CIM DB 기술과 응용 사례

차상균 / 유상봉

서울대학교 제어계측공학과 및 제어계측신기술연구센터 / 인하대학교 자동화공학과

1. 서 론

최근 컴퓨터 및 통신 기술의 급속한 발전과 보편화로, 전 세계적으로 정보의 수집, 가공, 유통이 신속하게 이루어지는 하부구조가 갖추어지고 있다. 이러한 하부구조는 특히 최근 출범된 WTO 체제와 함께, 생산 기업에게 변동하는 시장 여건에 따라 능동적으로 대처할 수 있는 체계를 요구하고 있다. 컴퓨터 통합 생산(CIM) 시스템은 제품 설계, 물류, 생산 등 기업 활동에 대한 단위 자동화 및 정보화는 물론 이를 관장하는 단위 시스템 및 외부 기관간의 정보의 흐름을 원활하게 하여 기업의 총체적 생산성 제고를 그 목적으로 한다.

CIM은 추진 기업의 관심에 따라 다음과 같이 몇 개의 적용 대상 영역으로 나누어지는데, 기업내에서 상대적 중요도에 따라 우선 순위를 정하여 단계별 추진이 가능하다.

(1) 유연생산시스템(FMS) 구축 : 제품의 실제 생산을 담당하는 생산라인의 자동화, 정보화 및 관련 활동과의 유기적 연계를 목적으로 함[1].

(2) 동시공학(concurrent engineering) 적용 : 컴퓨터상의 가상 공장(virtual factory) 환경 구축을 통해 제품 설계와 공정 설계의 병렬 수행을 추구.

(3) 기업통합(enterprise integration) : 원활한 정보 소통을 통해 기업내 설계 생산 관련 조직은 물론 여타 조직과의 유기적 통합을 추구[2], [3].

(4) 전자상업(electronic commerce) : CALS(Continuous Acquisition and Life cycle Support 또는 Computer-Aided Logistics Support)를 기반으로 관련 기업간의 원활한 정보 교환 체계 구축을 추구[4].

데이타베이스(DB) 기술은 이러한 CIM 추진시 고려하여야 할 각종 데이타 및 지식의 모델링, 공유, 저장 관리에 관련된 기술로서 성공적인 CIM 시스템의 구축 운용에 핵심적 역할을 한다. CIM DB는 재무 관리, 인력 관리 등 전통적인 DB 응용분야의 요구 사항뿐만 아니라, 다음과 같이 CIM에 고유한 기능을 필요로 한다.

(1) 복합 객체의 모델링 및 저장 : CAD/CAM 데이타, 멀티미디어, 기술 문서 등 비전통적인 복합 데이타의 표현 저장.

(2) 버전 관리 : 엔지니어링 과정에서 복수의 선택 사양을 시도해 볼 때 생성되는 버전 계층의 관리. 버전의 branching 및 merging 기능이 요구됨.

(3) 장기간 지속되는 트랜잭션(long-running transaction)의 지원 : 각종 설계 생산 활동은 순간적으로 종료되는 전통적 DB 트랜잭션과 달리 장시간 지속되는 경우가 많은데, 이런 활동을 DB 트랜잭션으로 지원.

이 외에도 생산 라인의 자동화 설비로부터 입력되는 각종 센서 데이타를 실시간 트랜잭션으로 처리하는 기능이 하위 공정 DB에서 요구된다.

본 고에서는 요소 기술 측면에서 DB 기술의 현황을 소개하고 CIM 응용 사례를 기술한다. 2장에서는 DBMS의 일반적 기능과 데이타 모델의 발달에 따른 DBMS 기술의 발전에 대해 간략하게 기술하며, 3장에서는 최근 CIM, 멀티미디어 데이타 처리 분야에서의 비전통적인 DB 요구사항을 충족시키는 객체지향 소프트웨어 및 DB 패러다임에 대해 소개한다. 4장에서는 FMS의 운용 관리를 위한 객체지향 DB 구축 사례에 대해 기술하며, 5장에서는 CALS에서 생산 데이타 교환 표준으로 채택하고 있는 ISO STEP 표준 DB 구현 사례를 소개한다. 마지막으로 6장에서 결론을 기술한다.

2. DB 기술의 현황

2.1 DBMS의 일반적 기능

DB 시스템은 일반적으로 DBMS(DataBase Management System)라는 부르는 시스템 소프트웨어에 의해 생성되고 관리되는데, 상용 DBMS는 대체로 다음과 같은 기능을 제공한다.

(1) 데이타 정의어(DDL : Data Definition Language) 및 조작어(DML : Data Manipulation Language) : 조작어는 선언적 언어(declarative language)가 바람직한데, 이 경우 사용자는 어떤 데이타를 원하는지만 정해주고 시스템이 알아서 최적의 계획으로 원하는 데이타를 찾아낸다.

(2) 트랜잭션 관리 : 모든 DB 접근 요구는 트랜잭션 단위로 처리되며, 트랜잭션은 그 내용이 모두 수행되거나, 수행이 아예 안되었던 것으로 처리되어야 한다(원자성). 또한 동시에 다수의 트랜잭션이 처리될 때, 한 트랜잭션의 입장에서 보면 마치 혼자서 수행되는 것 같이 되도록 동시성 제어를 한다. 트랜잭션은 하나의 일관성 있는 상태 변환 과정이 되어야 하며, 시스템에 이상이 있더라도 일관된 상태로 되돌아갈 수 있도록 회복 기능을 갖추어야 한다.

(3) 질의 처리 및 프로그래밍 인터페이스 : 선언적 질의를 최적화 처리하는 모듈과 C와 같은 일반 프로그래밍 언어와의 인터페이스를 제공한다.

(4) 응용 소프트웨어 개발 환경 : 4GL, 객체지향 소프트웨어 개발 환경 등을 제공하여 DB 응용 소프트웨어를 경제적으로 개발할 수 있도록 한다.

2.2 데이타 모델에 따른 DBMS 분류

DB 시스템 개발의 첫 단계는 대상 분야의 실세계에 존재하는 엔터티(entity)와 그들간의 관계를 추상화하여 DBMS의 DDL로 표현하여야 한다. 현재 이 모델링 과정을 도와주는 많은 소프트웨어 공학 툴들이 상용화되어 있으며, 일부는 대상 DBMS의 DDL로 표현된 스키마나 응용 프로그램 코드를 자동적으로 생성하기도 한다. DB 기술은 실세계 모델을 어떤 표현 요소로서 기술하느냐에 따라 다음과 같은 순서로 발전하여 왔다.

(1) 계층형(Hierarchical) DB : 60년대에 개발된 IBM IMS가 대표적인 DBMS로서 연관된 데이타를 트리 구조로 모델하도록 한다. 부품 소요 명세를 산출하는데 필수적인 BOM(Bill of Materials)이 대표적인 예인데, 제품의 생산에 소요되는 부품 및 어셈블리를 트리 구조로 모델한다.

(2) 망형(Network) DB : 계층형 모델을 확장하여 데이타간의 관계를 임의의 망형 구조로 표현할 수 있도록 한다. 동일 유형의 데이타를 환(ring)으로 엮어 집합(set)을 만든다.

