□ 기술해설 □

응용 사례

객체지향 데이타베이스 시스템의 CIM 응용

인하대학교 유 상 봉*
서울대학교 차 상 균**

목 차

1. 서 론

CAD, CAE, CAM, 데이타베이스 시스템, 전문가 시스템 등의 많은 컴퓨터 시스템이 생산성 향상을 위하여 개발되어졌다. 각각의 기술들은 그 동안 많은 시행착오를 거쳐 이제 어느 정도 성숙한 단계에 도달했고 생산현장에서 사용되어 그 효과를 인정 받고 있다. 이러한 컴퓨터 응용 시스템 사용자들의 제일 큰 요구사항은 시스템 통합이다(CIM). 예를 들어 CAD로 제작된 설계를 컴퓨터 화일 형태로 CAE 시스템으로 전송하여 분석되고 그 결과가 CAD 시스템에 다시 피드백 될 수 있으며, 이러한 과정을 거쳐 완성된 설계를 갖고 CAM 시스템은 로보트나 NC 공작기계의 프로그램을 생성할 수 있어야 한다. 또한 이러한 시스템들이 데이타베이스 시스템이나 전문가 시스템들과도 데이타를 교환할 수 있어야 하겠다. 물론 현재에도 동일 회사 제품이거나 몇몇 대표적인 제품들간에 제한된 데이타 호환 기능을 갖추고 있지만 전 생산 시스템의 통합은 아직 많은 연구가 필요하다[1].

CIM은 생산 시스템의 통합을 위하여 생산 활동에 포함되는 모든 정보를 관리한다[2,3,4,5]. 이러한 정보의 종류에는 형상정보, 공정정보, 일정정보, 품질정보, 그리고 기존의 관리정보가 포함된다. 이러한 정보의 통합은 기존의 CAD/CAM 제품들이 공통으로 이용할 수 있는 화일 포맷을 제정하면서 시작되었다. 현재 널리 쓰이고 있는 것은 1981년 ANSI Standard로 제정된 IGES (Initial Graphics Exchange Specification)이다 [6]. 하지만 IGES의 정의 가운데 모호한 부분이 발견되어 ISO는 새로운 표준으로서 1983년에 STEP(STandard for the Exchange of Product model data)의 개발에 착수하였다[7]. STEP은 그래픽 데이타 뿐만 아니라 제품의 전 생명주기에 포함된 모든 데이타를 표현하기 위하여 개발중이며 기본적인 부분은 이미 국제 표준으로 결정되었다.

* 정회원
** 종신회원

STEP에 의하여 정의된 표준 데이타는 화일을 이용하여 교환할 수 있으며 데이타베이스 시스템에 저장하여 공유할 수도 있다. 이와 같이 다양한 저장 시스템(Repository Systems)을 접속하기 위하여는 응용프로그램에 각종 화일 시스템이나 데이타베이스 시스템을 접속하는 코드가 포함되어야 한다. 이러한 문제점을 해결하기 위하여 STEP에서는 데이타 인터페이스 표준 (Standard Data Access Interface, SDAI)[8]을 정하였다. 즉, 각 데이타베이스 시스템에서 SDAI를 구현 하였을 때 응용프로그램은 데이타베이스 시스템의 종류에 관계없이 미리 정의된 인터페이스를 이용할 수 있어 Open CIM을 실현한다.

본 논문에서는 객체지향 데이타베이스 시스템인 Objectivity/DB[9]를 이용하여 구현한 SDAI를 설명한다. 객체지향 데이타베이스 시스템은 엔지니어링 데이타베이스 시스템, 멀티미디어, 지리정보 시스템 등과 같이 다양한 정보의 형태가 필요하고 데이타의 조작이 복잡한 경우에 많이 활용되고 있다. 본 연구에서 객체지향 데이타베이스 시스템이 STEP의 구현에 효과적으로 응용될 수 있다는 것을 알 수 있는데 이는 STEP의 모든 정보 모델이 객체지향 모델링 언어에 의하여 정의되어 있기 때문이다. 관계형 데이타베이스 시스템을 이용할 경우 스키마의 전환이 복잡하며 그에 따라 전체 인터페이스 프로그램도 복잡해 진다. 이러한 차이점을 설명하기 위하여 관계형 데이타베이스 시스템인 ORACLE[10]을 이용한 SDAI의 구현을 비교하여 설명한다.

2. 제품데이타의 교환과 공유를 위한 표준 (STEP)

ISO의 TC184/SC4 (Industrial Automation Systems/Representation and Exchange of Digital Product Data)에서 개발중인 STEP은 여러기존 양식(IGES, SET, VDAFS 등)을 바탕으로 개발되고 있으며 다음과 같은 점들이 보강된다[11].

