The Journal of Korean
Institute of CALS/EC
Vol. 3, No. 1, June. 1998

이종 데이터 모델에서의 STEP 표준 데이터 인터페이스(SDAI) 구현*

권용국**, 유상봉***

Implemetation of STEP Standard Data Interface (SDAI) on Multiple Data Models

Y. K. Kwon, S. B. Yoo

Abstract

SDAI (Standard Data Access Interface) is an interface specification of accessing various storage systems such as file systems and database management systems for STEP data. Using SDAI, both application program developers and CAD/CAM system developers can be relieved from dealing with STEP physical file or system dependent DBMS operations. In this paper, we present implementations of SDAI on different data models, i.e., relational, extended relational, and object-oriented. In order to implement SDAI, we need to translate the EXPRESS information model into target data models. The schema translation process for three different data models are compared and other implementation issues are discussed.

Key Words: *CALS, STEP, SDAI, EXPRESS, Data Model, CAD/CAM, API (Application Program Interface).*

---

* 본 연구는 97 년도 한국과학재단 특정기초연구 (과제번호: 97-02-00-09-01-3)의 지원으로 수행된 것임.

** 두산기계 FA 팀

*** 인하대학교 자동화공학과 교수

1. 서론

CAD/CAM 시스템을 통해 제품을 설계 개발하는 과정에서 효율적인 정보처리 및 생산성 향상을 위하여 각종 데이터베이스 시스템들을 도입하여 사용하고 있다. 이러한 시스템들은 각기 독자적인 데이터 모델과 데이터 접근 양식을 제공하고 있어 사용자의 목적과 환경에 따라 시스템을 변형하거나 필요에 따른 응용 프로그램을 작성하여 사용하고 있다. 그러나 사용자가 사용 목적에 맞는 형태로 응용 프로그램을 작성하기 위해서는 그 시스템에 대한 전반적인 이해가 필요하게 되어, 거기에 따른 추가적인 비용 및 인력이 문제점으로 대두된다. 이는 서로 다른 이기종 CAD/CAM 시스템들간의 정보 교환이 필요로 하는 경우에도 마찬가지이다. 새로운 시스템을 개발하고자 하는 경우에도 시스템 고유의 데이터 형식 및 접근 방법 외에도 사용자들이 많이 쓰고 있는 형식들을 고려해야 하는 부담이 따르게 된다.

이러한 문제점은 시스템의 종류에 독립적인 인터페이스 함수를 정의하고 이러한 인터페이스를 각 시스템에서 지원함으로써 해결될 수 있다. STEP (STandard for the Exchange of Product Data) [ISO, 1994a]에서는 저장 시스템의 종류에 독립적인 STEP 데이터 인터페이스의 표준으로 SDAI (Standard Data Access Interface)를 개발하였다 [ISO, 1996]. SDAI 를 이용함으로써 CAD/CAM 시스템을 포함하는 STEP 응용프로그램 개발자들은 STEP 물리 파일 [ISO, 1994c]이나 각종 데이터베이스 시스템을 직접 접근하지 않고도 STEP 데이터를 저장 또는 인출할 수 있다.

데이터베이스 시스템은 사용하는 데이터 모델에 의하여 계층형, 네트웍, 관계형, 객체지향형, 확장관계형 등으로 분류된다 [Elmasri, 1994]. 현재 관계형 데이터 모델이 가장 널리 쓰이고 있으나, 모델링 기능의 장점으로 인해 객체지향형과 확장관계형 모델의 도입이 점차 늘고 있는 추세이다. 본 논문에서는 기존에 널리 상용 중인 관계형 데이터베이스 시스템인 ORACLE [Oracle, 1990]과 향후 활용 가능성이 높은 객체지향형과 확장관계형 시스템으로 각각 Versant [Versant, 1994]와 UniSQL [UniSQL, 1992]을 이용한 SDAI 구현 방법을 설명하고 이 세 시스템에서의 구현 과정을 비교하였다.

본 본문의 구성은 다음과 같다. 2 장에서는 본 연구와의 관련 연구를 설명하였고, 3 장에서는 본 연구에서 구현된 전체 시스템의 구조에 대하여 전반적으로 서술하였다. 4 장에서는 Oracle, UniSQL, 그리고 Versant 에서의 스키마 변환 방법을 기술한다. 5 장에서는 이러한 스키마 변환을 이용한 각 데이터베이스 시스템에서의 SDAI 의 구현 방법을 설명하였다. 6 장에서는 본 본 연구

에서 SDAI 의 구현에 사용한 세가지 모델을 비교하였다. 7 장에서는 본 연구의 결론 및 향후 연구 과제에 대해 언급한다.

2. 관련 연구

STEP 데이타를 관계형 데이타베이스 시스템에 저장하기 위하여 EXPRESS 스키마를 관계형 데이타베이스 시스템의 표준 언어인 SQL 로의 변환하는 방법이 연구 되어졌다 [Metz, 1989] [Mosbacher, 1990]. 이러한 변환에서 주목할 점은 EXPRESS 로 표현된 엔티티 간의 상속된 속성 (inherited attributes)과 집합으로 표현된 속성(중첩 attributes)의 표현이다. 상속 되는 속성은 상속 받는 객체 (entity)에 해당하는 테이블에 정의 되며, 집합으로 표현된 속성은 관계형 데이터베이스 시스템에서 하나의 속성값으로 집합이 허용되지 않기 때문에 별도의 테이블이 정의된다. 관계형 데이터베이스 시스템을 이용한 SDAI 의 대표적인 구현은 미국의 STEP Tools 사의 STEP Database Adapter 가 있다 [STEP Tools, 1993b].

객체지향형 데이터베이스 시스템은 공학 및 과학 분야의 다양한 객체와 그들 간의 복잡한 관계를 효과적으로 모델링하고 클래스의 함수 (method)를 이용한 인캡슐레이션 (encapsulation)을 통하여 프로그램 개발과 유지 보수를 용이하게 할 수 있어 보급이 확대되고 있다. 특히 SDAI 의 구현에서 유리한 점은 EXPRESS 가 객체 지향 데이터 모델을 사용하고 있어 스키마 변환이 용이하다는 점이다. 구현된 시스템으로 DEC (Digital Equipment Corporation)의 C++ DAI 는 프로그래밍 언어로 C++을, 데이터베이스로는 객체지향형 데이터베이스인 Objectivity/DB 를 사용하였으며, 데이터 교환의 테스트를 위하여 솔리드 모델링 시스템인 I-DEAS 를 이용하였다[Whitehead, 1991]. STEP Tools 사의 STEP Database Adapter 는 객체 지향 데이터베이스 시스템으로 ObjectStore, OpenODB, 그리고 Versant 와의 인터페이스를 제공하고 있다 [Wilson, 1995]. 그리고 객체 지향 모델의 표준인 CORBA 를 통한 SDAI 의 구현이 미국의 General Dynamics 에서 수행되었다 [Rando, 1994].

