

C++ 객체에 영속성을 부여하는 바다-III/C++[4],객체지향 자료 모델을 지원하는 바다-III/OK[12], 자료 저장 시스템인 바다-III/MiDAS[13]로 구성된다. 그리

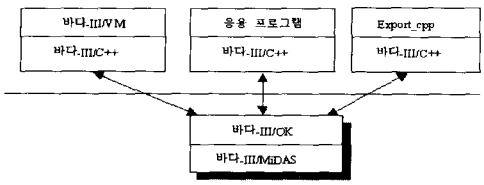

그림 1 바다-III 시스템 구성도

고 C++ 객체에 영속성을 부여하기 위해 필요한 인터페이스가 정의된 헤더 파일을 생성해 주는 유틸리티 export_cpp와 데이타베이스 구축을 도와주는 데이타베이스 관리자 도구인 바다-III/VM[14]으로 구성된다. 영속 객체를 생성하는 C++ 객체지향 응용 프로그램은 클래스 라이브러리로 제공되는 바다-III/C++와 통합되어 동작한다.

바다-III에서 지원하는 객체지향 자료 모델은 다음과 같다.

- 객체 : 모든 객체는 객체 식별자를 갖는다. 객체 식별자는 유일하며 변경 불가능한 것으로 객체의 소멸과 함께 없어지며 다시 사용되지 않는다.
- 복합 객체 : 객체는 다른 객체와 관계를 가짐으로써 복합 객체를 형성할 수 있다. 지원하는 관계는 has-a 관계로, 내포되는 객체가 내포하는 객체와 무관하게 존재할 수 있다.
- 클래스 : 타입 정의로서의 의미와 같은 타입을 갖는 객체들의 모임인 클래스 익스텐트의 의미를 갖는다. 타입은 객체의 정적인 특성을 나타내는 속성과 객체의 동적인 특성을 나타내는 메소드로 표현된다. 속성은 이름, 도메인으로 구성되고, 메소드는 메소드 인터페이스와 메소드 구현으로 구성되며, 메소드 오버로딩을 지원한다.
- 계승 : 단일 계승을 지원하고, 타입을 계승한다. 타입 계승은 속성과 메소드를 모두 계승하는 것이다. 메소드의 구현, 속성 이름이나 속성 타입은 계승받은 클래스에서 재정의할 수 있다. 클래스는 속성과 메소드로 표현한다.

3.2 바다-III/C++

바다-III/C++는 C++ 응용 프로그래밍 인터페이스로 데이타베이스 시스템을 사용하는데 필요한 기능인 ODL(Object Definition Language), OML(Object Manipulation Language), OQL(Object Query Language)과 환경 관리에 해당되는 인터페이스를 다음과 같이 16개의 인터페이스 클래스의 메소드와 일반 함수로 제공하고 있다.

- ODL 지원 인터페이스 : OM_PSCHEMA, OM_PCLASS, OM_PRIVILEGE
- OML 지원 인터페이스 : OM_BLOB, OM_CLOB, OM_SET<T>, OM_MULTISET<T>, OM_LIST<T>, OM_POBJECT, OM_REF<T>
- OQL 지원 인터페이스 : OM_ITERATOR<T>, OM_IR_ITERATOR<T>, OM_IR_HIT_LIST<T>, query()
- 사용 환경 및 트랜잭션 지원 인터페이스 : OM_ENVIRONMENT, OM_ERROR, OM_TRANSACTION

4. 이음새 없는 C++ 영속 객체 관리 인터페이스

바다-III/C++ 인터페이스 클래스중 프로그래밍 언어 C++ 객체에 영속성을 부여하고, 영속 객체를 관리하는 것은 OML에 해당되는 OM_POBJECT, OM_REF<T> 클래스로 이음새없는 인터페이스를 제공한다.