(1) 엔코딩 방식을 최적화하여 화일의 크기를 줄이고 기능을 강화한다.
(2) 엔티티 타입의 종류를 증가시킨다. 예를 들면, 자유형태의 곡면이나 3차원 객체를 첨가한다.
(3) 정보의 범위를 증가시킨다. 예를 들면, 생명 주기 데이타, 관리 데이타, 제어데이타 등을 첨가한다.
(4) 각종 랭귀지 바인딩(Language Bindings)을 개발하여 응용프로그램 인터페이스를 표준화한다.

2.1 STEP의 구조

STEP은 그 범위가 방대하여 부분에 따라 연구 진척에 상당한 차이가 있으며 어떤 부분은 벌써 IGES에서 수행 된 것도 있다. 따라서 STEP은 그림 1과 같이 여러 파트로 분류하여 개발되고 있다. 파트 21은 화일 양식을 정하고 파트 22는 SDAI의 기능과 C, C++, Fortran 등으로 쓰인 랭귀지 바인딩을 정한다. 파트 41부터 99까지는 여러 응용분야에서 공통적으로 쓰이는 일반적인 자원을 정의한다. 예를 들어, 각종 도형, 물질, 오차 등이 이에 속한다. 파트 101부터 200까지는 2차원 제도나 전기소자 등과 같은 전문적인 응용분야에서 쓰이는 자원을 표시한다. 파트 번호가 201 이상인 응용 프로토콜은 이미 정의된 자원을 이용하여 선박이나 자동차의 생산 같은 특정 응용분야에서 사용되는 정보의 형태, 의미, 관계 등을 정의한다. 결국 전문 분야의 응용 프로그램은 이러한 응용 프로토콜에 정의된 데이타를 생성하고 저장한다. STEP 인증을 위한 테스트 방법은 파트 번호 30번 대에서 정의되고, STEP의 모든 결과는 정보 모델링 언어인 EXPRESS[12]로 표현된다. EXPRESS에 관하여는 뒤에서 설명한다.

STEP에서 정의된 제품정보는 다음과 같은 3가지 방법을 이용하여 교환되고 공유될 수 있다.

(1) 레벨 1: Passive File Exchange
(2) 레벨 2: Active File Exchange
(3) 레벨 3: Shared Database

레벨 1의 Passive File Exchange는 응용프로그램 간의 정보교환이 배치(batch) 프로세스에 의하여 이루어지는 구성이다. 이것은 현재 IGES에 의한 방법과 유사하다. 그림 2에서 응