국내에서는 인하대학교에서 Oracle 을 이용한 SDAI 의 구현을 발표하였고 [김기환, 1996], 포항공대에서 객체지향형 데이터베이스 시스템에 트리거 기능을 추가한 능동적 객체지향형 데이터베이스 시스템에서 STEP 데이터의 관리를 위하여 EXPRESS 스키마를 변환하였다 [도남철, 1993] [도남철, 1996]. 본 논문에서는 관계형, 객체지향형, 그리고 확장관계형의 세가지 데이터 모델을 이용하여 EXPRESS 정보 모델을 변환하였고 이러한 각 시스템에서 SDAI 의 구현에 관련된 스키마 변환의 복잡도와 프로그램 기술의 난이도를 비교하였다.

3. 전체 시스템의 구조

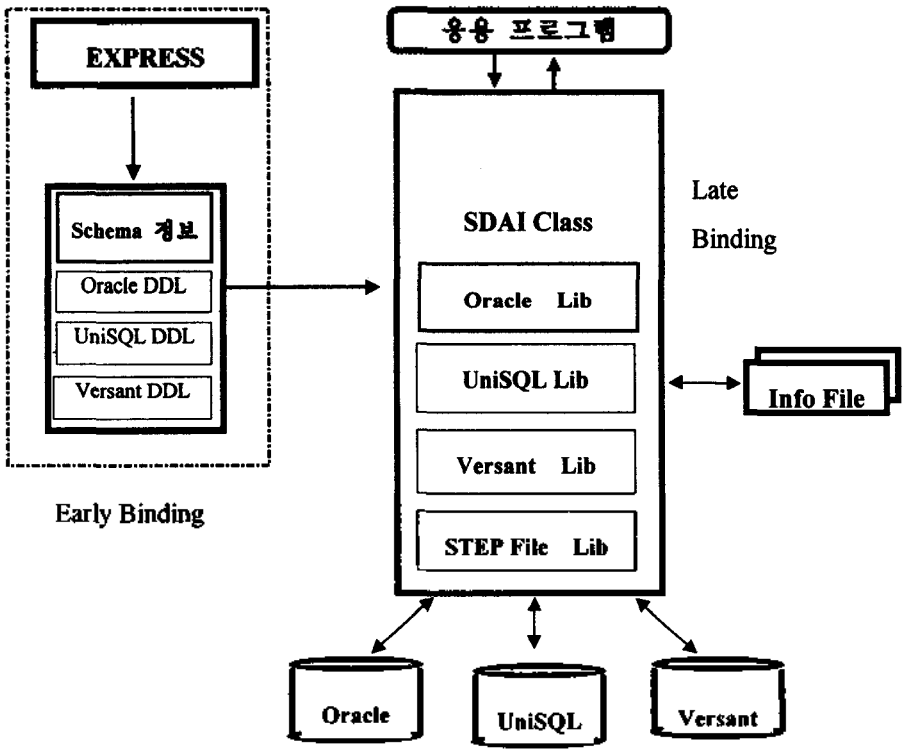
이종 데이터 모델을 이용한 SDAI 시스템의 구조는 그림 1과 같으며, 크게 두 부분으로 나뉘어진다. 첫째는 EXPRESS로 표현된 스키마를 EXPRESS 컴파일러인 fedex [Clark, 1992]를 이용한 프로그램을 통해 컴파일하면서 스키마가 담고 있는 정보를 작업 형식(working form)으로 추출해내는 초기 바인딩(early binding) 부분이다. 초기 바인딩에서는 스키마 정보를 이용해 인터페이스하고자 하는 데이터베이스 형태로 데이터 정의어(data definition language)를 작성한다. 둘째는 초기 바인딩을 통해 얻어진 정보, 즉 스키마의 내용을 담고 있는 스키마 헤더 파일, 초기화 파일 및 소스 파일을 각종 라이브러리들과 결합해 원하는 작업에 필요한 실행 파일을 생성해내는 후기 바인딩(late binding) 부분이다. 이때 필요한 라이브러리들은 STEP 파일과 관련된 라이브러리와 이종 데이터베이스와의 인터페이스들을 담당하는 라이브러리, 작업 형식 정보와 관련된 라이브러리 등이 요구된다.

초기 바인딩을 통해 스키마에 의존적인 정보를 추출한다. 그림 1에서 와 같이 EXPRESS 파일을 컴파일하는 과정에서 스키마 정보를 담은 파일들과 데이터베이스의 데이터 정의어 화일들을 생성한다. 스키마 정보에 관련된 파일은 엔티티들을 클래스 형태로 정의하는 "Schemas.h" 파일과 엔티티 클래스들을 초기화시키는 "Schemas.init.cc" 파일, 그리고 실제적인 초기화 함수와 사용자 정의 함수를 호출하는 메인 함수를 포함하는 "Schemas.cc" 파일 등 이다.

후기 바인딩은 사용자 정의 함수를 초기 바인딩 과정에서 생성된 각종 스키마 정보 파일과 SDAI 라이브러리, 각 데이터베이스 라이브러리 및 STEP 파일 라이브러리를 결합시켜 실행 파일을 생성하는 과정이다. 그림 1에서와 같이 응용 프로그래머는 SDAI 기능을 이용하여 각가 다른 세 개의 데이터베이스 시스템의 데이터에 접근할 수 있다. 각기 다른 데이터베이스 시스템이지만 사용자가 호출하는 함수는 동일하다.