(3) 관계형(Relational) DB : 70년대초 정립된 수학적 모델을 기반으로 모든 데이타를 테이블 형태의 자료 구조로 쪼개어 저장 관리하는 것을 원칙으로 한다. [5] 예컨데 트리 구조의 BOM 데이타도(어셈블리 ID, 부품 ID, 소요량)의 세 필드로 이루어지는 테이블에 플랫하게 저장된다. 70년대 프로토타이프 개발, 80년대 상용화를 거쳐 현재 비지니스 데이타 처리에서 가장 선호되는 DBMS이다. 선언적 질의 언어(SQL)를 지원한 최초의 모델로서, Oracle, Ingres, Informix, Sybase, DB2(IBM)가 대표적인 상용 관계형 DBMS이다.

(4) 객체지향형(Object-Oriented) DB : 설계 및 생산 데이타, 멀티미디어 데이타, 문서 관리 등 관계형 모델에 적합하지 않은 분야의 요구 조건을 고려하고, 객체지향 프로그래밍(OOP)의 장점을 지원하기 위해 80년대 중반부터 개발되었다.[6] C++와 같은 OOP 언어와의 유연한 통합(seamless integration)을 지원한다. 주요 상용 OODBMS로 ObjectStore(Object Design Inc.), Objectivity/DB, Versant, O2, Ontos 등이 있으며, 이들이 주축이 된 ODMG(Object Data Management Group)에서 산업체 표준[7]을 93년말 제정하여 현재 이 표준에 준하는 제품들이 출시되고 있다. 이외에 관계형 DB의 데이타 모델링상의 약점을 보강하면서 관계형 DBMS와 위에서 언급한 OODBMS 중간 위치에 있는 제품으로 UniSQL, Illustra 등이 있다.

3. 객체지향 소프트웨어 및 DB 기술

3.1 객체지향 소프트웨어 패러다임

3.1.1 클래스 및 객체식별자

객체지향 접근방식[8]에서는 문제 영역을 객체(object)들의 집합으로 보고, 같은 성질을 갖는 객체 그룹을 하나의 클래스(class)로 모델한다. 이 클래스는 프로그래밍 시스템 내에서 하나의 type이 된다. 객체는 데이타와 이들 데이타를 조작할 수 있는 함수들(method)로 구성되는데 데이타 부분은 객체가 가지고 있는 특징 및 상태 정보를 모델한다. 함수 부분은 객체의 동적 특성을 모델하며, 객체의 데이타 부분을 생성하거나 조작하는 일을 담당한다.

모든 객체는 자기만의 유일한 객체 식별자(object identifier)를 가진다. 이러한 객체 식별자는 객체가 생성될 때 프로그래밍 시스템에서 자동적으로 생성한다. 객체 식별자는 유일하며, 두 객체가 같은 데이타를 가지고 있다고 하더라

도 객체 식별자가 다르면 서로 다르다.

객체지향 패러다임에서는 객체 클래스를 정의할 때 그 구성 요소인 데이타와 함수들을 공적인(public) 부분과 사적으로 은닉하는 부분으로 나누어 한 캡슐로 정의한다. 객체 이용자는 단지 공적 부분만을 통해서 객체를 접근하여야 하는데, 이 캡슐화는 객체 이용자로 하여금 최소한의 정보만 알면 객체를 활용할 수 있도록 하며, 또한 소프트웨어의 유지 보수를 용이하게 한다. 즉, 객체의 공적 부분에 대한 정의만 유지하면, 사적 부분은 프로그램의 다른 부분에 영향을 주지 않고 언제나 수정할 수 있다. 그림 1은 캡슐화 개념을 전통적인 구조화 프로그래밍과 비교한 것이다. 구조화 프로그래밍에서는 데이타가 변경되면 데이타와 분리되어 있는 관련 프로시져들을 모두 찾아 수정해 주어야 하지만, 객체지향 프로그래밍에서는 클래스 내의 함수만 변경하면 된다.

3.1.2 클래스 상속

상속(inheritance)이란 하나의 클래스에서 다른 클래스로 그 구성 요소가 계승되는 것을 말한다. 예컨데, 볼트와 너트 객체의 경우, 볼트에는 직경, 나선 간격, 길이 등의 속성이 있고, 너트에는 직경, 나선 간격, 모양 등의 속성이 있다. 이들은 직경과 나선 간격이라는 속성을 공통적으로 가지고 있다. 상속 메카니즘을 활용할 경우, 볼트와 너트라는 클래스를 정의할 때 독자적인 속성은 각각 정의하고, 공통적 속성은 어셈블리 부품이라는 상위 클래에서 정의한 후 이 클래스에서 상속을 받도록 한다. 이 때, 어셈블리 부품 클래스를 수퍼클래스(superclass)라고 하고, 볼트, 너트 클래스를 서브클래스(subclass)라고 한다. 다음은 위에서 언급한 클래스들을 C++ 스타일로 표현한 것이다.

```
class Assembly Part {
    float partDiameter;
    float threadSpacing;
}
class Bolt : public AssemblyPart {
    float partLength;
}
class Nut : public AssemblyPart {
    char partType;
}
```

3.1.3 오버로딩과 다형성

전통적인 프로그래밍 언어에서는 하나의 프로그램에 존재하는 모든 함수들이 다른 함수들과 구별될 수 있는 유일한 이름을 가지도록 요구한다. 객체지향 프로그래밍 언어에서는 하나의 프로그램에 동일한 이름을 가지는 여러 개의 함수가 동시에 존재할 수 있도록 허용한다. 한 클래스 내에서 같은 이름을 가지는 두 개 이상의 함수는 결과 값이나 인수가 서로 달라서 이들에 의하여 서로 구별된다. 서로 상속 관계에 있는 두 클래스에서는 서브클래스가 수퍼클래스의 함수와 동일한 이름을 가지는 함수를 정의하면 수퍼클래스에서 정의된 함수는 효과가 없게 된다. 이런 현상을 오버로딩(overloading)이라 한다.

다형성(run-time polymorphism)이란 두 개 이상의 클래스에서 같은 이름과, 같은 인수와, 같은 결과 타입을 가지는 함수들이 정의되어 있을 때, 한 객체가 이 이름의 함수를 호출하는 경우 그 객체에 속하는 클래스에서 정의된 함수가 수행된다는 것이다.

