객체 지향 언어를 이용한 개념적 데이타 모델 구현에 관한 연구

서울대학교 경영학과 노 상규

University of Minnesota Salvatore T. March

Abstract

Lack of support for Entity-Relationship (ER) semantics in object-oriented programming languages (OOPL) remains a key roadblock to the effective object-oriented systems development. We extend the object-oriented programming language Smalltalk with Entity-Relationship semantics including entity integrity constraints, cardinality constraints, and set-oriented data access. The meta-classes: PermanentObject, EntityObject, Relationship, and JoinClass provide the necessary structures and operations to support Entity-Relationship semantics. These meta-classes greatly reduce efforts to implement an Entity-Relationship model into an object system.

I. 연구의 배경 및 필요성

객체 지향 시스템 개발은 기업에서 큰 인기를 얻고 있다[Guttman and Matthews, 1995]. 객체모델은 시스템에 대한 일관된 표현이 가능하기 때문에 전통적인 데이타/프로세스 모델보다 개발자가 이해하기 쉬우며 객체 지향 개발은 개발자가 설계와 구현된 프로그램의 각 구성요소를 재활용할 수 있기 때문에 전통적인 개발 방법보다 더 효율적이라고 한다[Yourdon, 1994; Rumbaugh, et. al., 1991].

그러나 실제로 개념적 객체 모델을 정보 시스템으로 구현하는데는 많은 작업이 필요하다. 이 문제는 대부분 객체 지향 언어가 개념적 모델의 의미를 충분히 지원하지 않는 데서 기인한다[Ling and Teo, 1993; Bertino, et. al., Kilian, 1991]. 객체지향시스템에서 엔터티(entity)는 보통 데이타 또는 연산을 공유하지 않는 클래스(class)로 정의되며 관계(relationship)는 전형적으로 포인터를 이용하여 표현된다[Narasimhan, et. al., 1994]. 이런 경우, 몇 가지의 불이익이 존재한다. 첫째로, 메타데이타(metadata)를 질의할 수 없다. 즉 사용자가 기본적인 객체의 구조를 이해하는 것이 어렵다. 또 제약조건을 집행하는데는 메타데이타가 필요하기 때문에 제약조건을 집행하는 메소드(method)를 구현하기가 쉽지 않다. 둘째로, 각각의 엔터티 클래스에 참조무결성과 같은 기본적인 엔터티/관계 제약 조건을 집행하는 메소드(method)들이 구현되어야만 한다. 셋째로, 관계형 질의어에서 표현하기 쉬운 질의(query)가 객체지향언어에서는 각 객체 수준에서의 검색을 요구하기 때문에 지극히 표현하기 어렵다. 이런 문제는 복수의 엔터티를 포함하는 질의(예를 들면, 조인(Join))에서 발생한다.

본 연구에서는 개념적 데이타 모델 구현을 지원하기 위한 메타클래스(metaclass)들(PermanentObject, EntityObject, Relationship, JoinClass)을 제안하고자 한다. 이러한 메타클래스들은 개념적 데이타 모델의 구현과 객체 지향 시스템에서의 질의 작성을 수월하게 할 것이다. 본 논문은 다음과 같이 구성되어 있다. 제 2장에서는 개념 데이타 모델의 기본적 개념에 대해서 서술하고, 객체 지향 시스템에서 이를 지원하기 위한 이전의 연구를 검토할 것이다. 제 3장에서는 개념적 데이타 모델을 구현하는 데 필수적인 메타클래스의 구조와 연산에 대해 설명하고 제 4장에서는 간단한 적용 사례를 소개할 것이다. 마지막 장은 본 연구의 한계와 향후 연구 방향에 대해 논할 것이다.

II. 개념적 데이타 모델의 기본적 개념

개념적 데이타 모델을 지원하는 시스템은 엔터티(entity)와 그 속성(attribute), 관계(relationship), 식별자(identifier), 서브타입/수퍼타입(subtype/supertype) 구조, 제약조건(constraints) 등을 정의하고 유지하는 메카니즘을 가져야한다[Elmasri and Navathe, 1994; Teorey, 1994]. 이러한 시스템은 데이타에 대한 검색뿐만 아니라 정의(즉 메타데이타)에 대한 검색까지도 지원해야 한다. 엔터티 인스턴스(entity-instance)는 자신의 각각의 속성 값과 관계에 대해 대답할 수 있어야한다. 엔터티(클래스)는 선택되어진 인스턴스들을 대답할 수 있어야 할뿐만 아니라 자신의 속성과 자신이 참여하고 있는 관계의 명칭 또한 대답할 수 있어야한다. 더 나아가 시스템이 효과적이기 위해서는 관계형 조인 연산에 일치하는 집합연산이 지원되어야 한다[Markowitz and Shoshani, 1993; Norrie,. 1993].