실제적인 작업의 순서는 데이터베이스 시스템이나 파일에 저장되어 있는 데이터를 메모리로 가져와 요구하는 작업을 하고 다시 데이터베이스 시스템이나 파일에 저장하는 과정의 순으로 이루어진다. 이때 데이터의 이동 단위는 모델로서 모델은 일련의 엔티티 모임이다. 정보 데이터가 메모리상에 존재할 경우 데이터는 딕셔너리 형태로 존재하게 되며 필요에 따라 각 데이터베이스의 라이브러리 및 STEP 파일

라이브러리를 이용하여 데이터베이스 시스템이나 파일과의 인터페이스를 수행하게 된다. SDAI 클래스는 각종 SDAI 오퍼레이션들을 멤버 함수로 갖고 있는 클래스로써 STEP 표준 데이터 인터페이스를 담당하며 모델에 관련된 저장소 정보를 참조할 수 있게 해 준다. 모델과 모델을 저장하는 데이터베이스 시스템에 관한 정보는 Info 테이블에 기록되며 모델의 조작 시 이 정보를 참조하여 사용자로 하여금 데이터베이스와 독립하여 작업을 수행할 수 있게 한다. Info 테이블의 내부 형식은 모델 이름과 데이터베이스 이름을 갖는 속성으로 구성되어 있다.



<그림 1> 표준 데이터 인터페이스 구조도

4. 스키마 변환

4.1 Oracle 에서의 스키마 변환

EXPRESS 정보 모델에 정의되어 있는 엔티티들은 Oracle 상에 엔티티 테이블로 변환된다. 엔티티 테이블은 기본적인 테이블로서 ID, EXPLICIT_ATTRIBUTE, 그리고 IMPLICIT_ATTRIBUTE 를 속성으로 가지고 있다.

테이블 이름은 테이블이 나타내는 엔티티의 이름을 짧게 만들어 'E$' 뒤에 덧붙이는 형태로 하였다. 이름의 글자수가 30 이 넘지 말아야 하는 Oracle 의 제약때문에 중첩 (Aggregate) 테이블 이름을 고려하여 14 자를 최대 크기로 하였다. 변환하는 방법은 다음과 같다.

1) 엔티티 이름이 13 자 이하일 경우에는 엔티티 이름 앞에 엔티티임을 상징하는 'E$' 를 덧 붙인다.

2) 엔티티 이름이 13 자가 넘을 경우에는 앞에서 부터 9 자만을 그대로 사용하고 엔티티 이름의 총 길이를 2 자리 숫자로 표시한후 '_' 를 덧 붙인다.

3) 엔티티 이름이 13 자가 넘을 경우 엔티티 이름의 제일 마지막 문자를 '_' 뒤에 덧 붙인다.

모든 엔티티 테이블의 첫번째 컬럼은 ID 이다. ID 는 각각의 엔티티 인스탄스들의 고유한 식별자이며, 테이블의 주요키(primary key)로 이용된다. 또한 다음과 같은 세가지 형태로 다른 테이블에 위해 참조될 수 있다. 첫째, 다른 엔티티에 의해 참조되는 엔티티의 경우 참조하는 엔티티의 속성값으로 이용된다. 둘째, 그 크기가 정의되어 있지 않은 중첩 속성(타입이 Array, Bag, List, Set 인 속성)를 가진 엔티티의 경우 그러한 속성들을 위한 속성 테이블이 생성된다. 엔티티 테이블과 속성 테이블을 연결하기 위해서 두 테이블은 같은 엔티티 식별자를 갖는다. 셋째, 상위 엔티티로부터 계승된 엔티티의 경우 상위 엔티티 테이블과의 연결을 위하여 같은 엔티티 식별자로 이용한다. 식별자는 고유한 정수형으로 나타낸다.

중첩 속성이 아닌 경우에는 바로 그 테이블에 하나의 컬럼이 된다. 예외적으로 Select 타입의 경우, 중첩 속성 중에서 그 크기가 제한된 경우는 하나 이상의 컬럼이 된다. 계승 받은 속성들을 위한 컬럼은 새로 생성하지 않으며, 상위 엔티티 참조를 통해 접근하도록 했다.

엔티티의 속성들은 그 타입에 따라 그 엔티티 테이블이나 그 엔티티 테이블이 참조하는 속성 테이블의 컬럼으로 표현된다. 속성이 중첩 이면서 그 크기가 정의되어 있지 않은 경우에는 속성의 테이블의 컬럼으로 변환되며, 나머지 경우에는 엔티티 테이블의 컬럼으로 변환되는데, 예외적으로 그 크기가 정의되어 있는 중첩과 Select

타입의 경우에는 하나 이상의 컬럼으로 변환된다.

속성의 타입이 엔티티 이거나 그 크기가 정의 되지 않은 중첩 타입의 경우 그 속성 값은 다른 테이블을 가르킨다. 엔티티 타입의 경우는 다른 엔티티의 식별자를 그 값으로 하며, 그 크기가 정의되어 있지 않은 중첩 타입의 경우 자신의 엔티티 식별자를 그 값으로 하여 그 엔티티에 속한 중첩 테이블에 있는 중첩 데이터를 가르키도록 하였다.

그 크기가 정의 되지 않은 중첩 속성들은 엔티티의 속성들로 변환되며, 각 속성은 SQL 에서 허용되는 최대 길이인 30 자 이하의 이름을 갖아야 하는 조건을 만족해야만 한다. EXPRESS 의 기본 타입들은 다음과 같이 Oracle 타입으로 변환되며 String 의 경우 240 자를 기본으로 하였다. 이와같은 EXPRESS 객체의 변환은 표 1 과 같다.

엔티티 속성은 다른 엔티티 테이블의 엔티티 식별자를 그 값으로 하며 INTEGER 값으로 표현한다. Enumeration 속성은 관계형 데이터베이스에서 지원하지 않는 데이터 타입이며 VARCHAR(240)으로 하며 그 값은 문자열의 형태로 저장된다. 중첩 속성의 경우 그 크기의 제한 여부에 따라 두 가지 경우로 나누어 생각해 볼 수 있다. 그 크기가 정의 되어 있지 않은 경우는 속성 테이블을 새로 구성하여 다루며 크기가 정해져 있는 경우는 그 크기 만큼의 컬럼을 엔티티 테이블에 구성한다. 같은 엔티티 테이블에서 접근함으로써 데이터베이스의 데이터 접근 시간이 단축되며 데이터로의 접근 방법도 용이하다.

<표 1> EXPRESS 의 Oracle 로의 변환

EXPRESS	ORACLE
INTEGER	INTEGER
REAL	FLOAT/DOUBLE
STRING	VARCHAR(240)
BOOLEAN	INTEGER
LOGICAL	INTEGER
BINARY	NUMBER
STRING(N)	VARCHAR(N)

컬럼 이름은 중첩 속성의 이름 뒤에 순서를 정의 하는 숫자를 덧붙여 사용한다. 예를 들어 Array [1:3] of Integer 의 경우 Integer 값을 갖는 세 개의 컬럼이 생성된다.