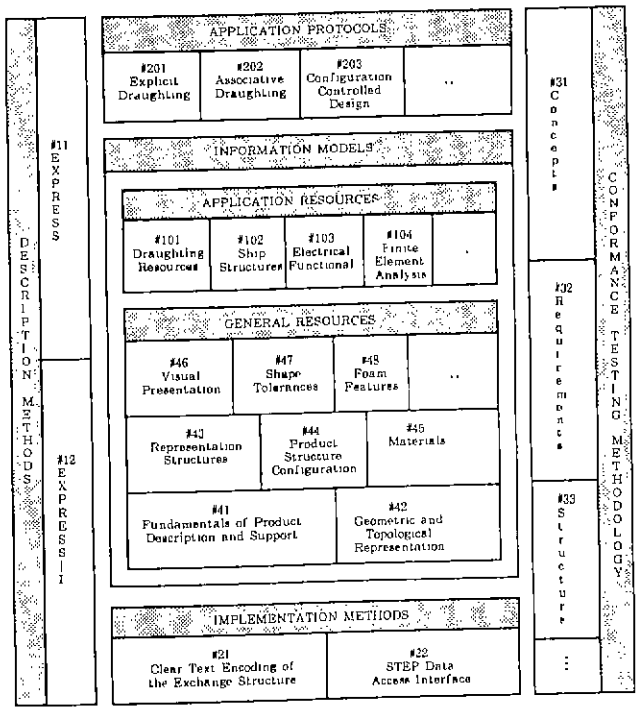

그림 1STEP의 구조

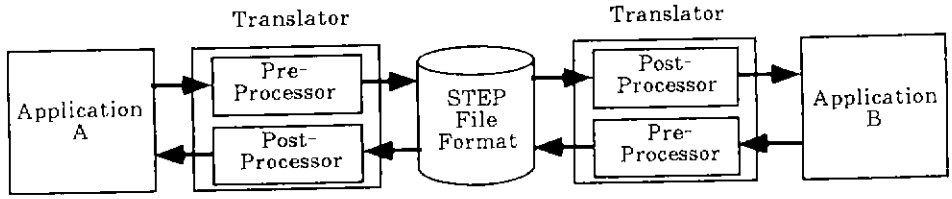

그림 2 STEP의 레벨 1 구현: Passive File Exchange

응용프로그램 A (예를 들어 하나의 CAD 시스템)가 STEP에 의하여 정의된 포맷에 맞추어 데이타를 생성한다. 이 화일이 응용프로그램 B로 옮겨져서 B에 알맞는 데이타 구조(Data Structure) 형태로 전환된다. 이때 각 응용프로그램은 자신의 화일 포맷과 STEP 포맷을 양방향으로 변환시킬 수 있는 프로세서가 필요하다.

레벨 2의 Active File Exchange를 위하여 응용프로그램은 제품 데이타를 중간 포맷으로 생성한다. 이 중간 포맷은 제품 데이타의 메모리 상주 모델인 작업 폼(Working Form)을 갖는다. 그림 3에서 응용프로그램 A와 B는 메모리 상주 모델인 작업 폼으로부터 제품 데이타 중 필요한 엔티티를 가져올 수 있다. 레벨 2에서는 이와 같이 작업 폼을 이용하여 제품 데이타를 엔티티 단위로 가져올 수 있고 STEP 화일 포맷을 이용하여 화일 전체를 가져올 수도 있다. 레벨 2를 구현한 예는 PDDI(Product Definition Data Interface)와 GMAP(Geometric Modeling Applications Interface) 등이 있다.

레벨 3인 Shared Database는 제품 데이타에 대한 논리적 기술과 물리적 구현을 데이타베이스 포맷으로 갖는다. 그림 4에 나타난 것과 같이 응용프로그램들은 공통 데이타베이스에 연결되어 필요한 정보를 가져간다. 예를 들면, 한 데이타베이스 시스템에 여러개의 응용프로그램을 동

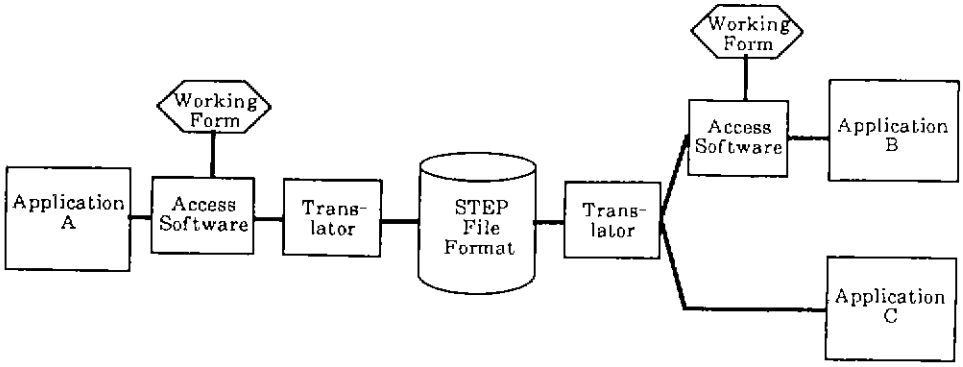

그림 3 STEP 레벨 2 구현: Active File Exchange

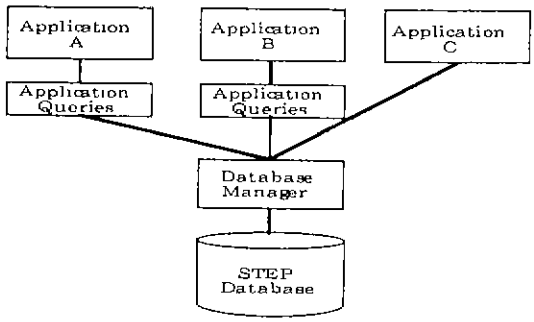

그림 4 STEP의 레벨 3 구현: Shared Database

시에 연결할 수 있다. 이때 데이타베이스에 저장된 정보는 모두 데이타 사전에 기술되어 있으므로, 응용프로그램은 필요한 데이타를 데이타베이스 관리 프로세스에 요청할 수 있고 반대로 계산 결과를 다시 저장하도록 요청 할 수도 있다. 미국의 Rockwell에 의하여 개발된 미국 공군의 IDS(Integrated Design System)는 이러한 시스템의 좋은 예이다.