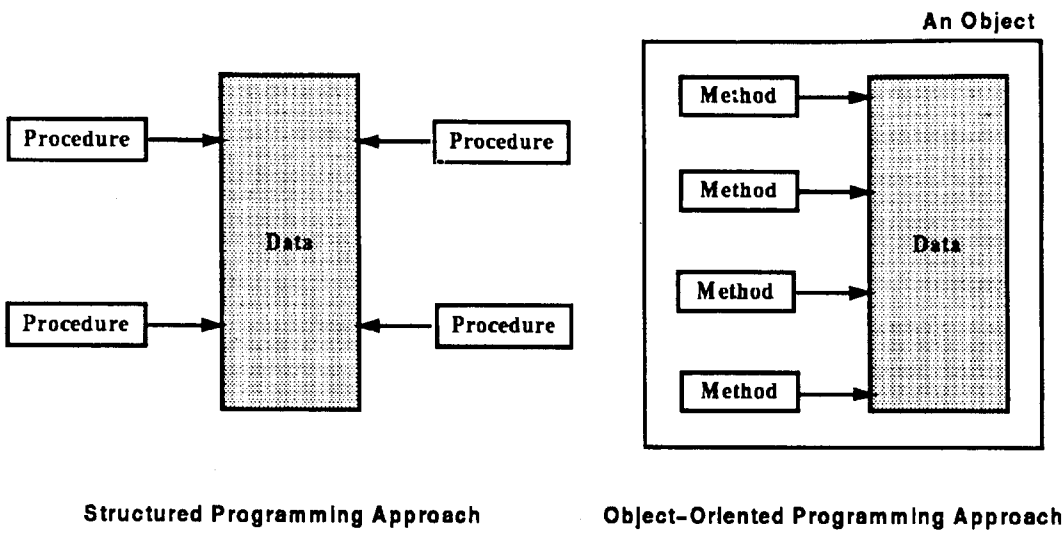


그림 1. 캡슐화

3.2 객체지향 DB

객체지향 DB는 C++와 같은 객체지향 프로그래밍 언어로 생성된 객체의 지속성(persistence) 및 여타 DBMS 기능을 지원하는 소프트웨어 시스템으로, 관계형 DBMS에 비해 프로그래밍 언어와의 유연한 통합을 제공한다. 즉, 프로그래머는 비교적 동일한 프로그래밍 모델로 데이타베이스 언어와 프로그래밍 언어를 다룰 수 있다.

객체지향 DBMS는 데이타 모델링 요소로서 지속 객체 클래스와 이진 관계성(binary relationship)을 제공한다. 이진 관계성은 프로그래머 입장에서 C++의 객체간의 포인터 또는 포인터 배열로 이해될 수 있다. 이진 관계성도 지속성을 가지며, 단방향 navigation뿐만 아니라 양방향 navigation도 가능하도록 선언될 수 있다.

객체지향 DBMS는 이외에도 설계 생산 분야에서의 장기간 지속되는 트랜잭션과 버전 관리를 지원하는 기능을 제공하며, 이런 기능들은 ODMG-93 표준의 개정판에 포함될 예정이다.

3.3 소프트웨어의 객체지향 통합

생산시스템을 구성하는 소프트웨어들은 대부분의 경우, 서로 다른 컴퓨터 환경에서 개발 및 설치되어 통신망으로 연결되어 운용되는 것이 보통이다. 따라서, 이형 컴퓨터시스템에 분산되어 있는 소프트웨어들이 유기적으로 연결되어 상호작용할 수 있도록 지원함은 생산시스템 전체의 능률을 제고하는데 큰 역할을 할 것이다. 객체지향 접근방식을 적용하여 이를 위한 연구들이 다수 진행되어 있는데, OMG (Object Management Group)의 CORBA(Common Object Request Broker Architecture), OSF(Open Software Foundation)의 DCE(Distributed Computing Environment), Microsoft의 OLE(Object Linkage and Embedding) 등은 그 예이다.

이중 CORBA에 대하여 좀더 살펴 보면, CORBA는 주요 컴퓨터 업체 등 360여개의 회원사로 구성된 OMG에 의하여 분산 환경의 소프트웨어 통합을 위해 제정된 표준이다. 그림 2는 CORBA의 구성을 보여 준다. 그림에서 Object Implementation은 다른 소프트웨어에서 필요로 하는 서비스를 제공하는 시스템을 객체지향 개념에 의거, 객체로 모델한 것이다. Object Implementation이 제공하는 서비스에 대한 인터페이스는 IDL(Interface Definition Language)을 사용하여 정의, ORB(Object Request Broker)에 등록된다. IDL은 language-neutral한 언어로 객체지향 개념을 따르기 때문에 각각의 소프트웨어 모듈에 대한 캡슐화와 상속 등을 지원한다. Client는 자신이 원하는 서비스를 ORB에 요구함으로써 실제 Object Implementation에 대한 구체적 정보없이도 서비스를 받아 작업을 수행할 수 있다. 이를 통해 상위 레벨에서는 하위 레벨의 통신, 이형의 컴퓨터 환경 등에 무관하게 상호작용 할 수 있게 된다.

이상에서 객체지향 접근방식과 이를 기반으로 한 객체지향 데이타베이스, 그리고 분산 소프트웨어의 통합에의 응용에 대하여 살펴 보았다. 객체지향 소프트웨어 기술은 소프트웨어의 설계, 구현 및 사후 유지, 보수를 용이하게 하므로, FMS 운용 관리 소프트웨어, 생산시스템 소프트웨어의 통합 관리 등에 적용할 경우 그 효용가치가 클 것으로 기대된다. 다음장에서는 개방적 FMS 운용 관리 소프트웨어 구조 도출을 위해 객체지향 개념을 적용한 사례를 기술한다.

4. 객체지향 FMS 통합

CIM 영역 중에서도 FMS는 생산 기업의 관심을 가장 많이 받는 영역이다. 시스템 통합의 관점에서 보면, FMS는 가공, 조립 등과 같은 단위 생산 공정을 수행하는 복수의 셀(FMC : Flexible Manufacturing Cell)과, 각 셀의 상태를 추적하고 제어하기 위한 제어호스트, 그리고 이들을 연결하는 통신망으로 구성된 분산 시스템이다.[1] 각 셀은 다시 로봇, 수치 제어 기기(NC machine) 등과 같이 프로그램이 가능한 생산 설비 하드웨어와 셀 제어기로 구성된다. 그림

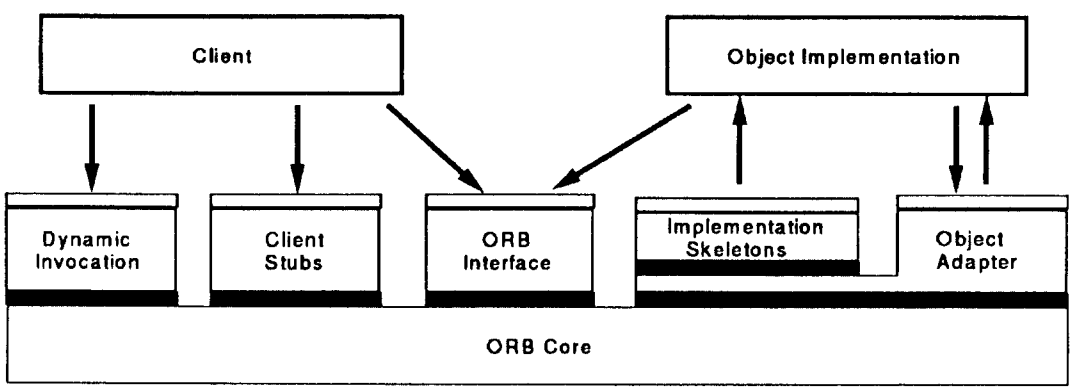


그림 2. CORBA

3이 이와 같이 계층 구조를 갖는 분산 FMS의 예를 보여 준다.

