DB Technology
and Management

객체 중심 측면 모델의 형식론
(The Formalism of Object-Oriented EA Model)

오선영
충남전문대학

백두권
고려대학교

객체 중심 측면 모델의 형식론

(The Formalism of Object-Oriented EA Model)

*　　**

오 선 영, 백 두 권

*　충남전문대학 전자계산학과

**　고려대학교 전산과학과

요 약

기존의 데이타 모델 및 설계 방법론들은 실세계의 데이타 객체에 대해 고정된 한 측면의 모델 표현만을 허용하기 때문에 여러 측면으로 관측이 가능한 실세계 객체들의 표현에 어려움을 갖는다.

본 논문에서는 객체의 측면 개념을 제공하는 객체 중심 측면 모델(OOAM : Object-Oriented EA Model)의 기본 개념을 보여주고 OOAM에 의해 구축되는 데이타베이스 스키마의 시맨틱을 분석하고 서술하기 위해 OOAM을 형식적으로 정의하였다. 먼저 데이타베이스 설계 과정에서 필요한 공리들을 설정하고 OOAM을 intension과 extension으로 각각 정의한 후 그들 사이의 관계를 정의하였다.

1. 서론

데이타베이스의 개념적 스키마에서 시맨틱을 취하는 것은 데이타베이스 설계 과정에서 가장 중요한 행동이다[1,15]. 개념적 스키마 설계의 가장 중요한 점은 어떻게 정보들을 효율적으로 구분하여 주어진 개념적 모델로 변형시키는 가이다. 그러나 이것은 설계 단계에서 수집된 정보들이 모호하고 부정확해서 일관성있고 정확하게 이루어지지 않고 있다. 그리고 기존의 제안된 여러개의 모델들은 정형

화의 부족으로 개념적 스키마의 분석을 다소 번거롭게 만든다.

데이타베이스 설계의 정형적인 접근은 데이타베이스 이론 분야에서 많이 연구되고 있다. 관계 데이타 모델은 현재까지 광범위하게 연구됨과 동시에 많은 시제품으로 사용되어 이상적인 진보(이론에서 구현까지)를 해왔다[15]. 관계 데이타 모델 이론은 1970년에 처음으로 소개되어 SQL/DS와 INGRES 등에서 구현되었다. 그러나 관계 데이타베이스 스키마에 대한 이론(데이타 종속, 정규화 등)은 많은 연구가 되어 왔으나[12], 객체중심데이타 모델에 대한 이론적인 접근은 [5, 15]과 같이 소수에 불과하다.

관계 데이타베이스 모델과 달리 객체 지향 데이타베이스 모델은 이론적인 뒷받침 대신에 SMALLTALK, COMMON Objects, C++과 같은 프로그래밍 환경으로 개발되어왔다. 특히 대부분의 연구들이 객체지향 특성, 예를 들면 객체 식별, 복합 객체, 상속, 메소드 등에 대한 정형화를 프로그래밍 관점으로 보고 logic에 대해 중점을 두고 있다[3, 7, 8, 9, 11]. 이와 같은 접근은 프로그래밍 언어와 DB 모델 사이의 갭에서 발생하는 문제들 때문에 데이타베이스 설계자에게 많은 어려움을 준다. 그리고 대부분이 스키마의 구문면에 중점을 둔 반면에 개념적 스키마 설계 과정과 모델의 시맨틱을 등한시하고 있다. 또한 intensional 수준의 시맨틱과 extensional 수준의 시맨틱 사이의 적당한 구분이 없다.

객체 중심 측면 모델(OOAM)은 의미데이타 모델(semantic data model)인 MAO 모델[18]을 객체 중심 데이타 모델로 확장시킨 것으로 메소드, N-항 연관성 등 새로운 개념을 첨가하여 데이타베이스의 모델링 능력을 증가시킨 모델이다[16,17]. OOAM은 실세계를 관점에 따라 여러 측면으로 모델링하므로 모델링 접근을 용이하게 하며 보다 더 융통성 있는 모델링 방법을 제공한다.

본 논문에서는 OOAM의 시맨틱과 구문을 형식적으로 정의한다. 먼저 2장에서는 OOAM의 기본적인 개념을 간략히 서술하며 3장에서 설계 과정에서 필요한 공리를 정의한다. 이 공리들은 데이타베이스에 관련된 시맨틱 특성들을 명확히 나타내기 위해 선택됐다. 다음으로 4장에서는 데이타베이스의 intension의 정형적인 정의를 내려 OOAM 모델에 의해 구성된 개념적 스키마의 클래스들을 보여준다. 또한 OOAM 스키마에서 정보의 검색을 가능하게 해 주는 path 함수를 정의한다. 5장에서는 데이타베이스 extension을 정의하고 intension과 extension사이의 관계가 서술된다. 마지막으로 6장에서는 본 논문에 대한 결론과 추후 연구과제를 제시한다.