2.2 EXPRESS 정보 모델링 언어

EXPRESS는 STEP에서 개발한 데이타의 논리적 구조 (Logical Scheme)를 표현하는 언어로서, 기존의 Entity-Relationship 모델의 확장이고 객체지향 패러다임을 따르고 있다. 본 절에서는 EXPRESS의 언어요소를 간단히 소개한다.

2.2.1 데이타 타입(Data Type)

EXPRESS의 데이타 타입은 다음과 같이 요약된다.

(1) Simple Data: Number, Real, Integer, Logical, Boolean, String, Binary.
(2) Aggregate: Array, List, Bag, Set.
(3) Constructed: Enumeration, Select.
(4) Entity Data Type: Entity Type은 Data Type으로도 사용 가능하다.
(5) 사용자가 정의한 Data Type: Abstract Data Type은 정의가 가능하다.

2.2.2 엔티티(Entity)

엔티티의 정의는 타입 계층(Type Hierarchy), 속성(Attribute), 지역 룰(Local Rule)을 포함한다. 타입계층은 SUBTYPE이나 SUPERTYPE으로 정의한다. 속성에는 Explicit, Derived, Inverse의 세가지가 있다. 지역 룰은 엔티티 타입에 속한 각 객체(Instance)에 적용하는 제약조건(Constraints)을 명시하며, UNIQUE절이나 WHERE절에 의하여 정의한다.

2.2.3 오퍼레이터 (Operator)

데이타 타입이나 엔티티 타입을 다루는데 필요한 다양한 오퍼레이터가 제공된다. 오퍼레이터는 Arithmetic, Relation, Binary, Logical, String, Aggregate, Entity 등으로 구분된다.

2.2.4 실행문 (Executable Statements)

EXPRESS는 Case, Escape, If-then, Repeat, While, Until, Return과 같은 실행문을 가지고 있으며 일반적인 알고리즘을 표현하는데 사용된

다.

2.2.5 Function

EXPRESS 언어는 Abs, ASin, ACos, ATan, Cos, Exists, Exp, Log, Sqrt와 같은 Built-in Function을 가지고 있다. 또한 사용자들은 실행문을 이용하여 자신의 고유한 함수를 정의할 수 있다.

2.2.6 전체 룰 (Global Rule)

Global Rule은 하나 혹은 여러개의 엔티티 타입 간의 제약조건을 명시하며, RULE을 이용하여 정의할 수 있다. RULE은 실행문과 그 결과에 의해 적합성을 결정하는 WHERE절로 이루어 진다.

2.2.7 EXPRESS의 간단한 예

다음 예는 엔티티 타입 person과 서브타입인 female을 포함한다.

```
ENTITY person
  first_name     : STRING;
  last_name      : STRING;
  birth_date     : date;
  children       : SET[0:?] OF person;
 DERIVE
  age            : INTEGER:=years(birth
                   _date);
INVERSE
  parents        : SET[0:2] OF person
                   FOR chilren;
END_ENTITY;
ENTITY female
  SUBTYPE OF (person);
  husband        : OPTIONAL male;
  maiden_name    : OPTIONAL STRING;
WHERE
  w1: (EXISTS(maiden_name) AND EXI-
      STS(husband)) XOR
      NOT EXISTS(maiden_name);
END_ENTITY;
```

위 예에서 person과 female의 2가지 엔티티 타입을 정의하였다. 엔티티 person은 first_name, last_name, birth_date, 그리고 children을 Explicit Attribute로 가지고 있다. Attribute인 age는 birth_date에서 유도된 것이다. 함수 years( )는 여기에서 정의는 생략되었지만, 주어진 birth_date로부터에서 age를 계산한다. 엔티티 female은 person의 서브타입이다. 서브타입은 슈퍼타입의 모든 속성을 상속받는다. Female의 제약조건 W1은 female이 maiden_name을 가지고 있다면 husband가 있어야 한다는 룰을 명시한 것이다.

3. 표준 데이타 인터페이스 (SDAI)

SDAI(STEP Data Access Interface)는 응용프로그램의 STEP 화일 또는 데이타베이스 시스템과의 접속 방법에 관한 표준이다. 이러한 표준을 만듦으로서 응용프로그램 개발자나 데이타베이스 개발자는 서로 독립적으로 프로그램을 개발하여 판매할 수 있다. 또한 복수의 응용프로그램들이 복수의 데이타베이스와 연결하여 데이타를 공유할 수 있다. 이때 SDAI는 복수의 응용프로그램과 복수의 데이타베이스 간의 브리지(Bridge) 역할을 함으로써 Open CIM을 실현한다.