FMS의 효과적 운용 관리를 위해서는 시스템 전체를 통합, 관리하고 제어하는 소프트웨어 모듈이 필수적이다. 이 모듈은 그림 3에서 제어호스트에 내재하면서 각 셀의 작업을 계획, 지시하고, 그 상태를 감시하는데, 이를 위해서는 셀 및 생산 설비의 상태, 생산 계획 등의 정보를 통합 관리하는 DB가 필요하게 된다. 그러나 FMS에는 제품의 설계 변경으로 인한 생산 공정의 변경, 새로운 셀 추가와 같은 FMS 구성상의 변경, 태스크 스케쥴링과 같은 운용 전략상의 변경 등 다양한 변화의 가능성이 존재한다. 따라서 FMS가 원래의 목표로 설정한 유연성을 달성하기 위해서는 통합 운용 관리 소프트웨어에서 이러한 변화를 유연하게 수용할 수 있어야 한다. 즉 주어진 외적 변화를, 가능하면 프로그램의 수정없이 데이타의 추가, 삭제만으로 수용하는 것이 바람직하고, 프로그램의 수정이 불가피한 경우에도 그 일부만 국지적으로 수정하면 되도록 함이 바람직하다.

이상의 문제를 인식하고 서울대학교 자동화시스템공동연구소에서는 객체지향 개념을 FMS 운용관리 소프트웨어 시스템의 설계에 적용하여 상용 OODBMS인 Objectivity/DB [12]상에서 구현한 바 있다. 이 장에서는 이 연구의 핵심 개념과 구현된 시스템인 FREE(FMS Run-time Executive Environment)의 구조 및 DB에 대해 기술한다[9], [10].

4.1 객체 지향 FMS 통합 모델

객체지향 FMS 통합 모델의 핵심은 FMS에 실재하는 셀과 관련 엔터티, 스케쥴러 등 관심의 초점이 되는 모든 개념을 객체로 모델하고 그들 사이의 관계를 이진 관계성으로 정의해 주는 것이다. 그림 4는 이러한 모델의 일부를 보여주고 있는데, 객체 클래스는 사각형으로, 이진 관계성은 객체 클래스간의 링크로 표현하고 있다. 객체 클래스들은 또한 상속 계층 구조를 형성하게되는데, FMS 셀을 예로 들면, 각 셀이 가지는 공통된 데이타와 함수는 수퍼클래스인 Cell 클래스에 정의하고, 보다 구체적인 데이타와 함수는 Cell 클래스의 서브클래스에 정의한다. 이들 서브 클래스로부터 생성된 객체들은 실제 FMS 셀의 상태와 동적 행위를 캡슐화하여, FMS 제어 소프트웨어의 관점에서 보면 실제 셀에 대한 에이전트 역할을 수행한다. 그림 5가 이러한 개념을 보여주고 있는데, 사선 타원은 객체를, 사선 화살표는 실제 셀과 Cell 객체간의 1 : 1 대응 관계를 나타낸다. FMS의 제어는 분산된 셀과 메시지를 주고 받는 과정을 반복하

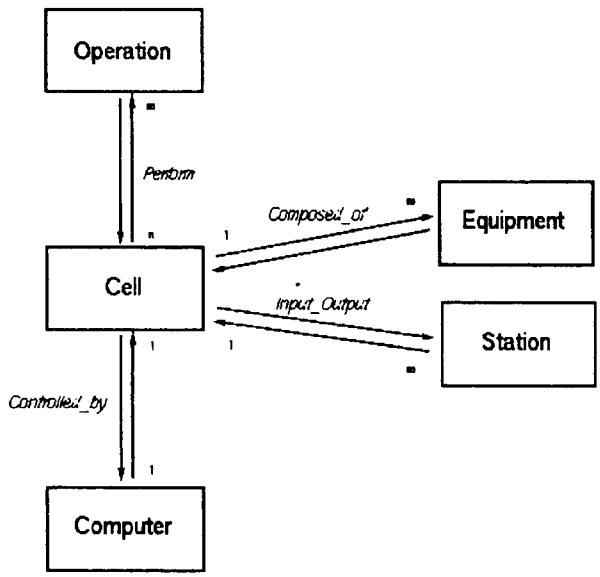


그림 4. Cell 클래스와 다른 클래스들과의 관계성

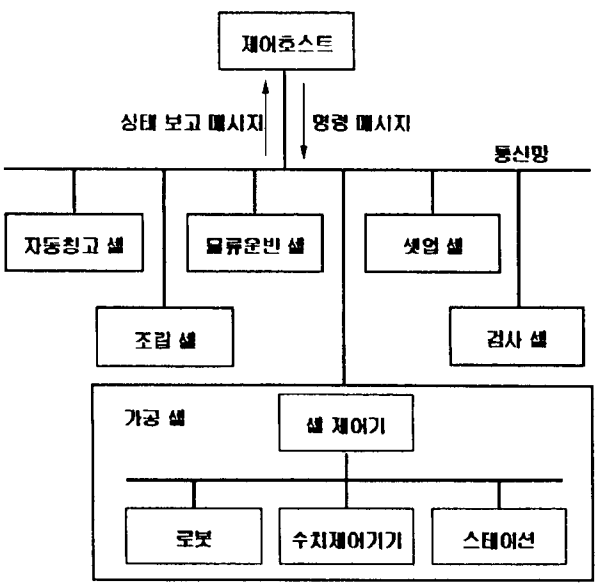


그림 3. 분산 FMS의 예

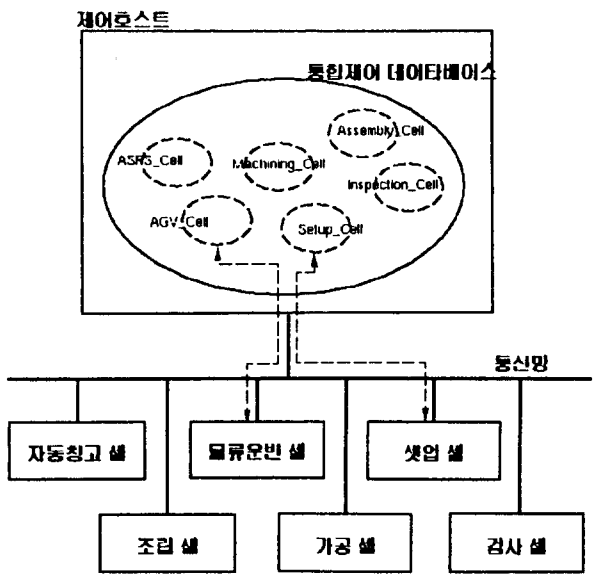


그림 5. 셀 객체

면서 이루어지는데, 통합 제어 모델에서는 이 과정이 Cell 객체와의 메시지 교환을 통한 상호작용으로 이해된다.