스키마의 변환 예로서 다음과 같은 간단한 EXPRESS 모델을 고려한다.

```
ENTITY person
    first_name      : name;
    last_name       : name;
    nick_name       : OPTIONAL STRING;
    birth_date      : date;
    age     : INTEGER;
    children        : SET [0:?] OF person;
    hair    : hair_type;
    korean      : LOGICAL;
END_ENTITY;
```

위 스키마에 대해 생성되는 Oracle 스키마는 다음과 같다.

```
CREATE TABLE E$person (ID INTEGER
  NOT NULL UNIQUE);
ALTER TABLE E$person ADD first_name
  VARCHAR(15) NOT NULL;
ALTER TABLE E$person ADD last_name
  VARCHAR(15) NOT NULL;
ALTER TABLE E$person ADD nick_name
  VARCHAR(240);
ALTER TABLE E$person ADD birth_date1
  INTEGER NOT NULL;
ALTER TABLE E$person ADD birth_date2
  INTEGER NOT NULL;
ALTER TABLE E$person ADD birth_date3
  INTEGER NOT NULL;
ALTER TABLE E$person ADD age INTEGER
  NOT NULL;
ALTER TABLE E$person ADD children
  INTEGER NOT NULL;
ALTER TABLE E$person ADD hair
  VARCHAR(240) NOT NULL;
ALTER TABLE E$person ADD korean
  VARCHAR(7) NOT NULL;
```

4.2 UniSQL 에서의 스키마 변환

UniSQL 의 경우 Oracle 과는 달리 Array, Bag, Set, List 의 중첩 타입들과 Select 타입과 유사한 타입을 지원하므로 스키마 변환이 용이하다. UniSQL 에서 지원하는 SET, MULTISET, SEQUENCE 등으로써 중첩 타입과 Select 타입 및 String, Binary, Boolean, Logical 과 같은 타입들을 대신할 수 있다. 또한 계승 관계에 있는 엔티티들의 관계 역시 UniSQL 에서 제공하는 계승 관계를 이용해 변환시킬 수 있다.

테이블의 이름은 테이블이 나타내는 엔티티의 이름을 그대로 사용하였다. 엔티티 ID 컬럼은 모든 엔티티 테이블의 첫번째 컬럼이며 컬럼의 이름은 ID 이다. ID 의 값들은 Integer 형태이며 각각의 엔티티 객체들의 고유한 식별자로써 사용된다. 속성의 두 가지 형태, 즉 엔티티 자신의 속성과 상위 엔티티로부터 계승받은 속성 중 자신의 속성은 직접적으로 변환하며 계승받은 속성은 엔티티 계승관계를 이용하여 변환한다. 각각의 속성들은 하나의 컬럼으로 표현되며, 그 컬럼의 이름으로 각각의 속성 이름이 사용된다. 이때 그 이름의

<표 2> EXPRESS 기본 타입의 UniSQL 데이타 타입으로의 변환

EXPRESS 타입	UniSQL 타입
INTEGER	INTEGER
REAL	FLOAT/DOUBLE
STRING	STRING
BOOLEAN	CHAR(3)
LOGICAL	CHAR(5)
BINARY	NUMBER
STRING(N)	CHAR(N)

길이가 30 자 이하여야 한다. EXPRESS 의 기본 타입들은 표 2 와 같이 UniSQL 타입으로 변환시킬 수 있다.

Oracle 에서 String 타입을 VARCHAR (240)으로 변환한 것과는 달리 UniSQL 에서는 UniSQL 에서 지원하는 String 타입을 그대로 이용하여 변환하였다. 엔티티 속성들의 타입은 참조하고자 하는 엔티티가 되며, 이것은 UniSQL 에서 지원하는 것으로써 이 때 실제적인 값은 UniSQL 내부적으로 생성된 값을 사용하게 된다.

Enueration 속성은 UniSQL 에서 지원하지 않는 데이터 타입이다. UniSQL 의 String 타입을 이용해 변환하였다. 중첩 속성으로는 Array, Bag, List, Set 등이 있는데 UniSQL 에서 지원하는 타입인 SET, MULTISET, SEQUENCE 등으로 유사하게 변환할 수 있다. 표 3 은 그 대응 관계를 나타낸 것이다.

<표 3> EXPRESS 중첩 타입의 UniSQL 로의 변환

EXPRESS 중첩 타입	UniSQL 타입
SET	SET
BAG	MULTISET
LIST/ARRAY	SEQUENCE

앞 절에서 사용한 EXPRESS 스키마의 예에 대해 생성된 UniSQL 스키마는 다음과 같다.

```
CREATE CLASS person (UID INTEGER NOT NULL UNIQUE);
ALTER CLASS person ADD ATTRIBUTE first_name CHAR(15) NOT NULL;
ALTER CLASS person ADD ATTRIBUTE last_name CHAR(15) NOT NULL;
ALTER CLASS person ADD ATTRIBUTE nick_name STRING;
ALTER CLASS person ADD ATTRIBUTE birth_date SEQUENCE(INTEGER);
ALTER CLASS person ADD ATTRIBUTE age INTEGER NOT NULL;
ALTER CLASS person ADD ATTRIBUTE children SET(person) NOT NULL;
ALTER CLASS person ADD ATTRIBUTE hair STRING NOT NULL;
ALTER CLASS person ADD ATTRIBUTE korean CHAR(5) NOT NULL;
```

4.3 Versant 에서의 스키마 변환

EXPRESS 스키마 상의 모든 엔티티들은 하나의 엔티티 클래스로 변환된다. EXPRESS 와 Versant 는 모두 다 객체지향적이라는 점에서 많은 유사점을 갖고 있기 때문에 쉽게 변환 시킬 수 있다. 모든 엔티티 클래스들은 Versant 내에 내부적으로 정의된 PObject 클래스의 하위 클래스로 정의되며 상위 클래스로부터 계승 받은 속성들과 오퍼레이션들을 이용하여 데이터베이스와 정보를 교환한다. 다음은 엔티티 클래스에 대한 일반적인 형태이다.

```
class entity_name:public PObject
{
    private:
        o_4b    ID;
        [속성데이터 타입]  [속성이름];
    public:
        [엔티티 관련 오퍼레이션];
}
```

각각의 엔티티가 갖고 있는 속성에 대한 접근은 엔티티가 갖고 있는 내부 오퍼레이션들을 통해서만 가능하다. 엔티티 클래스 이름은 그 엔티티 클래스가 표현하는 엔티티 이름을 사용한다.