4.1 영속성 부여

C++ 응용 프로그램의 객체에 영속성을 부여하는 인터페이스로 상용 표준안인 ODMG-97의 C++ 바인딩에서 제시한 d_Object 인터페이스를 보완하여 다음과 같은 인터페이스를 제시한다. OM_POBJECT은 객체의 생성, 삭제, 객체의 변경 여부, 영속 객체 여부 및 객체가 변경되었음을 알려주는 인터페이스 등을 제공한다.

```
class OM_POBJECT {
public :
  void * operator new(size_t size);                    // (1)
  void * operator new(size_t size, int persistent)
                                                       // (2)
  void * operator new(size_t size, int persistent,
         const char *typename);                        // (3)
  void   operator delete(void *);
  void   mark_modified();
};
```

** new 연산자에 명시된 size 정보는 컴파일러에 의해 생성되는 정보이므로 본 논문에서는 이후 이를 생략한 인터페이스를 사용하기로 함.

delete 연산자는 메모리내 객체 삭제 뿐만 아니라 영속 객체도 삭제하는 인터페이스이고, new 연산자 (1)번은 임시 객체를 생성하는 new 연산자이고, new 연산자 (2)번은 persistent에 의해 객체의 영속 여부를 결정하고, 생성되는 객체의 클래스는 new 연산자 다음에 나오는 클래스 명에 의해 결정한다. new 연산자 (3)번은

persistent에 의해 객체의 영속 여부를 결정하고, 생성되는 객체의 클래스는 명시된 매개 변수 typename에 의해 일차 결정하고, 잘못된 클래스명이면 new 연산자 다음에 나오는 클래스명에 의해 결정한다. (1)번과 (3)번 new 연산자는 ODMG-97에 정의된 인터페이스에서 사용하는 cluster, database 대신 persistent를 사용한 것으로, 이는 실제 데이타베이스 객체를 생성할 때 서버의 지원 범위에 연동되는 것으로 C++ 인터페이스 입장에서는 개념적으로 동일하다.

제시한 객체 생성 인터페이스는 영속 가능한 클래스의 모든 객체는 동일한 인터페이스를 이용하여 사용자가 원하는 대로 임시 객체, 영속 객체를 모두 생성할 수 있고, new 인터페이스 (2)번을 이용하면 생성되는 객체의 클래스명을 추가로 명시하지 않고도 영속 객체를 생성할 수 있으므로 개념상의 혼돈이 없고 생성된 데이타베이스 객체와 메모리 객체간의 타입 호환성을 자동으로 보장한다.

4.2 영속 객체 참조

C++ 응용 프로그램에서 영속 객체를 포인팅할 때 이용되는 영속 객체 참조 인터페이스인 OM_REF<T>는 ODMG-97의 d_Ref<T>에서 정의된 인터페이스와 마찬가지로 스마트 포인터가 가리키는 객체를 얻는 메소드, 포인팅 객체의 삭제 혹은 포인팅 객체의 삭제 여부, 메모리내 적재 여부 및 포인팅 객체의 동일 여부 등에 대하여 알려 주는 인터페이스를 제공한다. 대표적인 인터페이스는 다음과 같다.

```
Template <class T> class OM_REF {
public :
    d_Ref<T>& operator = (T *);// 객체 포인팅
    T* operator ->();// 포인팅하고 있는 객체 주소 제공
    T& operator *();// 포인팅하고 있는 객체 제공
    void delete_object(); // 객체 삭제
}
```

5. C++ 영속 객체 바인딩 시스템 설계 및 구현

5.1 주요 기능

C++ 바인딩 시스템에서 영속 객체를 관리하기 위해서는 다음과 같은 기능이 지원되어야 한다.

- 객체에 영속성 부여
- 영속 객체 참조
- 메모리내 적재된 객체 캐쉬 관리
- 메모리 객체와 데이타베이스 객체간의 형태 변환

가. 객체에 영속성 부여

제시된 인터페이스를 통해 C++ 응용 프로그램에서 영속 객체를 생성하는 new()를 호출하면 메모리 객체 생성 뿐만 아니라 프로그램 종료 후에도 지속되도록 데이타베이스에 객체를 생성하여야 한다. 데이타베이스에 객체를 생성한다는 것은 생성될 객체에 대한 클래스 정보를 알아야 한다는 의미이다. 그러나 제시한 (2)번 new 연산자는 클래스명을 입력 받지 않고 있으므로, 이에 대한 정보를 달리 얻어 이를 이용하여 데이타베이스에 객체를 생성한다. 그리고 typename이 명시된 인터페이스를 사용한 경우에는 데이타베이스 객체와 메모리 객체간의 타입 호환성을 점검해 주어야 한다.