SDAI는 EXPRESS에 의하여 정의된 정보모델을 논리적 스키마로 이용하여 화일 내용을 해석하거나 데이타베이스 스키마로 전환하여 데이타베이스 시스템을 이용한다. 또한 응용프로그램과의 접속을 위하여 C, C++, 그리고 Fortran으로 정의된 랭귀지 바인딩을 정하였고, 각 바인딩은 Early Binding과 Late Binding을 포함하고 있다.

3.1 SDAI의 기능

데이타 저장 시스템(화일 시스템 또는 데이타베이스 시스템)과 응용 프로그램간의 독립성을 유지하기 위하여 요구되는 SDAI의 기능은 다음과 같다.

(1) EXPRESS로 표현된 데이타의 관리(생성, 인출, 삭제, 수정)
(2) 단일 응용프로그램의 다수의 데이타 저장 시스템에 연결

(3) EXPRESS로 표현된 스키마 정보 관리
(4) EXPRESS로 정의된 제약조건(Constraints) 적용
(5) EXPRESS의 내정 상수(Built-in Constants) 지원

3.2 랭귀지 바인딩 (Language Bindings)

랭귀지 바인딩은 SDAI의 구현에 필요한 Function 또는 클래스 구조를 정의한다. 이러한 랭귀지 바인딩을 구현하기 위하여 EXPRESS 스키마에 종속된 부분(Early Binding)과 독립된 부분(Late Binding)이 필요하다.

(1) Early Binding: 스키마에 정의된 엔티티 타입, 속성, 계층구조, 룰 등 모든 정보를 내부 구조에 맞게 전환한다. C++로 구현할 때, 엔티티는 클래스로, 속성값(Attributes)은 프로텍트 된 변수로 전환된다.
(2) Late Binding: 특정 스키마에 종속되지 않는 일반적인 기능을 라이브러리와 같이 이용할 수 있게 한다. 예를 들어, STEP 화일을 읽고 쓰는 함수, 데이타베이스 시스템을 접속하는 함수, 내정 타입(Set, Bag, List 등)을 처리하는 함수, 제약조건을 확인하는 함수 등이 있다.

4. 객체지향 데이타베이스 시스템을 이용한 SDAI 구현

본 절에서는 객체지향 데이타베이스 시스템인 Objectivity/DB를 간단히 소개하고 이를 이용하여 구현한 SDAI를 설명한다. 관계형 데이타베이스 시스템인 ORACLE을 이용한 SDAI를 소개하고, 두 가지 방법의 차이점을 비교한다.

4.1 Objectivity/DB 객체지향 데이타베이스 시스템

Objectivity/DB는 C++을 확장하여 데이타의 Persistency, Concurrency Control, Indexing, Crash Recovery 등의 데이타베이스 관리 시스템의 기능을 구현하였다[9]. 영속적(Persistent)인 객체를 정의하기 위해서는 Objectivity/DB의 데이타 정의 언어(Data Definition Language, DDL) 프로그램에서 Objectivity/DB에서 제공하는 영속 클래스의 서브클래스로 선언하면 된다. 이러한 방법은 ODE[13]의 각 Instance의 생성시 영속성을 주는 방법과 구별된다. 정의된 영속적인 객체를 인출하기 위하여는 응용프로그램과 객체 사이의 연결을 위해 제공되는 핸들(Handle)이나 객체 레퍼런스(Object Reference)를 이용한다. 핸들이나 객체 레퍼런스는 각각의 객체마다 고유하게 배당된 OID(Object Identifier)를 사용한다.

Objectivity/DB에서는 객체간의 관계(Relationship)를 영속적인 포인터(Persistent Pointer)를 이용하여 저장한다. 이러한 관계에는 일방 관계(Unidirectinal Association)와 양 방향 관계(Bidirectional Association)가 있다. 일방 관계는 한 객체에서 다른 객체로의 추적(Traversal)을 한 방향으로만 지원하며, 양 방향 관계는 두 객체간에 서로 추적할 수 있게 한다. 또 대응하는 객체의 수에 따라 일대일(one-to-one), 일대다(one-to-many), 다대다(many-to-many) 관계로 나눌 수 있으며, 이러한 다양한 관계와 방향성이 모두 지원된다.