Smalltalk [Goldberg and Robson, 1989]이나 C++와 같은 객체 지향 프로그래밍 언어는 클래스 개념을 통해 엔터티와 속성에 대한 지원을 한다. 하나의 엔터티는 그 속성들을 인스턴스 변수 (instance variable)로 가지는 클래스로 표현될 수 있다. 그러나 개념적 데이타모델의 견지에서의 식별자에 대한 직접적인 지원은 제공되지 않는다 (각 인스턴스에는 유일한 객체 식별자(object identifier, OID)가 주어진다. 그러나 어떤 변수의 집합이 클래스의 인스턴스들 내에서 유일한 값을 가질 수 있도록 규정하는 방법은 없다.).

관계(Relationship)는 내장 객체(embedded objects)를 이용하여 지원된다. 관계를 표현하기 위해 인스턴스 변수는 관련된 객체(들의 집합)를 보유할 수 있다. 즉 인스턴스변수는 그들의 객체 식별자(OID)를 값으로 가질 수 있다. 이와 같은 인스턴스 변수를 사용하여 관련된 객체를 검색할 수 있고, 그들의 속성을 검색할 수 있다. 따라서 관계에 대한 인스턴스 수준에서의 검색은 가능하게 된다. 그러나 관계형 조인 연산에 대응하는 집합연산의 지원은 가능하지 않다.

제약 조건의 집행은 각 클래스에 의해 이루어진다. 제약 조건의 정의와 집행방법은 엔터티를 표현하는 클래스의 메소드에 숨겨져 있다. 각 클래스는 두 개의 인스턴스가 동일한 식별자 값을 가질 수 없도록 엔터티 무결성(entity integrity)을 점검하는 메소드를 가져야만 한다. 각 클래스는 그 클래스에 정의된 각 관계에 대한 관계 제약 조건을 충족시킬 메소드를 가져야만 한다. 관계는 하나 또는 두 개의 클래스, 즉 한 방향 또는 양쪽 방향으로 표현될 수 있다. 따라서 설계자는 어느 클래스에 관계가 표현되어져야 하는가를 결정하고, 제약조건을 집행하는 메소드를 개발하여야만 한다.

본 연구에서는 개념적 데이타 모델의 구현을 지원하기 위한 기본적 연산은

모든 엔터티와 관계에서 유사하다는 점에 착안하여, 이를 지원하는 클래스들을 개발하고, 엔터티들이 이 클래스들로부터 개념적 데이타 모델의 연산을 상속토록 하였다. 이러한 방법은 메소드들을 재활용함으로써 개발자의 노력을 감소시킬 수 있을 뿐만 아니라 사용자의 필요에 따라 검색할 수 있는 메타데이타의 유지가 가능하다.

III. 개념적 데이타 모델 구현을 위한 클래스 구조

개념적 데이타 모델에 대한 지원은 메타데이타를 저장하고 위에서 언급된 기능을 제공하는 포괄적인 클래스에 대한 정의로부터 시작된다[Elmasri, et. al., 1993]. 본 연구에서는 PermanentObject, EntityObject, Relationship, JoinClass라는 네 가지 클래스를 제안한다. 이러한 클래스는 그림 1에서 보여지는 바와 같이 객체 계층 구조를 이루고 있다.

PermanentObject는 기본적인 데이타 저장과 검색 능력을 제공하며(즉 *Instances, FindInstance:, AddInstance:, DeleteInstance:*) 이는 EntityObject와 Relationship이 상속받는다. EntityObject는 기본적인 검색 및 유지기능과 엔터티 무결성 제약조건을 집행하는 기능을 제공한다. 엔터티는 EntityObject의 서브클래스로 정의된다. Relationship은 모든 관계데이타를 관리하고 제약조건을 집행하는 역할을 한다. 관계는 Relationship의 인스턴스로 정의된다. 개념적 모델에 대한 질의는 SQL과 유사한 문법으로 작성된다. 질의는 서로 연결된 JoinClass의 인스턴스들로 표현된다. 메타클래스의 주요 메소드들에 대해 간략히 설명하면 다음과 같다(부록 1에 메소드가 정리되어 있다).