나. 영속 객체 참조

OM_REF<T>::operator->()를 통해 객체를 참조하면 객체 캐쉬에 이미 적재된 객체인지 확인하여 적재된 객체이면 해당 객체를 얻어 OM_REF<T>가 포인팅하도록 연결해 주고, 객체 캐쉬에 적재된 객체가 아니면 데이타베이스로부터 객체를 적재하여 객체 캐쉬에 등록하고 이를 OM_REF<T>가 포인팅하도록 연결한다. 객체를 연결 시에는 C++에서 요구하는 객체 모습으로 포인팅하여야 하므로 현재 적재되어 있는 객체 모습에 대한 파악 및 요구되는 객체 모습에 대하여 파악하여 이를 적절히 변환하여 준다.

다. 객체 캐쉬 관리

메모리에 적재된 객체는 객체 캐쉬에 관리된다. 이는 같은 객체를 클라이언트 프로그램 여러 곳에서 참조해도 하나의 객체로 존재함으로써 객체의 일관성 유지 및 메모리 낭비를 줄일 수 있고, 메모리에 이미 적재된 객체라면 다시 서버에게 요구하지 않아도 되므로 서버와의 통신 비용 및 서버의 I/O 수행 등 부담을 감소시킨다. 객체 캐쉬에 객체를 관리 시에는 메모리 객체와 데이타베이스 객체간의 일관성 유지가 중요하므로 이를 고려하여 캐쉬의 내용을 서버에 반영(flush) 또는 반납(free)하는 정책을 수립한다[4].

라. 객체 변환

데이타베이스에는 저장 장치의 효율성 및 특성을 고려하여 바이트 스트림으로 저장되므로 메모리에서 이를 사용하기 위해서는 이를 메모리 객체로 변환하여야 한다. 이와 같은 변환은 객체의 클래스 구조를 참조하여 속성별로 메모리내 해당 위치로 속성 값을 복사하거나, 메모리내 속성 값을 레코드내 해당 위치에 복사하는 방식으로 수행한다. 이 변환 메소드는 계승 관계에 있는 클래스간에는 하위 클래스를 상위 클래스로 볼 수 있으므로 이를 고려하여 메모리 객체에 해당되는 변환 메소

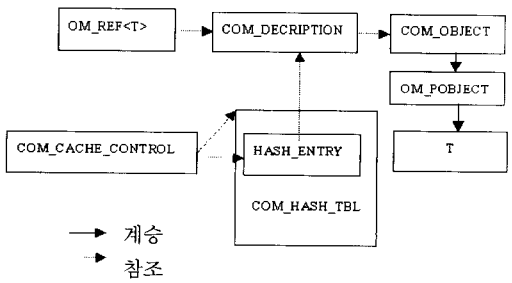

그림 2 영속 객체 관리 클래스간의 관계도

드를 이용하면서도 각 변환 메소드에서는 현재 변환하는 실제 저장 객체의 클래스 구조를 참조하여 변환한다 [4].

5.2 클래스 설계

C++ 영속 객체를 관리하기 위해 사용되는 클래스들의 구조 및 관계는 그림 2와 같다.

- COM_OBJECT : C++ 바인딩 시스템의 내부 구현 클래스로 실제 영속성을 관리
- OM_POBJECT : 영속성을 부여하는 C++ 인터페이스 클래스로 COM_OBJECT으로 부터 계승받은 클래스
- T : 데이타베이스에 정의되어 있는 사용자 정의 클래스로 OM_POBJECT으로 부터 계승받은 클래스
- OM_REF<T> : 영속 객체 참조 인터페이스 클래스로 COM_DESCRIPTION 클래스를 통해 영속 객체 포인팅
- COM_DESCRIPTION 클래스 : 객체 디스크립터로 객체 캐쉬내의 객체를 포인팅
- COM_HASH_TBL : 객체 디스크립터를 빠르게 접근하기 위해 사용하는 해쉬 테이블
- HASH_ENTRY : 키 값에 의해 해당 객체 디스크립터인 COM_DESCRIPTION를 가리키는 해쉬 엔트리
- COM_CACHE_CONTROL : 객체 캐쉬를 제어하는 클래스

5.3 영속성 부여 알고리즘

본 절에서는 영속 객체 관리 기능중에서 영속성 부여 기능의 설계에 대해서 중점적으로 설명한다. typename을 명시하지 않는 new 연산자를 지원하기 위해 생성할 객체의 타입을 얻는 방법과 typename을 명시한 경우에 실제 데이타베이스에 생성되는 객체와 C++ 프로그래밍 시스템에 의해 메모리내에 생성되는 객체 타입간의 호환성을 유지하는 문제에 대해 서술한다.