캡슐화와 상속을 활용한 객체지향 FMS 통합 모델에서는 FMS 구성 요소의 변경과, 제품 생산 공정의 변경, 스케쥴링 방식의 변경 등을 쉽게 운용 제어 시스템 소프트웨어에 반영할 수 있다. 예컨대, 그림 6은 Cell 클래스의 상속 계층 구조를 보여주고 있는데, 모든 Cell 객체들이 주제어 루프와의 동일한 접속 인터페이스를 공유한다. 이는 특정 FMS 셀의 구성이 변할 때 해당 객체의 수정으로만 대처할 수 있음을 의미한다. 또한 새로운 기능의 셀을 도입할 때 기존의 Cell 클래스로부터 통신 프로토콜 등 공통된 사항을 상속받음으로써 요구사항을 쉽게 소프트웨어에 반영할 수 있다.

4.2 FREE 구현

FREE 시스템은 SUN Sparcstation 10 워크스테이션을 제어호스트로 하여 Objectivity/DB[12]와 C++로 구현되었다. 제어호스트와 셀 제어기와의 통신은 TCP/IP를 이용하였으며, 오퍼레이터 모니터링 인터페이스로 객체지향 그래픽 인터페이스를 제공한다.[11]

Objectivity/DB는 기본 객체, 컨테이너, DB(database), FDB(federated database)로 이루어진 네 계층의 논리적 저장 구조를 지원하는데, 기본 객체를 모아두는 컨테이너는 디스크에서 물리적인 집중화(clustering)을 지원하며, 잠금(locking)의 최소 단위이다. DB는 컨테이너의 모임으로 각각의 DB는 하나의 파일에 대응된다. FDB는 논리적 저장 구조의 최상위 단계로 다수의 DB로 구성된다.

FREE 시스템은 서울대학교 자동화시스템공동연구소의 모델 플랜트를 대상으로 시험하였는데, 현재 하나의 FDB내에 FMS 상태 데이타베이스, 작업 일정 데이타베이스, 메시지 로그 데이타베이스를 각각 하나의 DB로 구현하고 있다. 메시지 로그 데이타베이스는 각 셀별로 컨테이너를 갖도록 하여, 셀과 주고 받는 메시지는 컨테이너 단위로 저장하도록 하였다. 이는 Objectivity/DB의 잠금의 최소 단위가 컨테이너이기 때문에 여러 셀과의 통신이 동시에 이루어질 수 있도록 하기 위함이다. 다음은 Objectivity/DB DDL로 기술된 Cell 클래스와 그 서브클래스인 ASRS-Cell 클래스의 일부를 기술한 것이다.

```
class Cell : public ooObj {
// Inherit persistence from ooObj
public :
  virtual ooStatus send_msg( ooHandle(Outgoing_
Msg)& msgH);
  virtual ooStatus interpret_msg(ooHandle(Incoming_
Msg)& msgH);
...
private :
  char name[IDSIZE];
  int status;
  char net_address[IPADDRLE-
NG]; // IP Address
  ...
  // User-Defined Relationships
  ooHandle(Station) IO_buffers[ ]
<-> CellA;
  ooHandle(Operation) Operations
[ ] <-> CellA;
};

class ASRS_Cell : public Cell {//
Inherit from Cell
public :
  void show_stored_pallets(void);
  ooStatus send_msg(ooHandle
(Outgoing—Msg)& msgH);
ooStatus store_pallet(ooHandle(Palle)& palletH);
ooStatus retrieve_pallet(ooHandle(Pallet)& palletH,
ooStatus interpret_msg(ooHandle(Incoming_Msg)&
msgH);
```

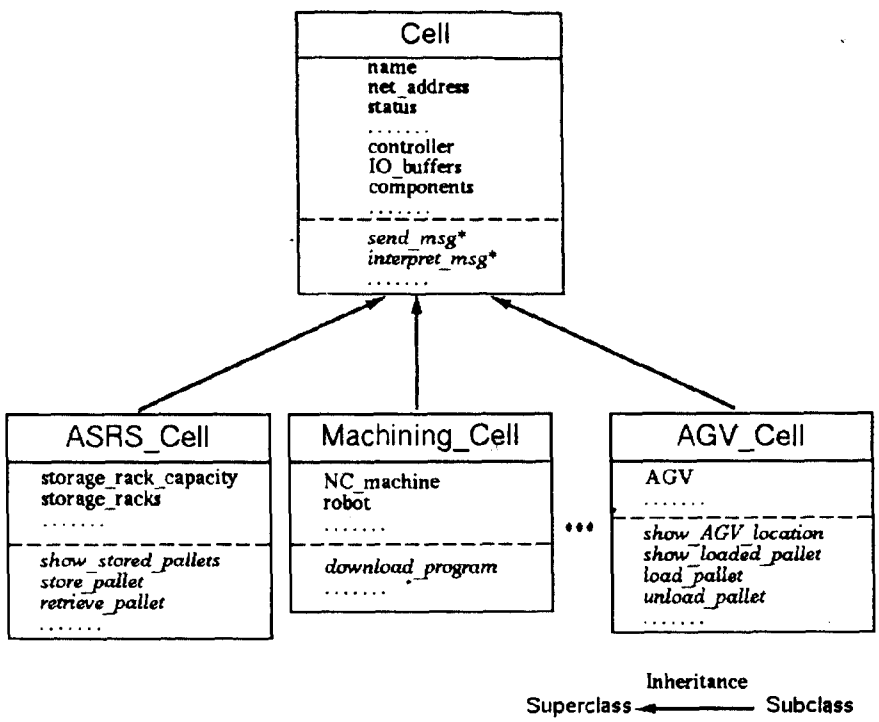


그림 6. Cell 클래스 상속 구조

```
int ID);
  ...
private :
  int storage_rack_capacity;
  int occupied_rack_count;
  ooHandle(Rack) storage_racks[ ];
  ...
};
```

여기서 Cell 클래스는 객체의 지속성을 지원하기 위하여 ooObj라는 시스템에서 정의한 지속 클래스의 서브클래스로 정의되었다. ooHandle은 지속 클래스 이름을 그 인수로 가지는데, 지속 클래스의 객체에 접근하기 위한 지속 포인터, 즉, 이진 관계성을 설정하는데 사용된다. 예를 들어 위의 Objectivity/DB 스키마에서 Cell 객체는 IO-buffers라는 이름의 링크를 통해 Station 객체와 이진 관계성을 가진다. [ ]의 존재는 관계된 객체가 여러 개임을 나타낸다. 일부 함수들은 가상 함수(virtual function)로 선언되어 있는데, 이는 Cell 클래스의 서브클래스에서 같은 이름의 함수를 재정의하여 사용할 것을 나타낸다.

5. 통합 제품 데이타베이스

CAD, CAM, CAE 등 컴퓨터를 이용한 많은 엔지니어링 소프트웨어들이 설계 생산 기업의 생산성 향상을 위하여 개발되었다. 각각의 이런 기술들은 그 동안 많은 시행착오를 거쳐 이제 어느 정도 성숙한 단계에 도달했고 생산현장에서 사용되어 그 효과를 인정 받고 있다. 그러나 이러한 툴들은 단위 과정의 자동화에는 기여하였으나, 각각이 독립된 데이타 모델과 데이타 저장소를 가지고 있기 때문에 관련 단위 과정의 툴과의 통합이 되지 않아 그 효용성에 한계가 있다.