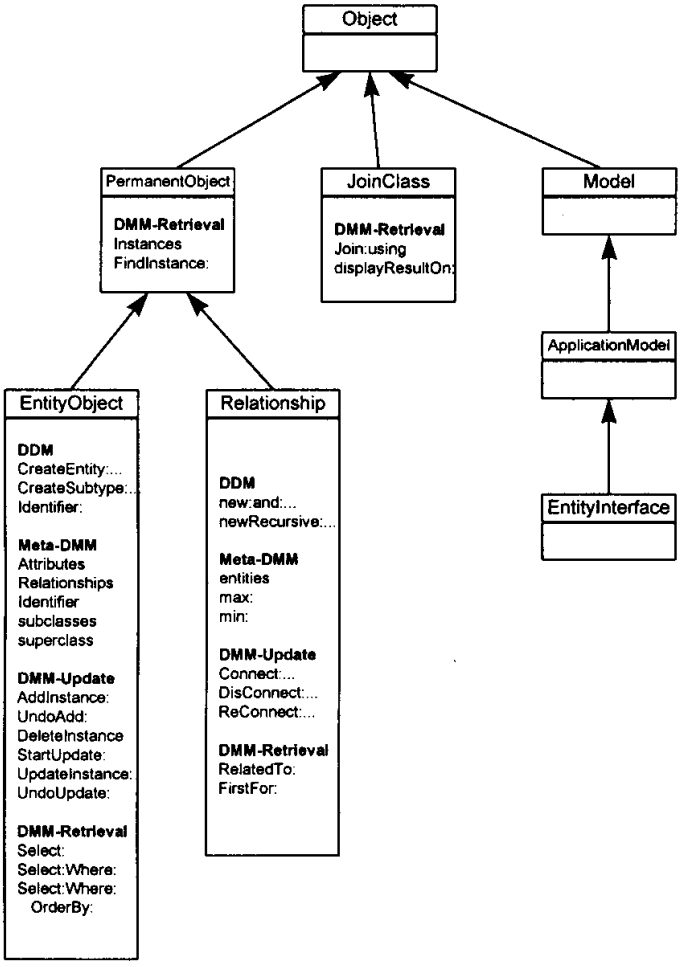

그림 1. 메타클래스 계층구조

엔터티는 EntityObject에게 자신의 하위 클래스로 엔터티를 생성하라는 메시지인 *CreateEntity:* 를 보냄으로써 정의된다. 데이타와 메타데이타를 검색하고 유지하는 메소드는 이것으로부터 상속된다. 속성은 CreateEntity의 매개변수이며, 인스턴스 변수로 정의된다. 각 속성의 검색(accessor)과 할당(assignment) 메소드는 자동으로 생성된다. 표준적인 Smalltalk의 관습에 따라, 양자는 각 속

성에 따라 명명되어진다. 할당 메소드 명칭 뒤에는 콜론(:)과 변수에 할당된 값을 가진 매개변수가 온다.

각 엔터티의 식별자는 메소드의 리스트를 매개변수로서 가지는 *Identifier:* 라는 메시지를 보냄으로써 정의된다. EntityObject로부터 상속된 이 메소드는 수신자 측에 인스턴스 메소드 *id*를 생성한다. *id* 메시지가 어떤 엔터티 인스턴스에 보내지면, 그것은 식별자의 값을 대답한다.

서브 타입은 EntityObject로부터 상속된 *CreateSubtype:* 라는 메시지를 엔터티에게 보냄으로써 생성된다. 서브 타입을 지원하기 위해서 Smalltalk에서 구현된 계층구조가 사용되는데, 이러한 계층구조는 각 서브 타입이 수퍼 타입으로부터 모든 변수와 메소드를 상속받는 것을 가능하게 한다. 그러나 이것은 서브 타입 / 수퍼 타입 구조를 배타적인 서브 타입을 가지는 계층 구조로 제한한다.

관계는 *new:* 라는 메시지를 Relationship 클래스에 보냄으로써 정의된다. Relationship은 자신의 인스턴스를 생성하고, 관계의 정의를 인스턴스 변수에 저장한다. 이 변수들은 관계의 이름, 연관된 엔터티, 최대/최소값을 포함한다. 또한 관계인스턴스도 인스턴스 변수에 저장된다. Relationship은 데이타 저장과 검색능력을 상속할 수 있도록 PermanentObject의 서브 클래스로서 정의된다. 관계가 정의되면, 각 엔터티에 관계를 검색하고 유지하는 인스턴스 메소드가 생성된다.