C++에서는 new()가 호출되는 시점에 클래스에 대한 정보를 얻을 수 없고 오직 메모리만 할당하므로 어떤 클래스에 객체를 생성할 지 알 수 없다. 클래스 명을 제공하는 메소드를 클래스의 멤버 함수로 정의한다 하더라도 객체가 생성된 후에나 객체가 속한 클래스에 정의된 메소드를 호출할 수 있으므로 new() 호출시 활용할 수 없다.

그러므로 본 논문에서는 메소드 중 가장 먼저 호출되는, 즉, 사용자 입장에서는 객체 생성시 호출되는 메소드인 생성자(constructor) 함수를 이용하여, 클래스에 대한 정보를 얻어 new() 호출시 메모리 객체뿐 만 아니라 데이타베이스에도 객체를 생성할 수 있게 하는 방법을 제안한다. 오버로드된 연산자 new()의 구현을 영속성 부여에 필요한 사전 작업을 하도록 정의하고, 영속 가능 클래스 OM_POBJECT에서 유도된 사용자 정의 클래스 T의 생성자 함수는 생성되는 클래스 정보를 알 수 있도록 구현을 정의한다. 필요한 생성자의 인터페이스와 구현 내용은 export_cpp 유틸리티에 의해 자동으로 생성해 준다.

그리고 메모리 객체와 데이타베이스 객체 타입간의 호환성 문제는 typename을 명시하는 new 인터페이스에서만 발생되는 문제로 생성자를 상기와 같이 정의하면 두 타입에 대한 정보를 모두 얻을 수 있으므로 이를 이용하여 타입간의 호환성을 점검한다.

가. 클래스 정의

영속성 부여에 관련되는 클래스는 COM_OBJECT, OM_POBJECT, T클래스로 각 클래스의 관계는 그림 3과 같다.

COM_OBJECT 클래스는 영속성 관리에 필요한 정보들을 저장할 속성들과 영속성 관리에 필요한 메소드들로 구성된다. 메소드는 생성자와 재정의된 연산자와 일반 메소드로 구분된다. 영속성 부여와 관련이 있는 메소드는 생성자와 재정의된 new 연산자이므로 이후 일반 메소드에 대해서는 생략한다. 영속성 부여와 직접적으로 관련이 있는 속성은 Persistent_flag, Object_id, Disk_format, Layout_info, Memory_flag이다.

- Persistent_flag : 객체의 영속 여부를 나타내는 플래그
- Object_id : 객체 식별자
- Disk_format : 데이타베이스에 저장되어 있는 객체 모습을 나타내는 버퍼
- Layout_info : 객체가 속하는 클래스 구조 정보
- Memory_flag : 현재 객체 모습이 응용 프로그램에서 사용되는 구조체 모습으로 변경되었는지 여부를 나타내는 플래그

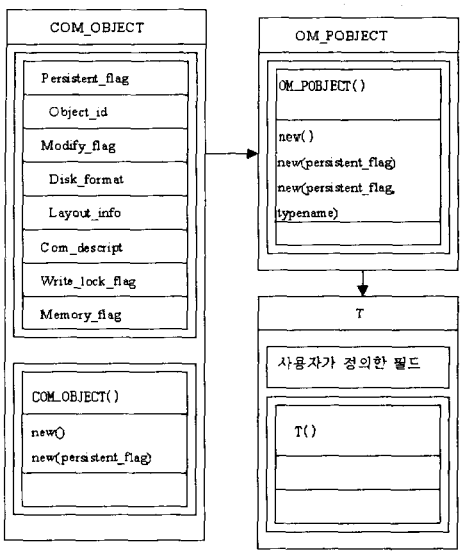

그림 3 클래스 구조

그밖에 응용 프로그램 동작 환경인 클라이언트내에 객체 캐쉬를 관리하며, 응용 프로그램에서 영속 객체에 접근하는 것을 제어하는데 필요한 정보들로 구성된다.