모든 엔티티 클래스들은 ID 라는 속성을 갖는다. 속성 ID 는 각 엔티티 인스탄스들의 고유한 식별자로 이용되며 그 값은 정수이다. 엔티티가 갖고 있는 속성들은 모두 엔티티 클래스의 속성으로 된다. Versant 에서 지원하지 않는 Select 와 중첩 타입들은 각 타입과 유사한 클래스로 대치하여 일 대 일로 변환하였다. EXPRESS 의 기본적인 데이터 타입들은 표 4 와 같이 Versant 에서 지원하는 데이터 타입으로 대응 관계를 갖는다.

<표 4> EXPRESS 타입의 Versant 타입으로의 변환

EXPRESS 데이터 타입	Versant 데이터 타입
INTEGER	o_4b
REAL	o_float/o_double
BOOLEAN	o_bool
LOGICAL	o_bool
STRING	PString
BINARY	Vstr<o_1b>

엔티티 속성은 C++ 언어와 마찬가지로 엔티티와 이름이 엔티티 속성의 데이터 타입이 되며 엔티티 인스탄스를 그 속성의 값으로 갖는다. Enumeration 속성은 PString의 형태로 Versant 데이터 타입으로 변한다. 중첩 속성의 경우는 그 중첩 속성을 구성하는 요소들의 데이터 타입과 크기를 갖고 있는 중첩 클래스를 생성한 후 그 클래스를 속성이 참조하는 형태를 갖는다. 이때 요소들의 데이터 타입은 동적 변수의 형태를 갖는다. 다음은 Integer 값을 갖는 요소들로 이루어진 중첩 데이터 타입이 클래스화된 예이다.

```
class Type_Aggregate
{
    o_4b size;
    o_4b type[];
}
```

앞 절에서 사용한 EXPRESS 스키마의 예에 대해 생성된 Versant 스키마는 다음과 같다.

```
class person : public PObject
{
public:   o_4b id;
      PString first_name;
      PString last_name;
      PString nick_name;
      Aggregate_birth_date birth_date;
      o_4b age;
      Aggregate_children children;
      PString hair;
      PString korean;
      person(){};
};
```

5. SDAI 오퍼레이션의 구현 방법

SDAI (standard data access interface)는 STEP 파일 [ISO, 1994c]로 표현된 데이터와의 인터페이스 오퍼레이션들을 기본적으로 제공하여야 한다. EXPRESS로 표현된 데이

터의 스키마는 초기 바인딩 시 저장하고자 하는 데이터베이스의 데이터 정의언어 문장으로 변환되며, 후기 바인딩 시 필요한 소스 파일 과 헤더 파일 그리고 초기화 파일 등이 생성된다. 실제적인 SDAI 오퍼레이션들은 두 가지 종류로 크게 나눌 수 있다. 주 기억장치에 있는 정보를 다루는 오퍼레이션과 실제 데이터베이스와의 인터페이스를 담당하는 오퍼레이션으로 나뉘며 이러한 두 가지 종류의 오퍼레이션을 통해서 모든 SDAI 의 기능들이 이루어진다.

5.1 Oracle 에서의 구현

Oracle 은 데이터 조작언어로 SQL 을 사용하며 Oracle 에서 지원하는 SQL*Plus 프로그램을 통해 사용자로 하여금 대화형 SQL (Interactive SQL)을 사용토록 하고 있다. 그러나 Oracle 의 경우 EXPRESS 의 데이터 계층적 구조를 지원하지 않으며 SQL*Plus 이외에 프로그래머가 직접 데이터베이스에 접근할 수 있는 방법인 Embedded SQL 의 경우 C++를 지원하지 않고 있다. 이에 대한 해결책으로 CB++ (CommonBase C++ Library [Bernhard, 1994]) 를 사용하였다. CB++ 는 C++를 지원하며 클래스를 사용하여 데이터베이스의 접근할 수 있도록 하고 있으며 SQL 문법을 이용한 데이터의 생성과 수정 및 Query 를 지원한다.

초기 바인딩 과정에서 생성되는 Oracle 용 데이터 정의 파일인 "Schemas_odl.sql" 은 EXPRESS 의 스키마 정보를 Oralce 에서 이용하도록 스키마 변환한 결과를 갖고 있다. Oralce 을 위한 데이터 정의어 파일은 "CREATE TABLE" 문장들로 이루어져 있다. 이 파일은 EXPRESS 의 스키마 정보를 담고 있는 엔티티 테이블 및 속성테이블을 구성하는 SQL 문장들이다. SQL 문장들이 의미하는 테이블들을 데이터베이스에 생성하기 위해서 Oracle 에서 지원하는 SQL*Plus 를 사용하였다. SQL*Plus 는 대화형 명령 이외에도 파일을 통한 접근이 가능하며 이때 사용하는 SQL*Plus 상의 명령어는 Start "파일이름" 즉 Start "Schemas_odl" 이다. 이때 확장자 "sql"은 자동적으로 첨가된다.

5.2 UniSQL 에서의 구현

UniSQL 은 데이터 조작 언어로 기본적으로 SQL 의 기능을 지원하면서 동시에 UniSQL 이 가진 객체 지향적인 4 가지 확장을 지원하는 SQL/X 를 사용한다. UniSQL 은 SQL/X 를 사용할 수 있는 프로그램으로서 isqlx 과 sqlx 를 지원하는데 isqlx 는 대화형 프로그램의 형태이다. 또한 프로그래머가 데이터베이스를 직접 조작할 수 있도록 API(Advanced Programming Interface) function 들을 C 와 C++ 형태로 지원하고 있다. 초기 바인딩 과정에서 UniSQL 을 지

원하는 데이터 정의어 파일이 생성된다. 이 파일의 이름은 "Schemas_vdl.sql"이며 UniSQL 에 맞는 스키마 정보를 갖고 있다. 이 스키마 정보는 EXPRESS 의 스키마 정보가 UniSQL 에 맞게 변환된 결과이며 일반적으로 "CREATE CLASS" 문장들로 이루어진다.