각 엔터티는 EntityObject로부터 데이타 검색 능력을 상속받는다. 이는 SQL과 비슷한 메소드 *Select:Where:OrderBy:*를 포함하고 있다. 이 메시지의 매개 변수들은 단일 엔터티의 인스턴스들에 대한 프로젝션(projection), 실렉션

(selection), 순서(ordering) 조건을 정의한다. 이 메시지의 수신자인 엔터티는 단일 엔터티 질의의 정의를 저장하는 JoinClass의 인스턴스를 대답한다. 그 질의는 *displayResultOn:* 이라는 메시지를 보냄으로써 실행된다.

복수 엔터티에 대한 질의 또한 JoinClass의 *Join:using:*이라는 메시지에 의해 지원된다. 첫 번째 매개 변수로 다른 JoinClass 인스턴스, 두 번째 매개 변수로 관계를 포함한 메시지가 JoinClass 인스턴스에 보내지면, 이 메시지는 조인 연산의 정의를 포함하는 새로운 JoinClass 인스턴스를 생성하고, JoinClass인스턴스들로 질의그래프(query graph)로 구성한다. *Join:using:* 이라는 메시지를 이용하여 사이클과 recursion을 포함하는 다양한 방식으로 복수의 엔터티로 구성된 질의를 표현할 수 있다. 질의는 *displayResultOn:* 이라는 메시지를 JoinClass 인스턴스에게 보냄으로써 실행된다.

IV. 적용 사례

이 장에서는 위의 메타클래스를 간단한 개념적 데이터 모델 구현에 적용한 사례를 소개하고자 한다. 그림 2의 개념적 데이타 모델을 구현하기 위해 필요한 메시지는 아래와 같다. 다음은 Smalltalk에서 실행 가능한 스크립트이다. (Demo는 새로운 클래스들이 생성되어지는 Smalltalk의 범주(category)이다. [VisualWorks, 1994])

```
EntityObject CreateEntity: #Employee attributes: 'eno ename ssn eaddress
wageRate' under: 'Demo'.
```

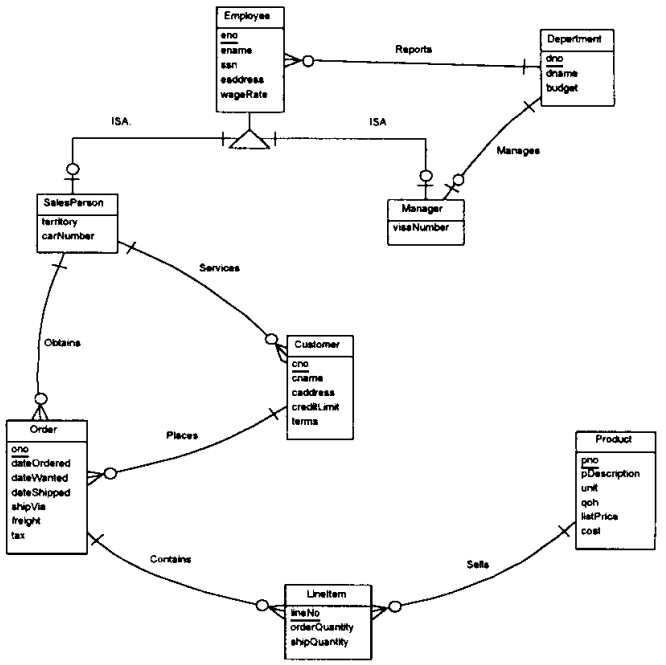

그림 2. 개념적 데이터 모델 사례

EntityObject CreateEntity: #Department attributes: 'dno dname budget' under: 'Demo'.

EntityObject CreateEntity: #Customer attributes: 'cno cname caddress creditLimit terms' under: 'Demo'.

EntityObject CreateEntity: #Order attributes: 'ono dateOrdered dateWanted dateShipped shipVia freight tax' under: 'Demo'.

EntityObject CreateEntity: #LineItem attributes: 'lineNo orderQuantity shipQuantity' under: 'Demo'.

EntityObject CreateEntity: #Product attributes: 'pno pDescription unit qoh listPrice cost' under: 'Demo'.