OM_POBJECT 클래스는 COM_OBJECT 클래스를 계승받은 클래스이므로 COM_OBJECT 클래스의 속성과 재정의된 new 연산자를 계승받는다(C++에서 생성자는 특별한 메소드로 계승 대상이 아니므로 계승되지 않는다). OM_POBJECT에는 자체내에 정의한 속성들은 없고, 영속성 관리를 위하여 생성자와 제시한 영속 객체 생성을 위한 new 연산자를 정의하고 있다.

T 클래스는 OM_POBJECT 클래스를 계승받은 클래스이므로 OM_POBJECT 클래스의 속성과 OM_POBJECT 클래스 자체내에서 재정의된 new 연산자를 계승받고 COM_OBJECT 클래스의 속성을 계승받는다. 그리고 T 클래스 자체에 사용자가 정의한 속성과 두 개의 생성자가 있다. 이 생성자는 영속성 관리에 필요한 메소드로 해당 클래스마다 export_cpp 유틸리티에 의해 헤더 파일에 생성된다.

나. 동작 알고리즘

C++ 프로그램 시스템에서 일반 프로그래밍 객체(즉, 임시 객체)를 생성하는 경우와 마찬가지로, T라는 클래스가 OM_POBJECT으로 부터 직접, 간접으로 유도된 영속 가능한 클래스라면, T의 영속 객체 생성은 OM_POBJECT에 정의된 연산자 new()와 T 클래스의 생성자에 정의된 절차에 따라 수행된다. 오버로드된 연산자 new()와 생성자의 세부 알고리즘은 다음과 같다.

1) 연산자 new

OM_POBJECT의 연산자 new는 각 인터페이스에 따라 typename이 명시된 경우에는 typename을 클래스 식별자로 변환하여 COM_OBJECT의 연산자 new를 호출한다.

OM_POBJECT::operator new(persistent, typename) 인터페이스는 연산자 new 다음에 클래스명을 OM_ POBJECT으로 사용하여 생성하는 경우도 가능하게 구현한다. 이렇게 함으로써 생성될 객체가 속하는 클래스의 정의문이 없이도 데이타베이스에 해당 객체를 생성할 수 있으므로 응용 프로그램 개발 도구와 같이 사전에 클래스 정의문을 얻을 수 없는 프로그램 개발시 유용하다.

COM_OBJECT의 연산자 new의 수행은 주메모리에 객체 영역을 할당하고, 영속 객체 여부에 대한 정보(persistent_flag)와 클래스 식별자(class_id)를 저장하여 이후 생성자가 동작시 이용할 수 있게 한다.

2) 생성자

영속 가능 클래스의 생성자인 T()와 T(classname)의 인터페이스와 구현은 export_cpp 유틸리티를 이용하여 클래스 정의문에 대한 헤더 파일을 생성시 자동으로 만들어 주므로 사용자는 전혀 의식없이 사용할 수 있게 한다. 각 클래스의 생성자 메소드에 대한 알고리즘은 다음과 같다.

```
COM_OBJECT::COM_OBJECT()
{
    initialize attribute except Persistent_flag and
      Object_id
}
OM_POBJECT::OM_POJBETC(const char *classname) :
    COM_OBJECT()
{
    get memory_object_class_id from classname
      parameter
    get database_object_class_id from Object_id field
    // check whether typename is given in new operator
    if database_object_class_id = VALID_VALUE
    then // typename given
            check type compatibility between database
                object and memory object
     else // typename not given
            database_object_class_id=memory_
                object_class_id
    create object in database
```

```
}
C::C(const char *classname): OM_POBJECT
  (classname)
{
    initialize attributes defined in C
}
T::T() : S("T")
{
    initialize attributes defined in T
}
```

1. COM_OBJECT() 생성자는 new 연산자에서 값이 대입된 Persistent_flag와 Object_id 자리를 제외한 나머지 영역을 초기화한다.

2. OM_POBJECT(classname)는 COM_OBJECT() 생성자를 호출 후 자체 생성자 내용을 수행한다. OM_POBJECT(classname) 자체 생성자에서는 Object_ id의 클래스 식별자를 확인하여 new 연산자 수행시 typename이 명시되었는지 확인한다. typename이 명시되지 않은 경우에는 주어진 classname을 이용하여 데이타베이스에 객체를 생성한다. typename 매개변수를 사용한 경우에는 typename 정보와 classname 정보를 이용하여 타입 호환성을 점검하고 유효하면 데이타베이스에 객체를 생성한다.