CIM 체계에서는 모든 관련 단위 과정이 연관 데이타를 공유하여 시스템 전체 활동의 병렬도를 최대화하는 것이 바람직하다. 예를 들어 CAD로 출력된 설계를 온라인으로 CAE 시스템으로 전송, 분석이 되고, 그 결과가 CAD 시스템에 다시 피드백되도록 함이 바람직하다. 이러한 과정이 반복된 후 완성된 설계를 갖고 CAM 시스템은 로보트나 NC 공작기계의 프로그램을 생성할 수 있어야 한다. 물론 현재에도 동일 회사 제품이거나 몇몇 대표적인 제품들간에 제한된 데이타 호환 기능을 갖추고 있지만 전 생산 시스템의 통합은 아직 많은 연구가 필요하다.

CIM은 생산 시스템의 통합을 위하여 생산 활동에 포함되는 모든 정보를 관리한다. 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. 하지만 IGES의 정의 가운데 모호한 부분이 발견되어 ISO는 새로운 표준으로서 1983년에 STEP (STandard for the Exchange of Product model data)의 개발에 착수하였다.[13] STEP은 그래픽 데이타뿐만 아니라 제품의 전 생명주기에 포함된 모든 데이타를 표현하기 위하여 개발 중이며 기본적인 부분은 이미 국제 표준으로 결정되었다. STEP은 또한 미국 국방성, 상무성의 지원을 받는 CALS 이니셔티브의 한 표준으로 채택되고 있다.

STEP에 의하여 정의된 표준 데이타는 화일을 이용하여 교환할 수 있으며 데이타베이스 시스템에 저장하여 공유할 수도 있다. 이와 같이 다양한 저장 시스템(repository system)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)을 정하였다.[14] 즉, 각 데이타 베이스 시스템에서 SDAI를 구현 하였을 때 응용프로그램은 데이타베이스 시스템의 종류에 관계 없이 미리 정의된 인터페이스를 이용할 수 있어 Open CIM을 실현할 수 있다.

이 장에서는 객체지향 DBMS인 Objectivity/DB[12]를 이용하여 구현한 SDAI를 설명한다. 본 연구에서 객체지향 DBMS가 STEP의 구현에 효과적으로 응용될 수 있다는 것을 알 수 있는데 이는 STEP의 모든 정보 모델이 EXPRESS[15]라는 객체지향 모델링 언어에 의하여 정의 되어 있기 때문이다. 관계형 데이타베이스 시스템을 이용할 경우 스키마의 변환이 복잡하며 그에 따라 전체 인터페이스 프로그램도 복잡해 진다. 이러한 차이점을 설명하기 위하여 관계형 데이타베이스 시스템인 ORACLE[16]을 이용한 SDAI의 구현을 비교하여 설명한다.

5.1 제품데이타의 교환과 공유를 위한 표준 (STEP)

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

(1) 엔코딩 방식을 최적화 하여 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 프로그래밍 언어 바인딩(Language Bindings)을 개발하여 응용프로그램 인터페이스를 표준화한다.

5.1.1 STEP의 구조

STEP은 그 범위가 방대하여 부분에 따라 연구 진척에 상당한 차이가 있으며 어떤 부분은 벌써 IGES에서 수행된 것도 있다. 따라서 STEP은 그림 7과 같이 여러 파트로 분류하여 개발되고 있다. 파트 21은 화일 양식을 정하고 파트 22는 SDAI의 기능과 C, C++, Fortran 등으로 쓰인 프로그래밍 언어 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 생산같은 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 EXPRESS로 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 batch 프로세스에 의하여 이루어지는 구성이다. 이것은 현재 IGES에 의한 방법과 유사하다. 그림 8에서 응용프로그램 A(예를 들어 하나의 CAD 시스템)가 STEP에 의하여 정의된 포맷에 맞추어 프로덕트 데이타를 생성한다. 이 화일이 응용프로그램 B로 옮겨져서 B에 알맞는 데이타 구조 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 9에서 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 PDD(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 10에 나타난 것과 같이 응용프로그램들은 공통 데이타베이스에 연결되어 필요한 정보를 가져간다. 예를 들면, 한 데이타베이스 시스템에 여러 개