Objectivity/DB의 논리적 저장 구조는 FDDB(Federated Database), 데이타베이스, 컨테이너(Container), 그리고 기본 객체로 나뉘어진다. 여기서 기본 객체는 데이타베이스에 저장하기 위한 가장 기본적인 단위로서 반드시 하나의 컨테이너에만 속하게 되어있다. 기본 객체들의 모임인 컨테이너는 디스크상의 위치를 적절히 배치하여(Clustering) 효율적인 검색이 가능하게 할 수 있다. 또한 스캔(Scan), 인덱스, 잠금(Lock) 등이 컨테이너 단위로 만들어 진다. 데이타베이스는 기본 컨테이너와 사용자가 정의한 컨테이너를 포함하고 각각의 데이타베이스는 물리적으로 하나의 화일에 대응된다. FDDB는 Objectivity/DB의 논리적 저장구조에서 가장 상위 레벨로서 여러개의 데이타베이스를 포함 할 수 있다. FDDB의 각 데이타베이스는 네트웍으로 연결된

여러 시스템에 분산되어 존재할 수 있다.

4.2 SDAI 구현

SDAI의 구현을 위하여 EXPRESS로 정의된 스키마를 Objectivity/DB의 DDL로 전환한다. 이때 EXPRESS로 표현된 전체 스키마가 Objectivity/DB에서 하나의 FDDB가 되고, 전체 스키마에 포함되는 각각의 스키마는 하나의 데이타베이스로 전환된다. 엔티티 타입은 하나의 영속 클래스에 대응되며 각 엔티티 타입마다 하나의 컨테이너를 생성한다. 이렇게 함으로써 인덱싱과 잠금의 범위를 한 엔티티 타입의 범위로 하였다. EXPRESS에서는 속성의 상속(Inheritance)이 가능하여 슈퍼타입과 그의 서브타입들이 있다. EXPRESS의 한 엔티티 타입이 Objectivity/DB의 클래스로 전환되었을 때 그 클래스에 상위 클래스(Superclass)가 없을 경우 Objectivity/DB의 ooObj 클래스의 서브클래스로 정의하여 영속적인 클래스로 만든다. 다른 영속적인 클래스의 서브클래스는 자연적으로 영속적인 클래스가 된다.

속성은 엔티티 클래스의 데이타 멤버로 전환되어 엔티티의 핸들이나 멤버 함수를 통하여 접근한다. Enumeration의 경우는 Objectivity/DB에서 enum 타입으로 변환되며, 소속 item들의 이름은 중복되지 말아야 한다. STEP working form에서 읽어들인 string을 enum 값으로 변환시켜 주는 기능은 비영속적인 클래스에 의하여 수행하도록 하였다. Simple entity attribute type은 하나의 엔티티가 다른 엔티티의 속성으로 사용되는 경우로 포인터를 통하여 구현하였다. Array, bag, set, list와 같은 aggregate entity attribute type은 일대다 관계에 해당되며, 각각의 데이타 타입을 위하여 비영속 클래스인 SdaiArray, SdaiBag, SdaiSet, 그리고 SdaiList를 정의하여 필요한 기능을 처리하였다.

이상에서 설명한 Objectivity/DB를 이용한 SDAI의 구조는 그림 5와 같다. 여기서 SDAI의 Early Binding이 Objectivity/DB의 논리적 구조(Logical Scheme)를 정의한다. 이것은 Objectivity/DB의 DDL이 C++와 같은 구문을 사용하기

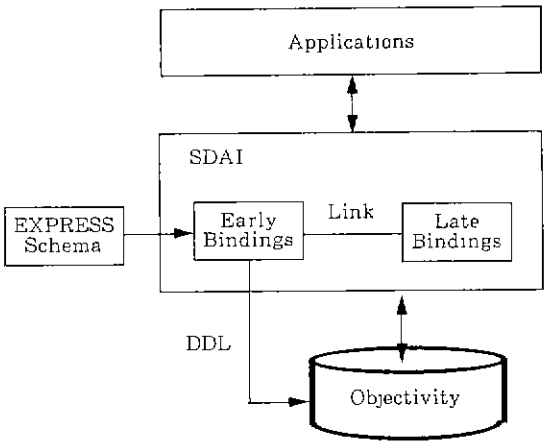

그림 5 Objectivity/DB를 이용한 SDAI 구조

때문에 가능하다. Late Binding에는 SdaiArray, SdaiBag 등과 같은 클래스들이 정의되어 있어 앞에서 설명한 SDAI의 기능을 수행한다

예를 들어 다음과 같은 간단한 EXPRESS 스키마를 고려한다.

```
SCHEMA EX_SCHEMA;
 ENTITY PERSON;
  NAME : STRING;
  AGE  : INTEGER;
 END_ENTITY;
END_SCHEMA;
```