서브 타입은 수퍼타입 엔터티에 적절한 메시지를 보냄으로써 생성된다.

```
Employee CreateSubtype: #SalesPerson attributes: 'territory carNumber'.
Employee CreateSubtype: #Manager attributes: 'visaNumber'.
```

각 속성에 대해 인스턴스 변수와 인스턴스 메소드가 생성된다. Manager와 SalesPerson은 Employee의 서브 클래스로 정의된다. Manager와 SalesPerson은 Employee로부터 모든 속성과 메소드를 상속받는다.

관계는 Relationship에 적절한 메시지를 보냄으로써 생성된다(Many는 큰 정수 값의 global 변수로 선언된다.).

```
Relationship new: Employee and: Department withMin: 0 andMin: 1 withMax: Many andMax: 1 named: 'Reports'.
Relationship new: Manager and: Department withMin: 0 andMin: 1 withMax: 1 andMax: 1 named: 'Manages'.
Relationship new: SalesPerson and: Customer withMin: 1 andMin: 0 withMax: 1 andMax: Many named: 'Services'.
Relationship new: SalesPerson and: Order withMin: 1 andMin: 0 withMax: 1 andMax: Many named: 'Obtains'.
Relationship new: Customer and: Order withMin: 1 andMin: 0 withMax: 1 andMax: Many named: 'Places'.
Relationship new: Order and: LineItem withMin: 1 andMin: 0 withMax: 1 andMax: Many named: 'Contains'.
Relationship new: LineItem and: Product withMin: 0 andMin: 1 withMax: Many andMax: 1 named: 'Sells'.
```

마지막으로 각 엔터티에 대해 식별자가 정의된다.

Employee Identifier: 'eno'.

Department Identifier: 'dno'.

Customer Identifier: 'cno'.

Order Identifier: 'ono'.

LineItem Identifier: 'Contains lineNo'.

Product Identifier: 'pno'

Employee, Department, Customer, Order, Product는 독립적인 엔터티이며, 그들의 식별자는 단일한 속성이다. LineItem은 Order에 종속되어 있다. 따라서 Order와 LineItem사이의 Contains관계를 대답하기 위해 생성된 메소드인 Contains가 식별자에 포함되어있다. Manager와 SalesPerson은 Employee의 서브 타입이며, 식별자로서 수퍼 타입으로부터 'eno'를 상속받는다. 위의 스크립트는 개념적 데이터 모델 정의 화면(그림 3)을 이용하여 실행될 수도 있다.

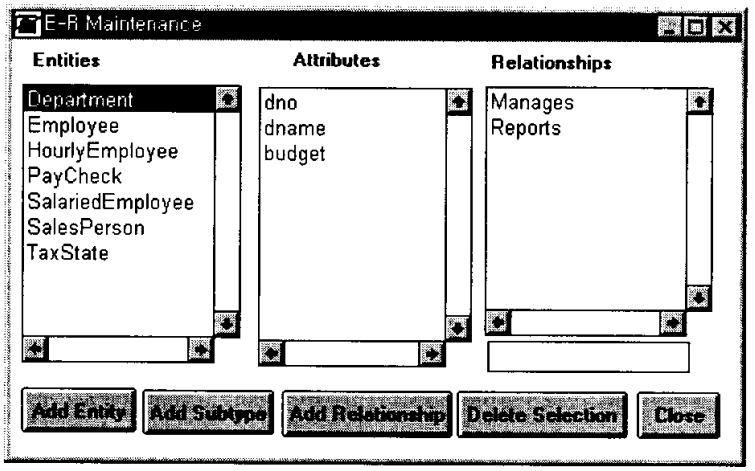

그림 3. 개념적 데이터 모델 정의 화면

메타데이타는 그림 1의 Meta-DML 메소드를 이용하여 질의될 수 있다. 각 엔터티는 자신의 Attributes, Relationships, subclasses, superclass를 대답할 수 있다. 각 관계는 자신의 entities, min:, max: 를 대답할 수 있다.

데이타는 다음과 같은 방식으로 갱신된다. 엔터티의 새로운 인스턴스는 EntityObject로부터 상속된 *new* 메시지를 엔터티에 보냄으로써 생성된다. 각 엔터티 인스턴스의 속성들의 값은 생성된 할당 메소드를 이용하여 갱신된다. 관계는 생성된 관계 메소드를 이용하여 유지된다. 예를 들면, Employee와 Department를 연결하는 Reports라는 관계가 정의될 때, 인스턴스 메소드인 *Reports:* 는 Employee와 Department 양자에 생성되었다. 이는 수신자측 인스턴스와 매개변수인 다른 엔터티의 인스턴스를 연결해준다. 이 때 Relationship의 메소드인 *Connect:and:using:*을 이용한다.