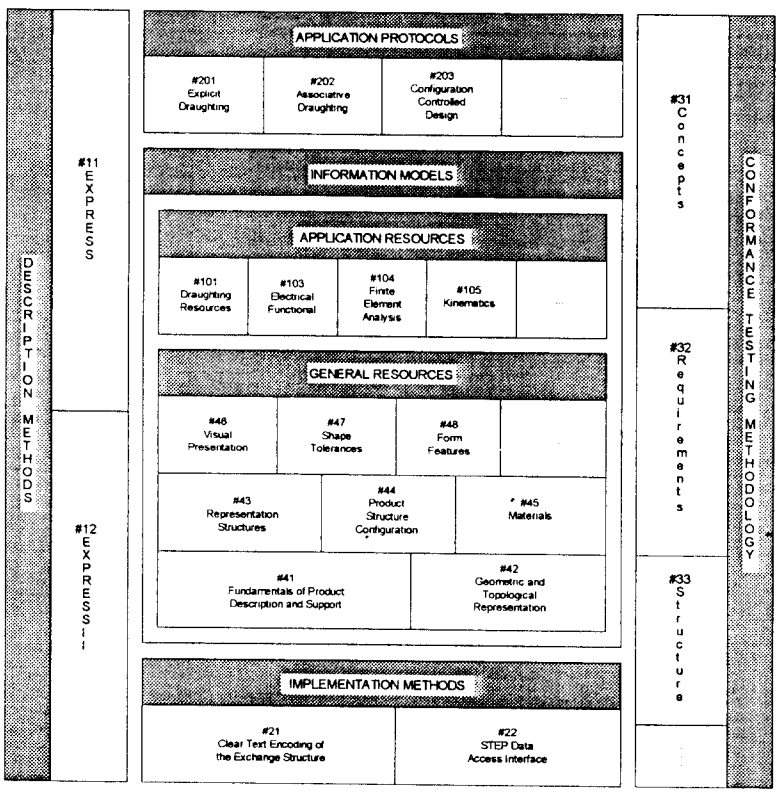


그림 7. STEP의 구조

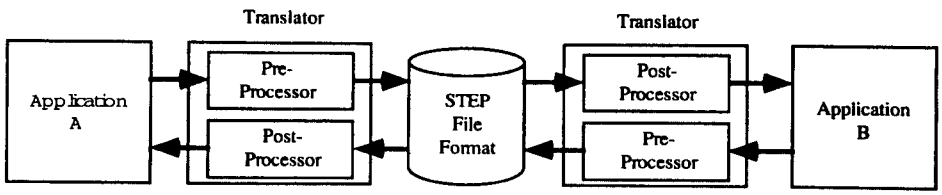


그림 8. STEP의 레벨 1 구현 : Passive File Exchange

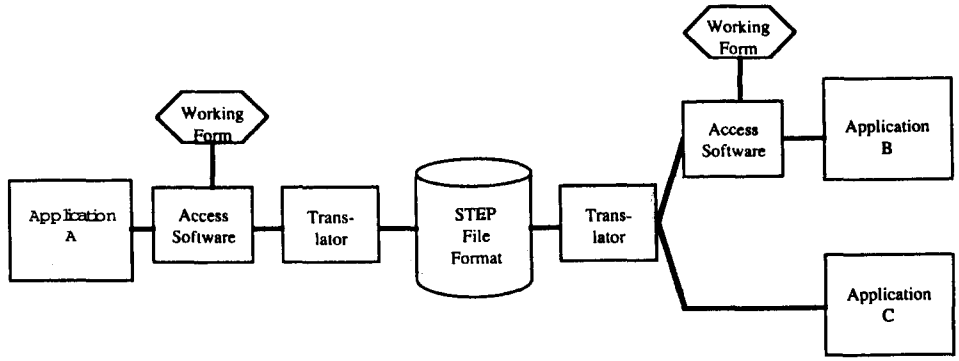


그림 9. STEP 레벨 2 구현 : Active File Exchange

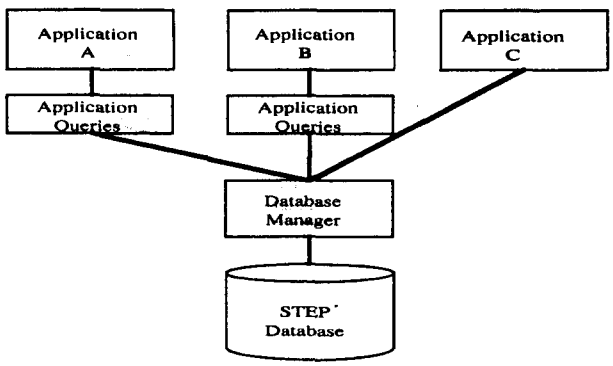


그림 10. STEP의 레벨 3 구현 : Shared Database

의 응용프로그램을 동시에 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 요청할 수도 있다. 미국의 Rockwell에 의하여 개발된 미국 공군의 IDS(Integrated Design System)는 이러한 시스템의 좋은 예이다.

5.1.2 EXPRESS 정보 모델링 언어

EXPRESS는 STEP에서 데이타의 논리적 스키마를 표현하는 언어로서, 기존의 Entity-Relationship 모델의 확장이고 객체지향 패러다임을 따르고 있다. 본 절에서는 EXPRESS의 언어요소를 간단히 소개한다.

가. 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.

나. 엔터티

엔터티의 정의는 타입 계층(type hierarchy), 속성(attribute), 지역 룰(local rule)을 포함한다. 타입 계층은 SUBTYPE이나 SUPERTYPE으로 정의한다. 속성에는 Explicit, Derived, Inverse의 세 가지가 있다. 지역 룰은 엔티티 타입에 속한 각 객체에 적용하는 제약조건(constraints)을 명시하며, UNIQUE절이나 WHERE절에 의하여 정의한다.

다. 오퍼레이터(Operator)

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

라. 실행문(Executable Statements)

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

마. 함수(Function)

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

바. 전역 룰(Global Rule)

전역 룰은 하나 혹은 여러 개의 엔티티 타입 간의 제약조건을 명시하며, RULE을 이용하여 정의할 수 있다. RULE은 실행문과 그 결과에 의해 적합성을 결정하는 WHERE절

로 이루어 진다.

사. 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  EXISTS
           (husband)) XOR
           NOT EXISTS(maiden_name);
END_ENTITY;
```

위 예에서 person과 female의 2가지 엔터티 타입을 정의하였다. 엔티티 person은 first_name, last_name, birth-date, children을 명시 속성(explicit attribute)로 가지고 있다. 속성 age는 birth_date에서 유도된 것이다. 함수 years()는 여기에서 정의는 생략되었지만, 주어진 birth_date로부터에서 age를 계산한다. 엔터티 female은 person의 서브타입이다. 서브타입은 슈퍼타입의 모든 속성을 상속받는다. Female의 제약조건 W1은 female이 maden-name을 가지고 있다면 husband가 있어야 한다는 룰을 명시한 것이다.

5.2 표준 데이타 인터페이스(SDAI)

SDAI(STEP Data Access Interface)는 응용프로그램의 STEP 화일 또는 DB 스템과의 접속 방법에 관한 표준이다. 이러한 표준을 만듦으로서 응용프로그램 개발자나 DB 개발자는 서로 독립적으로 프로그램을 개발하여 판매할 수 있다. 또한 복수의 응용프로그램들이 복수의 DB 연결하여 데이타를 공유할 수 있다. 이때 SDAI는 복수의 응용프로그램과 복수의 DB 연결하는 가교 역할을 함으로서 개방형 CIM을 실현한다.

SDAI는 EXPRESS에 의하여 정의된 정보모델을 논리적 스키마로 이용하여 화일 내용을 해석하거나 DB 스키마로 전환하여 DB 시스템을 이용한다. 또한 응용프로그램과의 접속을 위하여 C, C++, 그리고 Fortran으로 정의된 프로그래밍 언어 바인딩을 정하였고, 각 바인딩은 early binding과 late binding을 포함하고 있다.

5.2.1 SDAI의 기능

데이타 저장 시스템(화일 시스템 또는 DB 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

5.2.2 프로그래밍 언어 바인딩(Language Bindings)

이 바인딩은 SDAI의 구현에 필요한 함수 또는 클래스 구조를 정의한다. 이러한 바인딩을 구현하기 위해 EXPRESS 스키마에 종속된 부분(early binding)과 독립된 부분(late binding)이 필요하다.

(1) Early Binding : 스키마에 정의된 엔터티 타입, 속성, 계층구조, 룰 등 모든 정보를 내부 구조에 맞게 전환한다. C++로 구현 할 때, 엔티티는 클래스로, 속성값(Attributes)은 프로텍트 된 변수로 전환된다.
(2) Late Binding : 특정 스키마에 종속 되지 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.

5.3 객체지향 DB를 이용한 SDAI 구현

본 절에서는 OODBMS인 Objectivity/DB를 이용하여 구현한 SDAI를 설명하고, 관계형 DBMS인 ORACLE을 이용한 SDAI 구현과의 차이점을 비교한다.

5.3.1 객체지향 DB를 이용한 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. 엔터티 타입은 하나의 영속 클래스

에 대응 되며 각 엔터티 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 전환 되었을 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 엔터티 클래스의 데이타 멤버로 전환되어 객체 핸들이나 멤버 함수를 통하여 접근 한다. Enumeration의 경우는 Objectivity/DB에서 enum 타입으로 변환되며, 그 구성 요소들의 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 11과 같다. 여기서 SDAI의 Early Binding이 Objectivity/DB의 논리적 구조(Logical Scheme)를 정의한다. 이것은 Objectivity/DB의 DDL이 C++와 같은 신택스를 사용하기 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 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

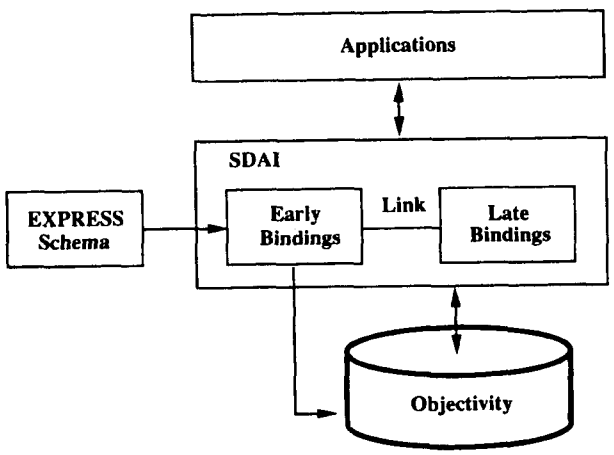


그림 11. Objectivity/DB를 이용한 SDAI 구조

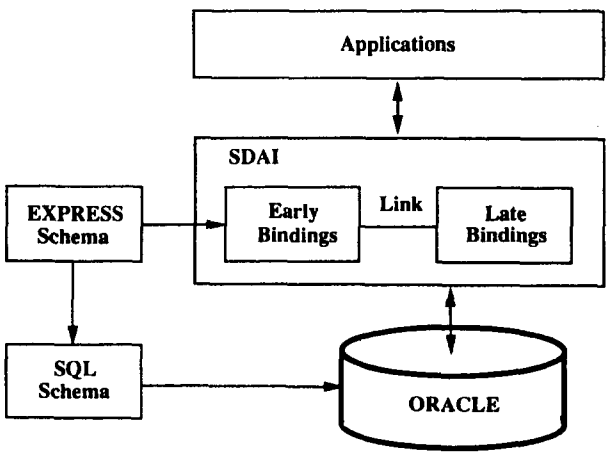


그림 12. ORACLE을 이용한 SDAI 구조

데이타베이스 시스템인 ORACLE을 사용할 때는 다르게 되는데 그 이유는 ORACLE의 스키마 정의를 위하여 SQL 프로그램이 필요하기 때문이다. ORACLE을 이용한 SDAI의 구조는 그림 12와 같다.

앞에서 예로 들은 엔터티 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 VALUES (
    'PERSON$E1', 'NAME', 'STRING', 'NULL', 0, 0, 0,
1);
INSERT INTO SDAI$ATTRIBUTES VALUES (
    'PERSONE1', 'AGE', 'STRING', 'NULL', 0, 0, 0, 2);
```