스키마정보에 맞게 엔티티 테이블 (UniSQL 에서는 클래스라 부르기도 한다) 들을 구성하도록 되어 있는 SQL/X 문장들은 UniSQL 에서 제공하는프로그램인 isqlx 혹은 sqlx 를 통해 데이터베이스 상에서 수행된다. 이때 sqlx 의 명령어는 sqlx 뒤에 옵션으로 -i 뒤에 스키마 파일 이름 즉 "Schemas_vdl.sql" 을 주면되며, isqlx 의 경우 isqlx 상에서 read 명령어로 스키마 파일을 읽어 run 명령어로 실행하면 데이터베이스에 엔티티 테이블이 생성된다.

데이터베이스로의 데이터 저장과 데이터베이스로부터의 정보 추출은 SQL/X 문장들을 통해 이루어지며 이러한 문장들은 UniSQL 에서 지원하는 C++ API function 들을 통해 수행된다. SQL/X 가 SQL 을 포함하고 있으므로 UniSQL 에서만 지원하는 타입들 - SET, MULTISET, SEQUENCE - 을 포함하지 않는 엔티티 테이블은 Oracle 과 마찬가지 형태로 정보를 저장, 추출할 수 있다. 모든 수행의 기본 단위는 엔티티이며 저장의 경우는 "INSERT INTO" 문장을, 정보 추출의 경우에는 "SELECT" 문장을 사용한다. Versant 는 Oracle 과는 달리 계층적 정보 구조를 지원하므로 엔티티를 속성으로 갖는 경우와 계승 관계를 갖는 엔티티들의 경우에 있어 보다 용이하게 정보를 저장, 추출할 수 있다.

5.3 Versant 에서의 구현

Versant 는 데이타 조작언어로 C++/VERSANT 를 사용한다. C++/VERSANT 는 각종 데이터의 저장 , 접근 및 추출을 지원하는 클래스들로 이루어져 있다. 이러한 구조는 앞서 설명한 Oracle 과 UniSQL 의 구현에서처럼 SQL 이나 SQL/X 언어로써의 접근이 아닌 C++ 형태의 접근을 필요로 함을 의미한다. C++/VERSANT 의 구조는 C++ 과 유사하며 데이터베이스 접근에 필요한 상위 클래스들을 두고 있다. 표 5 는 관계형 데이터베이스 시스템에 대응되는 C++/VERSANT 를 나타내고 있다.

<표 5> VERSANT 와 관계형 시스템의 비교

C++/VERSANT	Relational SYSTEM
Database	database
PDOM class	RDBMS Interface
PClass class	table
Pobject class	record, row
Class 속성	column

초기 바인딩 시 생성된 스키마 정보는 C++/VERSANT 를 위한 소스 파일 , 헤더 파일 및 초기화 파일로 변환된다. Versant 용 컴파일러는 이러한 파일들을 컴파일하여 실행 파일을 생성하며, 생성된 실행 파일을 실행함으로써 데이터베이스에 대한 접근이 가능하게 된다. 이러한 형태는 SDAI 바인딩과 유사한 형태이다.

EXPRESS 의 스키마 정보는 Versant 에 맞는 스키마 헤더 파일로 변환된다. 이 파일은 EXPRESS 스키마가 가지고 있는 엔티티들에 대응하는 엔티티 클래스들을 선언하고 있다. 모든 엔티티 클래스들은 C++/VERSANT 에서 지원하는 클래스들로부터 상속 받아야 한다. C++/VERSANT 컴파일러는 이 헤더 파일을 이용하여 컴파일 시 sch 라는 확장자가 붙은 Versant 용 스키마를 생성한다. 이 스키마를 데이터베이스 상에 생성하기 위해서 Versant 는 "sch2db" 라는 프로그램을 지원하고 있다.

C++/Versant 는 기존의 C++ 이외에 데이터베이스의 접근을 위한 클래스들과 몇 가지 예약어를 가지고 있다. Persistent 는 데이터베이스에 저장될 클래스 및 변수들을 생성할 때 사용하는 예약어이다. Persistent 로 생성된 클래스의 경우 트랜스액션의 커밋(commit) 시 데이터베이스에 그 값들을 저장한다.

데이터베이스에 저장되어 있는 데이터들의 접근은 일반적인 C++에서의 접근과 같다. 클래스가 갖고 있는 속성에 접근하는 방법에는 두 가지가 있다. C++ 과 마찬가지로 public 으로 선언된 속성의 경우에는 직접적인 접근이 가능하며, private 로 선언되어 있는 경우에는 그 클래스의 멤버 함수를 이용한 접근만이 가능하다. 따라서 public 으로 선언된 경우의 데이터 저장은 일반적인 C++에서의 변수에 값을 대입하는 형식으로 저장하고 추출은 단순히 변수를 참조하는 것으로 이루어진다.

5.4 데이터 오퍼레이션

STEP 데이타 인터페이스를 위한 다중 데이타베이스 시스템의 조작은 엔티티의 모임인 모델을 이용한다. 대부분의 조작은 엔티티의 리스트를 대상으로 이루어지며 화일 또는 Oracle, UniSQL, 그리고 Versant 와의 인터페이스를 담당하는 함수들이 호출된다. 사용자는 특정한 데이타베이스 시스템을 지정하여 조작이 가능하며 모델과 그 모델을 저장하고 있는 데이타베이스를 표시하는 Info 화일을 이용하여 데이타베이스로부터 독립하여 조작할 수 있다.

5.4.1 단일 모델의 조작

모델은 하나의 데이타베이스 시스템에 저장되며 하나의 모델이 복수의 데이타베이스 시스템에 저장될 수 없다. 따라서 단일 모델의 조작은 단일 데이타베이스와의 인터페이스를 의미한다. 새로운 모델을 생성하였을 경우 저장할 데이타베이스를 선택할 수 있으며 특별이 지정하지 않은 경우에는 기본적으로 Oracle 이 선택된다. 데이타베이스의 지정은 SdaiModel 클래스의 멤버 함수인 SetRepository 로 이루어지며 SetRepository 함수는 R_ORACLE, R_UNISQL, 그리고 R_VERSANT 와 같은 매개 변수를 가지고 호출된다.

```
SdaiModel single_model;
single_model.ReadFromFile("single.spf");
single_model.SetRepository(R_VERSANT);
...
single_model.WriteRepository();
```

위 코드는 "single.spf" 라는 STEP 화일로부터 데이타를 읽어 저장소로 Versant 를 지정하여 저장하는 예이다. 이때 모델은 고유 이름으로 "single.spf" 를 가지게 되며 데이타베이스로부터 "single.spf" 라는 모델을 추출할 경우에는 다음과 같은 형태를 가진다.

```
SdaiModel single_model;
single_model.ReadFromRepository("single.spf");
```

이때 "single.spf" 라는 이름의 모델은 Info 화일을 참조하여 자동적으로 Versant 로부터 추출된다. 모델의 이름은 필요에 따라 Name() 함수를 이용하여 변경할 수 있다.