새로운 인스턴스의 속성과 관계의 모든 값들이 입력되었을 때, 그 인스턴스는 자신을 매개변수로 Addinstance: 메시지를 자신이 속한 엔터티에게 보냄으로써 지속성을 가지게 된다. 이 메소드는 EntityObject로부터 상속된 것이고, 모든 무결성 제약 조건을 확인하고 집행한다. 따라서 한 명의 종업원은 다음과 같이 추가될 수 있다.

```
e := Employee new initialize.
(e getValues isNil) ifFalse: [Employee AddInstance: e].
```

여기서 *initialize* 는 employee의 모든 속성과 관계를 그들의 기본 값으로 초기화하며, *getValues*는 데이타 입력 화면을 보여주고, 적절한 속성과 관계 값을 얻기 위해 사용자와 상호 작용한다. (이 때 사용자가 데이타 입력을 취소하

면 nil을 대답해야한다) 이 메소드들은 개발자에 의해서 작성되어야 한다.

데이타는 다음과 같은 방식으로 검색되어진다. 개별적인 인스턴스는 매개변수로 식별자를 가지는 *FindInstance:* 메시지를 엔터티에게 보냄으로써 검색될 수 있다. 검색되면, 인스턴스는 메시지에 따라 자신의 속성과 관계를 대답할 수 있다. 예를 들면, 종업원번호가'1000'인 종업원의 부서 번호를 검색하려 할 때, 다음과 같이 사용할 수 있다.

```
(Employee FindInstance: '1000') Reports dno.
```

매개변수로 '1000'이라는 값을 가지는 *FindInstance:* 메시지가 보내어졌을 때, Employee는 해당하는 employee 인스턴스를 대답한다. 또한 *Reports* 메시지를 받으면, 그 employee 인스턴스는 그가 속한 department 인스턴스를 대답한다. *dno*라는 메시지를 받으면 그 department 인스턴스는 자신의 dno를 대답한다.

간단한 질의는 *Select:Where:OrderBy:* 라는 메시지를 적절한 엔터티에 보냄으로써 정의될 수 있다. 예를 들면, 식별자순으로 임금이 $100,000 이상인 모든 종업원의 번호와 이름을 보여주는 질의를 다음과 같이 수행할 수 있다.

```
(Employee Select: 'eno ename' Where: [:e | e wageRate > 100000]
OrderBy: [:e1 :e2 | e1 id <= e2 id]) displayResultOn: Screen
```

조인 질의는 *Join:using:* 라는 메시지를 적절한 JoinClass 인스턴스에 보냄으로써 정의된다. Employee와 Department를 Report관계를 이용하여 조인하기

위해 다음과 같이 쓸 수 있다.

(Employee Select: 'eno ename') Join: (Department Select: 'dno dname') using: 'Reports'.

서브 타입도 조인될 수 있다. 또한 *Join:using:* 메시지 송신의 결과가 *Join:using:* 메시지를 인식하는 JoinClass 인스턴스이기 때문에, Join은 순차적으로 수행될 수 있다. 판매원과 그들의 부서, 그들이 서비스하는 고객을 얻기 위해, SalesPeople과 Department를 Reports관계로 결합하고, 그 결과에 Customer를 Services관계로 결합한다.

((SalesPerson Select: 'eno ename') Join: (Department Select: 'dno dname') using: 'Reports') Join: (Customer Select: 'cno cname') using: 'Services'.

V. 한계점과 향후 연구의 방향

본 연구는 객체 지향 시스템에서 개념적 데이타 모델의 지원 가능성을 예시하고 있다. 네 개의 클래스-PermanentObject, EntityObject, Relationship, JoinClass-는 개념적 데이타 모델의 정의, 유지, 검색에 필수적인 구조와 연산을 제공한다.