3. T() 생성자는 먼저 classname을 자신의 이름으로 지정하여 자신의 바로 상위 클래스에 대한 생성자 S(classname)을 호출하고 자체 생성자 내용을 수행한다. 상위 클래스는 또 다시 자신의 상위 클래스에 대한 C(classname) 생성자를 호출하고, 이것이 회귀적으로 호출되어 마지막에는 OM_POBJECT(classname)이 호출된다. 여기서 classname은 T 클래스 생성자이면 "T"가 되고, 최초의 클래스명, 즉, "T"가 마지막 OM_POBJECT의 생성자에까지 전달되어 최종적으로 T에 속하는 객체가 생성되게 한다.

5.4 구현 결과

바다-III/C++ 시스템은 C++ 언어로 구현되어 있으며, UNIX와 WINDOWS 환경에서 사용 가능하다. 현재 바다-III/C++ 인터페이스를 이용하여 전자 도서관, 게시판 관리 시스템 등 다양한 응용 프로그램이 개발되어 사용중이다.

바다-III/C++에서는 상기와 같이 설계된 연산자 new와 생성자를 이용하여 클래스명을 입력으로 주지 않고도 해당 클래스의 객체를 생성할 수 있는 시스템을 제공하고 있다. 이 시스템은 임시 객체를 생성할 때 사용하던 인터페이스와 같은 형태로 영속 객체를 생성할 수 있으므로 사용자에게 개념상의 혼돈없이 일관성있게 객체를 생성할 수 있게 하고, 데이타베이스 객체와 메모리 객체간의 타입 호환성을 보장한다. 또한 데이타베이스 객체와 메모리 객체의 타입 호환성 점검이 필요한 경우에도 생성될 타입에 대한 정보를 생성자 수행시 알고 있으므로 이를 이용하여 점검할 수 있기 때문에 객체 생성시 오류를 미리 방지할 수 있다.

그러나 이 방법에서는 사용자가 생성자를 정의하여 사용하기 위해서는 시스템에서 제공하는 생성자(T(), T(char *))와는 다른 인터페이스를 갖도록 정의하여야 하고, 또한 내부 구현시 항상 시스템에서 제공하는 기본 생성자와 같은 방식으로 상위 클래스의 생성자를 호출하도록 자체 생성자를 구현하여야 한다는 제약 사항이 따르고 있다.

6. 결 론

객체지향 데이타베이스 시스템과 객체지향 프로그래밍 시스템을 통합한 방식에 의해 C++ 응용 프로그램 객체에 영속성을 부여하는 시스템으로 ODMG-97의 C++ 바인딩 방식에서 제안한 인터페이스를 개선한 시스템을 설계 구현하였다. C++ 응용 프로그램에서 영속 객체를 생성할 때 일반 C++ 응용 프로그램 객체를 생성하는 개념과 같은 개념으로 사용할 수 있으며, 또한 객체가 속하는 클래스명을 명시적으로 주지 않고도 사용 가능하므로 사용자에게 사용의 용이성 및 일관성을 제공할 수 있다. 또한 객체 생성시 데이타베이스 객체와 메모리 객체간의 타입 호환성을 보장할 수 있다. 그리고 클래스 구조 정보에 대한 클래스 정의문이 없이도 영속 객체 생성이 가능하므로 클래스 정의문을 사전에 얻을 수 없는 응용 프로그램에서도 사용할 수 있다.

제안된 방법에서는 사용자가 클래스의 생성자를 정의하는 것에 약간의 제약이 간다는 단점이 있으므로 차후 이를 보완할 수 있는 방법에 대한 연구가 필요하다.