모델은 엔티티 리스트를 포함하고 있으며 일반적인 작업의 형태는 다음과 같다.

```
SdaiEntityListH example_list = single_model.GetEntityList();
SdaiEntityH example_entity = example_list->Rewind();
while( example_entity)
{
    // 엔티타의 조작
    example_entity->Next();
}
```

5.4.2 다중 모델의 조작

여러 개의 모델을 동시에 이용하는 조작을 말한다. 조작에 사용되는 모델의 수 만큼 모델의 선언이 필요하며 단일 모델의 조작에서와 마찬가지로 모델의 고유 이름을 통하여 데이타베이스와 독립된 형태로 작업을 수행할 수 있다. 예를 들어 "적장" 이라는 속성을 포함하는 "사람" 이라는 엔티타들의 모임인 "person model" 모델이 Oracle 에 저장되어 있고 "주소" 라는 속성을 포함하는 "회사" 라는 엔티티들의 모임인 "company model" 모델이 Versant 에 저장되어 있을 때 사용자는 모델의 저장소와 상관없이 다음과 같은 조작을 통해 특정한 사람의 직장 주소를 추출할 수 있다.

```
SdaiModel A_model;
SdaiModel B_model;
A_model.ReadRepository("person model");
B_model.ReadRepository("company
model");
SdaiEntityListH personList =
A_model.GetEntity("person");
SdaiEntityListH companyList =
B_model.GetEntity("company");
SdaiEntityH a_person =
    personList->Rewind();
while(a_person)
{
    ...
    // 특정한 사람의 검색 작업
    SdaiString str =
    a_person->GetAttr("company");
    SdaiEntityH b_company =
    companyList->Rewind();
    while(b_company)
    {
        if(!strcasecmp(
b_company->GetAttr("name") , str))
        {
            printf(" Person Name : %s,
            Company Address : %s \n",
            a_person->GetAttr("name"),
        b_company->GetAttr("address"));
            ... //기타 원하는 작업
        }
        b_company =
            companyList->Next();
    }
    a_person = personList->Next();
}
```

6. 데이터 모델의 비교

SDAI 의 구현에 있어서 각 데이터 모델 간의 차이점은 표 6 과 같이 요약될 수 있다. 이 비교는 본 연구에서 사용한 각 데이터베이스 시스템의 기능을 중심으로 작성되었으나, 다른 데이터베이스 시스템도 데이터 모델에 따라 유사한 성격을 나타낸다.

<표 6> 데이터 모델의 비교

	관계형	확장 관계형	객체지향형
인터페이스 함수 형태	문자열 형태의 질의	문자열 형태의 질의	C++ 프로그램
데이블 이름 변환의 필요	Yes	No	No
계승 관계 지원	No	Yes	Yes
중첩 타입을 위한 별도의 테이블/클래스의 필요	Yes	No	Yes
스키마 변환	Complex	Simple	Simple
인터페이스 함수의 크기	Large	Medium	Small
인터페이스 함수의 간결성	Complex	Medium	Simple

표 6에서와 같이 데이터베이스 인터페이스 함수의 형태는 관계형과 확장관계형 모델에 대하여 문자열 형태의 질의를 사용하는 반면 객체지향 모델에서는 영구 변수에 대한 C++ 문장을 사용한다. C++ 문장은 컴파일 과정을 거쳐 실제로 데이터베이스와 데이터를 교환하는 반면 문자열 형태의 질의는 문자열을 분석하는 인터프리터를 통해야 하는 한 단계의 과정이 더 요구된다. 이는 실행 속도에도 영향을 미쳐 별도의 질의어 번역 과정을 필요로 하지 않는 객체지향형에서의 처리 속도가 관계형과 확장관계형에 비해 빠른 처리 속도를 보인다. 또한 관계형과 확장관계형의 경우 데이터베이스 Query 언어인 SQL 과 SQL/X 를 알아야 하는 반면 데이터베이스 인터페이스 함수의 형태가 C++ 문장 형태를 갖는 객체지향형에서는 별도의 질의 언어를 사용하지 않고도 C++ 문장 형태만으로 조작이 가능하다. 이는 C++에 익숙한 프로그래머에게는 유리하게 작용하지만 기존의 관계형 데이터베이스 사용자에게는 선언적인 SQL 대신 순차적인 프로그램 언어를 이용해야 하는 부담이 된다.

다음으로 각 데이터 모델을 대상으로 EXPRESS 의 데이터 타입 중 중첩 타입의 지원 방법을 비교한다. 관계형에는 이를 직접 지원하는 타입이 없어 중첩 타입을 위한 새로운 테이블이 요구되며, 확장관계형의 경우에는 SET, MULTISET, SEQUENCE 등의 타입으로 대체 가능하다. 객체지향형의 경우에도 Set, Array, List, Dictionary 를 제공하고 있다. 중첩 타입을 지원하지 않는 관계형의 경우 EXPRESS 의 중첩 타입을 데이터베이스에 저장하기 위해서 중첩 데이터 타입을 위한 별도의 테이블을 생성하고 저장하였다. 반면 확장관계형과 객체지향형의 경우 EXPRESS 의 중첩 타입과 유사한 타입들을 데이터베이스에서 지원하므로 1 대 1 관계로 쉽게 변환할 수 있다. 그러나 중첩 타입을 위해 별도의 테이블을 갖고 있는 관계형에서는 데이터베이스와의 데이터 교환 시 엔티티 테이블과 중첩 테이블을 결합하는 조인 오퍼레이션의 수행이 필요로 하게 된다. 이는 데이터베이스 인터페이스 함수가 확장관계형과 객체지향형에 비해 상대적으로 복잡해짐을 의미하며 처리 속도 또한 상대적으로 늦음을 말한다.