위 SQL 프로그램에서 엔터티의 이름을 변경한 이유는 ORACLE에서 테이블의 이름이 30자로 제한되어 있기 때문에 이를 극복하기 위해서이다. 이 예에는 들어 있지 않지만, 엔터티의 계층 관계 때문에 상속되는 속성들도 모두 엔터티의 정의에 포함 되어야 한다. 이는 객체 지향 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.

6. 결 론

본 고에서는 CIM SW 개발시 고려하여야할 DB 요구사항과, 객체지향 SW 및 DB 기술의 발전 추세를 기술하였으며, 객체지향 DB를 FMS 운용 관리 소프트웨어 개발과 ISO STEP표준에 의거한 통합 제품 DB 구축에 적용한 사례를 기술하였다. 이러한 CIM DB 기술, 특히 객체지향 DB 기술에 기반한 CIM 기술은 최근 크라이언트/서버 분산 컴퓨터 환경을 전제로 급속히 확산되고 있는 WWW(World-Wide Web)과도 좋은 조화를 이루어 보편화될 여지가 클 것으로 기대된다. OODBMS 벤더들간에 제정된 ODMG-93 표준은 이러한 추세를 더욱 가속화 시킬 것으로 기대되는데, 이런 면에서 현장의 CIM 소프트웨어 개발자를 위한 객체지향 프로그래밍 및 OODB에 대한 체계적인 실습 교육이 국내에서 시급하다.

※ 본 논문의 연구는 서울대학교 발전기금 일반학술연구비(92년)와 한국전력공사의 전력기술기조 연구비(93년) 지원을 받아 수행되었음.

참 고 문 헌

[1] W. W. Luggen, Flexible Manufacturing Cells and Systems, Prentice-Hall, 1991.

[2] Y.V. Ramana Reddy et al, "Computer Support for Concurrent Engineering," IEEE Computer, January, 1991.

[3] M. R. Cutkosky et al, "PACT : An Experiment in Integrating Concurrent Engineering Systems," IEEE Computer, January 1991.

[4] Navy CALS Home Page, http :

[5] E. F. Codd, "A Relational Model for Large Shared Databanks," CACM, vol. 13, Jun. 1970.

[6] F. Bancilhon, "Object-Oriented Database Systems," in Proc. ACM SIGACT SIGMOD Symp. Principles of Database Systems, Mar. 1988.

[7] R. G. G. Cattell, ed., The Object Database Standard : ODMG-93, Morgan Kaufmann, 1994.

[8] G. Booch, Object-Oriented Analysis and Design with Applications, Second Ed., Benjamin Cummings, 1994.

[9] 박장호, 차상균, "분산 FMS의 통합제어를 위한 객체지향 데이타베이스," 한국통신학회 논문지, 제19권, 제10호, 1994년 10월.

[10] S. K. Cha, "Object-Oriented Integration of Distributed Flexible Manufacturing Systems," Proc. of 3rd Int'l Conference onComputer-Integrated Manufacturing, pp 737-746, Singapore, July, 1995.

[11] 권혁일, "FMS 모니터링을 위한 객체 지향 그래픽 UIMS의 구현," 석사 학위논문, 서울대학교, 1994년 2월.

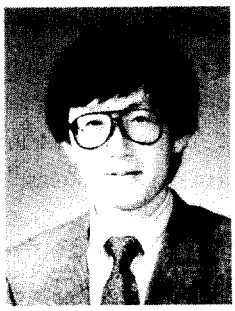
[12] Objectivity, Inc., Objectivity/DB C++ Interface Guide, ver. 3.5, 1995.

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

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

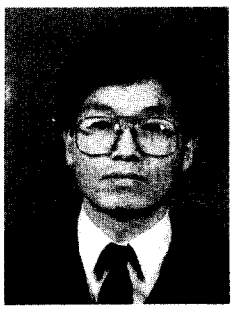
[15] ISO, "EXPRESS Language Reference Manual," TC184/SC4/WG5 Document Number 151, August 1992.

저 자 소 개



차 상 균

1980 서울대학교 공과대학 정기공학과 학사
1982 서울대학교 공과대학 제어계측공학과 석사
1991 미국 Stanford대학교 전기공학과(컴퓨터시스템)박사
Dacom, Texas Instruments, IBM Palo Alto, Hewlett-Packard Lab 근무
1992~ 현재 서울대 제어계측공학과 조교수
TEL/(02)880-7319 FAX/885-6620
(151-742) 서울시 관악구 신림동 산 56-1



유 상 봉

1987 서울대학교 공과대학 제어계측공학과 학사
1986 미국 Arizona대학교 전기 및 컴퓨터공학과 석사
1989 미국 AT & T Bell Lab, Holmdel NJ, 연구원
1990 미국 Purdue대학교 전기 및 컴퓨터공학과 박사
1990~1992 삼성전자 컴퓨터부문 선임연구원
1992~ 현재 인하대학교 자동화공학과 조교수
TEL/032-860-7386 FAX/032-863-4386
(402-751) 인천광역시 남구 용현동 253