그러나 현재의 시스템은 여러 면에서 한계점을 가지고 있으며 이는 본 연구의 향후 연구 과제이다. 먼저 어떤 제약조건이 갱신연산에 의해 위반되면, 그 연산은 취소된다. 다른 제약 조건 위반 연산에 대한 규정이 향후 연구의 과제가 될 것이다. Cascade Delete와 같은 연산은 명백하게 지원되어야 한다. 그러나 사용자가 규정한 추가적인 연산 또한 지원되어야만 한다. 예를 들면, 상호 의존적인 제약 조건(예를 들면, 0이상인 최소값 한계를 가지는 관계들에 대한 제약 조건)의 규정과 실행이 향후 연구의 방향이 될 수 있다. 둘째로, 상호 배타적인 서브 타입을 가지는 계층 구조만이 지원된다는 점이다. 모든 서브 타입 제약 조건을 지원하는 중복적인 서브 타입을 규정하는 것이 이후 연구의 방향이 될 것이다. 마지막으로 현재의 연구는 Aggregation 관계를 지원하지는 않는다는 점이다. Aggregation이란 다른 엔터티의 '부분'을 구성하는 엔터티들의 집합이다.

参考文獻

Bertino, E., Negri, M., Pelagatti, G., and Sbattella, L., "Object-Oriented Query Languages: The Notion and the Issues," IEEE Transactions on Knowledge and Data Engineering, vol 4, no 3, June 1992, pp. 223-237.

Elmasri, R., Larson, J., and Kouramajian, V., "Automatic Class and Method Generation for Object-Oriented Databases," Proceedings Third International Conference on Deductive and Object-Oriented Databases, December 6-8, 1993, Springer-Verlag, Berlin, 1993, pp. 395-414.

Elmasri, R. and Navathe, S. B., Fundamentals of Database Systems (2nd edition), Addison-Wesley/Benjamin/Cummings, Redwood City, CA, 1994.

Goldberg, A. and Robson, D., Smalltalk-80, Addison-Wesley, Reading, MA, 1989.

Guttman, M. and Matthews, J., The Object Technology Revolution, John Wiley & Sons, Inc., New York, 1995.

Kilian, M. F., "Bridging the Gap Between OO and ER," Proceedings of the 10th International Conference on Entity-Relationship Approach, University of Michigan Press, Ann Arbor, October 23-25, 1991, pp. 445-458.

Ling, T. W. and Teo, P. K., "Toward Resolving Inadequacies in Object-Oriented Data Models," Information and Software Technology, (35, 5), May 1993, pp. 267-276.

Markowitz, V. and Shoshani, A, "Object Queries Over Relational Databases: Language, Implementation, and Applications," Proceedings of the 9th International Conference on Data Engineering, IEEE Computer Society Press, Los Altimos, CA, 1993, pp. 71-80.

Narasimhan, B., Navathe, S. B., and Jayayaman, S., "On Mapping ER and Relational Models into OO Schemas," Proceedings of the 12th International Conference on Entity-Relationship Approach, Springer-Verlag, Berlin, 1994, pp. 402-413.

Norrie, M. C., "An Extended Entity-Relationship Approach to Data Management in Object-Oriented Systems," Proceedings of the 12th International Conference on Entity-Relationship Approach, Springer-Verlag, Berlin, 1993, pp. 390-401.

Rumbaugh, J., Blaha, M., Premerlani, W., and Lorensen, W., Object-Oriented Modeling and Design, Prentice Hall, Englewood Cliffs, NJ, 1991.

Rumbaugh, J., "Relations as Semantic Constructs in an Object-Oriented Language," Proceedings OOPSLA Conference, October 4-8, 1987, pp. 466-481.

Teorey, T. J., Database Modeling and Design (2nd edition), Morgan Kaufmann, San Francisco, CA, 1994.

VisualWorks, User's Guide, ParcPlace Systems, Inc., 1994.

Yourdon, E., Object-Oriented Systems Design, Yourdon Press-Prentice Hall, Englewood Cliffs, NJ, 1994.

부록 1. 메타클래스 메소드

1. 데이타 정의 메소드 (Data Definition Methods)

CreateEntity: eName **attributes**: attrList **under**: appName

EntityObject에 이 메시지를 보내면 EntityObject의 서브클래스로 eName이 생성된다. 이 클래스는 attrList에 포함된 인스턴스 변수를 가지게 되며 appName라는 범주에 생성된다.