위 스키마로부터 다음과 같은 C++ 헤더 화일이 생성된다.

```
class PERSON : public ooObj { /* inherits persistancy */
  protected:
   SdaiString_NAME;  /* attributes */
   SdaiInteger_AGE:
  public:
   PERSON( ): /* constructors */
   PERSON(SdaiString a, SdaiInteger b);
   PERSON(PERSON& e)
   ~PERSON( ); /* destructor */
   SdaiString Name( ) { return "PERSON"; } /* returns entity name */
   SdaiString NAME( ); /* return the value of attribute NAME */
   void NAME (SdaiString x);   /* initialize
```

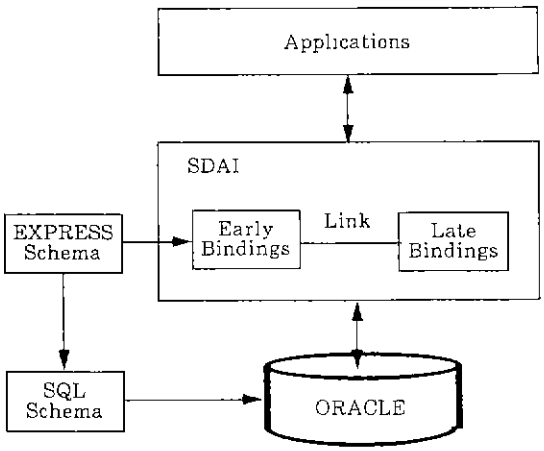

그림 6 ORACLE을 이용한 SDAI 구조

```
    the attribute NAME */
    SdaiInteger AGE(  ); /* return the value
    of attribute AGE */
    void AGE (SdaiInteger x);/* initialize the
    attribute AGE */
};
```

이 C++ 헤더 화일이 SDAI의 Early Binding과 Objectivity/DB의 DDL 프로그램으로 사용된다. 그러나 관계형 데이타베이스 시스템인 ORACLE을 사용할 때는 다르게 되는데 그 이유는 ORACLE의 스키마 정의를 위하여 SQL 프로그램이 필요하기 때문이다. ORACLE을 이용한 SDAI의 구조는 그림 6과 같다.

앞에서 예로 들은 엔티티 PERSON으로부터 출력되는 SQL 프로그램의 일부는 다음과 같다.

```
/* makes an entity name unique */
INSERT INTO SDAI$NAMES VALUES (
  'PERSON', 'PERSON$E1', 'ENTITY');
/* creates a table for the entity PERSON
*/
CREAE TABLE PERSON$E1 (
  ENTITY_ID  INTEGER      PRIMARY
             KEY;
  NAME       CHAR(240) NOT NULL;
  AGE        INTEGER NOT NULL
);
/* information about the attributes */
INSERT INTO SDAI$ATTRIBUTES VA-
LUES (
  'PERSON$E1', 'NAME'. 'STRING', 'NULL',
0, 0, 0, 1);
INSERT INTO SDAI$ATTRIBUTES VA-
LUES (
  'PERSON$E1', 'AGE', 'STRING', 'NULL',
0, 0, 0, 2);
```

위 SQL 프로그램에서 엔티티의 이름을 변경한 이유는 ORACLE에서 테이블의 이름이 30자로 제한되어 있기 때문에 이를 극복하기 위해서다. 이 예에는 들어 있지 않지만, 엔티티의 계층 관계 때문에 상속되는 속성들도 모두 엔티티의 정의에 포함되어야 한다. 이는 객체지향 모델에 의한 EXPRESS 스키마가 관계형인 SQL로 바뀌기 때문이다. 이러한 스키마 변환으로 인하여 사용자는 큰 차이를 느끼지 않지만 프로그램의 크기와 복잡성은 크게 증가하고 그에 따라 프로그램의 수행 속도는 느려진다.

5. 결 론

ISO 프로젝트인 STEP은 CAD, CAM, CAE 등의 자동화 시스템들이 사용하는 데이타 구조를 정의한다. 이것은 현재 각 자동화 시스템간의 서로 다른 화일 양식이나 데이타베이스 구조의 차이로 인한 통합의 어려움을 극복하기 위한 것이다. STEP에서 정의된 표준 데이타 들은 화일이나 데이타베이스 시스템을 통하여 교환 또는 공유 된다. 본 논문에서는 객체지향 데이타베이스 시스템인 Objectivity/DB를 이용하여 STEP 데이타를 공유하는 시스템을 설명하였다. STEP에서 데이타 모델링 언어로 사용중인 EXPRESS, 표준 데이타 인터페이스로 정의한 SDAI의 C++ 바인딩, 그리고 Objectivity/DB가 모두 객체지향 모델을 사용하고 있기 때문에 SDAI의 구현에서 쉽게 일관된 스키마를 유지할 수 있다. 한편, 관계형 데이타베이스 시스템인 ORACLE을 이용할 경우, 엔티티 간의 상속 관계를 고려하여 테이블의 속성을 정의해야 한다. 이러한 데이타 모델의 변화는 프로그램의 복잡성을 증가시켜서