클래스간의 계승 관계 역시 관계형에서는 지원하지 않으며, 확장관계형과 객체지향형에서는 지원하고 있다. 계승 관계를 지원하지 않고 있는 관계형의 경우 자신의 속성뿐만 아니라 상위 클래스의 속성들 또한 포함하고 있어야 하므로 스키마 변환 시 상위 엔티티를 참조하여 테이블을 생성하는 과정이 요구된다. 이러한 계승 관계 및 중첩타입의 지원 여부로 인해 스키마 변환 시 관계형은 복잡한 과정이 요구되며 확장관계형과 객체지향형은 비교적 간단하다.

7. 결론

본 논문에서는 생산정보의 공유 및 교환을 위한 국제 표준인 STEP 데이터와 표준 인터페이스인 SDAI 의 구현 방법을 서로 다른 데이터 모델을 사용하는 데이터베이스 시스템에서 구현하고 이러한 구현 과정을 비교하였다. 데이터 모델로는 현재 가장 널리 쓰이고 있는 관계형 모델, 객체지향 모델링 기능을 제공하는 객체지향형 모델, 그리고 관계형 모델에 객체지향 기능을 추가한 확장관계형 모델을 이용하였다. 본 논문에서는 EXPRESS 에서 지원하는 중첩 데이터 타입과 클래스 간의 속성 계승 관계를 이러한 세가지 데이터 모델로의 변환 방법을 중심으로 변환 과정을 비교 설명하였다.

EXPRESS 언어는 정보 모델 뿐만 아니라 함수(function 또는 procedure)와 규칙

(rule)을 이용하여 데이터 무결성 제약조건 (integrity constraints)을 정의할 수 있다. 다수의 응용프로그램 간의 많은 데이터 교환이 이루어지는 통합 환경에서 데이터 무결성 제약조건의 확인은 매우 중요하다. 향후 연구로서 본 연구에서 구현한 SDAI에 이러한 제약조건의 확인과 동시성 제어를 포함하여 분산 컴퓨팅 환경에서의 통합된 설계 및 생산 시스템의 구현이 필요하다 [M.Hardwick, 1996] [M.Marache, 1997].

참 고 문 헌

[김기환, 1996] 김기환, 유상봉, 차상균, *표준 제품 데이터 인터페이스 구현*, 정보과학회논문지 (C), 제 2 권, 제 3 호, 1996, 9, pp. 233 – 242.

[도남철, 1993] 도남철, 최인준, 김광수, "Implementation of PDES/STEP in an Object-oriented Databases," 대한산업공학회 1993 년 추계학술대회논문집, 서울, 1993.

[도남철, 1996] 도남철, *능동적 객체 지향형 데이터베이스에서 사용자 정의 제약 조건의 역방향 전달*, 박사학위논문, 포항공과대학교 산업공학과, 1996.

[Clack, 1992] S. N. Clark, *Fedex: The NIST EXPRESS Translator*, National Institure of Standards and Technology, USA, 1992.

[Elmasri, 1994] R. Elmasri and S. Navathe, *Fundamentals of Database Systems*, Second Edition, The Benjamin/Cummings Publishing Company, Redwood City, California, 1994.

[Hardwick, 1996] M. Hardwick et al., "Sharing manufacturing information in virtual enterprises", Communications of the ACM, Vol 39, pp. 46 – 54, 1996.

[ISO, 1994a] ISO, *Industrial Automation Systems and Integration – Product Data Representation and Exchange – Part 1: Overview and Fundamental Principles*, Geneva, 1994.

[ISO, 1994b] ISO, *Industrial Automation Systems and Integration – Product Data Representation and Exchange – Part 11: Description Methods: The EXPRESS Language Reference Manual*, Geneva, 1994.

[ISO, 1994c] ISO, *Industrial Automation Systems and Integration – Product Data Representation and Exchange – Part 21: Implementation Methods: Clear Text Encoding of the Exchange Structure*, Geneva, 1994.

[ISO, 1996] ISO, *Industrial Automation Systems and Integration – Product Data Representation and Exchange – Part 22: Implementation Methods: Standard Data Access Interface Specificaiton (Draft International Standard)*, Geneva, 1996.

[Lofferedo, 1994] David Lofferedo and Martin Hardwick, "Efficient Database Implementaion of EXPRESS Information Models", procedings of EXPRESS User Group International Conference, 1994.

[Marache, 1997] M. Marache et al., "A CORBA-based infrastructure managing STEP distributed models

for virtual reality applications," Proceedings of European Conference on Product Data Technology Days 1997, pp. 119-128, 1997.

[Metz, 1989] W. P. Metz and K.C. Morris, *Translation of an Express Schema into SQL*, PDES Inc. internal document, 1989.

[Mosbacher, 1990] Robert A. Mosbacher, *Translating Express to SQL*, NISTIR 90-4341, NIST, 1990.

[Oracle, 1990] Oracle Coperation, *SQL Language Reference Manual*, 1990.

[Rando, 1994] T. Rando and L. McCabe, "Mapping EXPRESS/SDAI into the CORBA Standard," *Proc. Third International Conf. of EXPRESS User's Group*, 1994.

[STEP Tools, 1993a] STEP Tools Inc., *ST-Developer Reference Manual*, 1993.

[STEP Tools, 1993b] STEP Tools Inc., *ST-Oracle Reference Manual*, 1993.

[Strassl, 1994] Bernhard Strassl, *CommonBase Database Access Library for C/C++* , University of Vienna, 1994.

[UniSQL inc, 1992] UniSQL inc, *Interactive SQL/X*, 1992.

[Versant, 1994] VERSANT Object Technology, *C++/VERSANT Usage Guide* , 1994.

[Whitehead, 1991] Curtis Whitehead, *STEP implementation Prototype Package for LEVEL III Database Prototype using OODB Technology*, Digital Equipment Corporation, 1991.

[Wilson, 1995] P. Wilson, "EXPRESS Tools and Services," *Proc. Fourth International Conf. of EXPRESS User's Group*, 1995.

저자소개

권용국

권용국은 인하대학교 자동화공학과에서 석사학위를 취득하였으며, 두산기계의 FA 팀에서 연구원으로 재직 중에 있다. 주요 관심 분야는 데이터베이스, 생산 자동화, 네크워크, CIM 등이다.

유상봉

유상봉은 퍼듀대학교에서 전기 및 컴퓨터공학 박사학위를 취득하였으며, AT&T Bell 연구소, 삼성 전자 컴퓨터 부문 등을 거쳐 현재는 인하대학교 자동화공학과 부교수로 재직중에 있다. 주요 관심 분야는 공학 데이터베이스, 객체 및 지식 베이스, 시스템 통합 등이다.