CreateSubtype: eName **attributes**: attrList

EntityObject의 서브클래스(즉 엔터티)에 이 메시지를 보내면 그 엔터티의 서브클래스로 eName라는 클래스가 생성된다.

new: e1 **and**: e2 **withMin**: min1 **andMin**: min2 **withMax**: max1 **andMax**: max2 **named**: rName

Relationship에 이 메시지를 보내면 e1과 e2 엔터티 사이에 rName이라는 관계를 생성한다. 이때 최대/최소값도 지정한다.

newRecursive: e **withMin**: min1 **andMin**: min2 **withMax**: max1 **andMax**: max2 **named**: rName1 **converselyNamed**: rName2

Relationship에 이 메시지를 보내면 recursive관계를 생성한다. 이 때는 관계의 방향마다 하나씩 두 개의 관계이름이 필요하다.

Identifier: idList

엔터티에 이 메시지를 보내면 idList에 포함된 속성과 관계로 식별자가 정의된다. 이 식별자는 엔터티 무결성 제약조건을 집행하는 데 사용된다.

Attributes

엔터티에 이 메시지를 보내면 그 엔터티의 모든 속성의 이름을 대답한다.

Relationships

엔터티에 이 메시지를 보내면 그 엔터티의 모든 관계의 집합을 대답한다.

Identifier

엔터티에 이 메시지를 보내면 그 엔터티의 식별자를 답한다.

subclasses

엔터티에 이 메시지를 보내면 그 엔터티의 서브타입을 답한다.

superclass

엔터티에 이 메시지를 보내면 그 엔터티의 수퍼타입을 답한다.

entities

관계(즉 Relationship의 인스턴스)에 이 메시지를 보내면 그 관계에 참여하는 2 엔터티를 답한다.

max: e

관계에 이 메시지를 보내면 엔터티 e의 인스턴스가 그 관계에 참여하는 다

른 엔터티의 한 인스턴스에 연관될 수 있는 최대값을 답한다.

min: e

관계에 이 메시지를 보내면 엔터티 e의 인스턴스가 그 관계에 참여하는 다른 엔터티의 한 인스턴스에 연관될 수 있는 최소값을 답한다.

2. 엔터티 검색 및 유지 메소드 (Entity Access and Maintenance Methods)

Instances

엔터티에 이 메시지를 보내면 그 엔터티와 서브타입의 모든 인스턴스를 답한다.

FindInstance: anID

엔터티에 이 메시지를 보내면 그 엔터티의 인스턴스 중 식별자 값이 anID인 인스턴스를 답한다.

AddInstance: anInstance

엔터티에 이 메시지를 보내면 엔터티와 관계제약조건을 만족시키는 경우에 한하여 anInstance를 저장한다.

UndoAdd: anInstance

엔터티에 이 메시지를 보내면 *anInstance*에 대한 add연산을 무효화 시킨다.

DeleteInstance: anInstance

엔터티에 이 메시지를 보내면 관계제약조건을 위반하지 않는 경우에 anInstance를 삭제한다.

StartUpdate: anInstance

엔터티에 이 메시지를 보내면 anInstance에는 lock이 걸리고 갱신연산의 취소에 대비하여 log를 시작한다.

UpdateInstance: anInstance

엔터티에 이 메시지를 보내면 엔터티와 관계제약조건을 만족시키는 경우에 한하여 anInstance를 영구히 갱신한다.

UndoUpdate: anInstance

엔터티에 이 메시지를 보내면 anInstance에 대한 갱신연산을 무효화한다.

3. 관계 검색 및 유지 메소드 (Relationship Access and Maintenance Methods)

Connect: i1 **and**: i2 **using**: rName

Relationship에 이 메시지를 보내면 관계제약조건을 만족시키는 경우에 한하여 인스턴스 i1과 i2사이에 rName관계를 성립시킨다.

DisConnect: i1 **and**: i2 **using**: rName

Relationship에 이 메시지를 보내면 관계제약조건을 위반하지 않는 경우에 한하여 인스턴스 i1과 i2사이의 rName관계를 제거한다.

ReConnect: i1 **from**: i2 **to**: i3 **using**: rName

Relationship에 이 메시지를 보내면 관계제약조건을 위반하지 않는 경우에 한하여 인스턴스 i1과 i2사이의 rName관계를 제거하고 i1과 i3사이의 관계를 성립시킨다.

RelatedTo: i1 **using**: rName

Relationship에 이 메시지를 보내면 인스턴스 i1과 rName관계로 연관된 인스턴스들을 답한다.

FirstFor: i1 **using**: rName

Relationship에 이 메시지를 보내면 인스턴스 i1과 rName관계로 연관된 인스턴스들 중 첫 번째 인스턴스를 답한다.