참 고 문 헌

[1] Won Kim, Nat Ballou, Hong-tai Chou, Jorge F. Garza, Darrell Woelk, "Integrating An Object-Oriented Programming System with a Database System," OOPSLA'88 Proceedings, Sept. pp.142-152, 1988

[2] ISO 9075-1992, "Information Processing System - Database Language SQL," ISO, 1992.

[3] 이미영, 채미옥, 김평철, 전성택, "바다-III/C++: 바다-III에서의 C++ 결합 방법", 한국정보과학회 '95 가을 학술발표논문집(A), 제22권 2호, pp.271-274, 1995.

[4] 이미영, 조옥자, 김영균, 전성택, "C++ 객체에 영속성을 부여하는 클래스 라이브러리의 설계 및 구현", 한국정보과학회, 정보과학회논문지(C), 제3권 제5호, pp.549-557, 1997.
[5] Cattell, R.G.G., The Object Database Standard: ODMG 2.0, Morgan Kaufmann, 1997.
[6] Rakesh Agrawal, Shaul Dar, Narain Gehani, "The O++ Database Programming Language: Implementation and Experience," Proc. IEEE 9th Int'l Conf. Data Engineering, pp.61-70, 1993.
[7] 박종목, 조완섭, 황규영, "C++ 기반 객체지향 데이타베이스 시스템에서 직교적 지속성을 제공하기 위한 강제계승방법", 한국정보과학회, 정보과학회논문지(B), 제22권 제 11호, pp.1510-1519, 1995.
[8] Deux, O., "The O2 System," CACM, 34(10), pp. 34-48, 1991.
[9] Lamb,C., "The ObjectStore Database System," CACM, 34(10), pp. 50-63, 1991.
[10] UniSQL Inc., UniSQL/X User's Manual, UniSQL Inc., 1995.
[11] 이미영, 허대영, 김명준, "바다-III 멀티미디어 DBMS 설계", SoftEXPO'97 Conf. pp. 193-200, 1997.
[12] Mi-Ok Chae, Ki-Hyung Hong, Mi-Young Lee, June Kim, Ok-Ja Cho, Sungtaeg Jun, Young-Kyun Kim, "Design of the Object Kernel of BADA-III:An Object-Oriented Database Management System for Multimedia Data Services," International Workshop on Network and System Management 1995, pp.143-152, 1995.
[13] 이진수, 박순영, 채미옥, 김준, 허대영, "MIDAS-II의 설계 및 구현", 한국정보과학회 '93 가을학술발표논문집, 제20권 2호, pp.183-196, 1993.
[14] 김완석, 이용현 정광철, 전성택, "바다-III/VM의 설계", 한국정보처리학회, '96 춘계 학술발표논문집, 제3권 1호, pp.660-665, 1996.
[15] 조옥자, 이미영, 전성택, "바다-III 객체 관리자의 설계 및 구현", 한국정보처리학회 '96 추계 학술발표논문집, 제3권 제2호, pp.161-166, 1996.
[16] Bjarne Stroustrup, The C++ Programming Language, Addison-Wesley Publishing Company, 1991.

이 미 영

1981년 서울대학교 식품영양학과(이학사, 계산통계학 부전공). 1983년 서울대학교 대학원 계산통계학과(이학석사). 1983년 ~ 1985년 한국전기통신연구소. 1988년 ~ 현재 한국전자통신연구원 컴퓨터소프트웨어기술연구소 선임연구원. 관심분야는 데이타베이스 시스템, XML 문서 관리, 객체지향 자료 모델링, 질의 처리 및 정보 검색.

김 명 준

1978년 서울대학교 계산통계학과 이학사. 1980년 한국과학기술원 전산학과 이학석사. 1986년 프랑스 Nancy 제1대학교 응용수학 및 전산학과 이학박사. 1980년 ~ 1981년 아주대학교 종합연구소 연구원. 1981년 ~ 1986년 프랑스 Nancy 전산학연구소(CRIN) 연구원. 1993년 프랑스 Univ. of Nice Sophia-Antipolis 방문교수. 1986년 ~ 현재 한국전자통신연구원 컴퓨터소프트웨어기술연구소 책임연구원. 2000년 ~ 현재 한국전자통신연구원 컴퓨터소프트웨어기술연구소 소장. 관심분야는 데이터베이스, 분산시스템, 인터넷 서비스.