프로그램의 개발 및 유지에 필요한 비용을 증가시킨다. 이점은 향후 강력한 모델링 능력과 우월한 개발 효율로 인한 객체지향 모델 또는 프로그램의 보급이 증가함에 따라, 객체지향 데이타베이스의 이용도 확대 될 것이라는 예측을 뒷받침한다.

이제까지 객체지향 데이타베이스 시스템은 효율적인 응용프로그램 개발은 가능하지만 일반 사용자를 위한 질의어가 표준화되지 않아 사용하기 어려운 것으로 인식되어 왔다. 이러한 문제점은 최근 객체지향 데이타베이스 시스템 업체간에 질의어의 표준에 합의함으로써 가까운 장래에 해결될 것으로 보인다. 또한 관계형 데이타베이스 시스템들도 객체지향 기능이 보완되고 있어, 두 서로 다른 데이타 모델에서 출발한 데이타베이스 시스템들은 외부적으로 유사하게 되고 있다. 데이타베이스 시스템을 선정할 때 가장 중요한 요인은 응용 프로그램에 포함될 데이타의 특성이다. CIM, 멀티미디어, 지리정보 시스템 등과 같이 대상 데이타의 형태가 다양하고 응용프로그램의 개발을 위해 복잡한 데이타 조작이 필요한 분야에 객체지향 데이타베이스 시스템의 장점이 잘 이용될 수 있다.

참고문헌

[1] ESPRIT Consortium AMICE, Open System Architecture for CIM, Springer-Verlag, 1989.

[2] Fumihiko Kimura, "Factory Automation and CAD-Product Modelling for Advanced 5anufacturing Automation", 한국정밀공학회지 제 8권 제 2호 pp 7~26, 1991.

[3] Kai Mertins and Wolfram Sussenguth, "Integrated Information Modelling for CIM", Computer-Integrated Manufacturing Systems, 4(3), pp. 123~131, 1991.

[4] Allan Hodgson and Gerry Waterlow, "Integrating the Engineering and Production Function in a Manufacturing Company," Computing & Control Engineering Journal, March 1992.

[5] Susan Urban, Jami Shah, and Mary Rogers, "Engineering Data Management: Achieving Integration through Database Technology," Computing & Control Engineering Journal, June 1993.

[6] Dimitris Choratas, "The Engineering Database", Butterworths, 1988.

[7] Howard Mason, "STEP Part1: Overview and Fundamental Principles," ISO, 1992.

[8] ISO, "Standard Data Access Interface," TC184/SC4/WG7 Document Number 300, May 1993.

[9] Objectivity Inc., "Objectivity/DB C++ Interface Guide, Version 2.1." 1993.

[10] ORACLE Inc., "SQL Language Reference Manual, Version 6.0," 1990.

[11] Society of Manufacturing Engineers, "PDES: The Enterprise Data Standard," 1989.

[12] ISO, "EXPRESS Language Reference Manual," TC184/SC4/WG5 Document Number 151, August 1992.

[13] A. Biliris, N. Gehani, D. Liuwen, E. Panagos, and T. Roycraft., "ODE 2.0 User's Manual," AT & T Bell Laboratories, Murray Hill, New Jersey, U.S.A., 1993.

유 상 봉

1982 서울대학교 공과대학 제어계측공학과 졸업(학사)
1986 미국 University of Arizona 공과대학 전기 및 컴퓨터공학과 졸업(석사)
1990 미국 Purdue University 공과대학 전기 및 컴퓨터공학과 졸업(박사)
1989 미국 AT & T Bell Laboratory, Holmdel, NJ, 연구원
1990 ~.1992 삼성전자 컴퓨터부문 선임연구원
1992 ~현재 인하대학교 자동화공학과 조교수
관심 분야 : Engineering Database Systems, Real Time Database Systems, System Integration.

차 상 균

1980 서울대학교 공과대학 전기공학과 학사
1982 서울대학교 공과대학 전기공학과 석사
1991 미국 Stanford대학교 전기공학과(컴퓨터시스템) 박사 Texas Instruments, IBM Palo Alto, Hewlett-Packard Lab 근무
1992 ~현재 서울대 제어계측공학과 조교수