2. 객체 중심 측면 모델

OOAM은 여러 측면으로 관측이 가능한 실세계의 데이타 객체들을 직접 데이타베이스의 다중 측면 객체로 모델링할 수 있도록 한다. 이를 위해 OOAM의 스키마는 엔티티를 나타내는 객체 클래스와 각 엔티티를 이해하는 관점인 측면 클래스로 구성된다. 모든 엔티티는 객체로 표현되고 이러한 객체는 엔티티 타입에 해당하는 클래스에 속하며 속성과 메소드를 공유하게 된다. 객체에 대한 연산은 해당 객체의 메소드로 표현되고 연산의 실행은 이 객체에 메시지를 보냄으로써 처리된다.

2.1 클래스

OOAM에서는 클래스를 객체 클래스(object class)와 측면 클래스(aspect class)로 구분한다. 객체 클래스는 모델링 과정에서 표현하고자 하는 실세계의 물리적 대상체들이나 개념, 사건, 혹은 추상화된 사고 등을 나타낸다.

측면 클래스는 객체 클래스에 대한 한 이해관점을 나타낸다. 한 객체 클래스는 여러 측면을 가질 수 있으며, 측면을 통해 다른 여러 객체 클래스들과 연계된다. 즉, 객체 클래스들은 측면개념을 통해 상호간의 암시적, 혹은 명시적 관계성을 갖는다. 전통적인 객체 중심 시스템에서 측면은 추상화된 관계성으로 볼 수 있다. OOAM에서 측면 클래스는 P-타입, G-타입, A-타입의 세가지로 나누어진다. OOAM의 기본 개념은 그림 1과 같다.

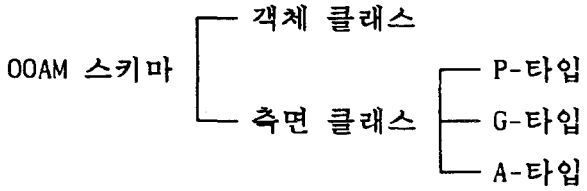

그림 1. OOAM의 기본개념

OOAM의 스키마는 OOAM 다이어그램을 이용하여 그림으로 표현할 수 있다. OOAM 다이어그램은 객체 클래스을 나타내는 객체 클래스 노드와 객체 클래스 노드에 대한 이해 관점을 나타내는 측면 클래스, 객체 클래스 노드와 측면 클래스들을 연결하는 링크로 구성되는 그래픽 구조의 다이어그램이다.

클래스에는 유사한 객체들의 공통되는 성질, 즉 이들의 스키마 정보와 이 객체들에 적용 가능한 연산에 대한 정보가 포함되며 인스턴스는 해당 클래스의 스키마 정보에 따라 실제값을 갖는 객체이다.

MAO 모델과 달리 OOAM은 연관성 타입을 두지 않고 연관성 추상화 개념을 A-타입에 의해 나타내게 하였으며 S-타입과 P-타입을 묶어 G-타입으로 나타내었다.

2.2 관계성

측면 클래스는 객체 클래스에 대한 한 이해관점을 나타낸다. 즉, 객체 클래스들은 측면 클래스를 통해 상호간의 암시적 혹은 명시적 관계성을 갖는다. OOAM에서 지원하는 추상화 관계성은 측면 클래스로 IS_A 관계를 나타내는 G-타입과 PART_OF 관계를 나타내는 P-타입, 연관성 관계를 나타내는 A-타입이 있다.

(1) G-타입 측면

G-타입 측면은 객체 클래스들의 유사한 성질들을 공유하기 위한 추상화로 일반 객체 클래스와 더욱 세분된 클래스 사이의 IS-A 또는 A-PART-OF 관계성을 나타낸다. IS-A 관계의 하위 클래스는 상위 클래스들로부터 특성을 계승받는다[14].

G-타입 측면은 2가지의 종류로 분류될 수 있는데[2] 하나는 배타적 일반화(disjoint generalization)로 하위 객체 클래스들이 상위 객체 클래스의 배타적 부분집합으로 이루어진 것이다. 즉, 객체타입의 객체들이 다수의 상호배재적인 객체타입들로 분할되어 이해될 수 있다[6]. 다른 하나는 하위객체 클래스들이 상위 객체 클래스들의 중첩된 부분 집합으로 이루어진 중첩 일반화(overlapping generalization)로서 주어진 객체타입이 여러 중복 가능한 객체타입들로 나누어질 수 있음을 나타낸다.

그림 2는 객체타입 'student'가 G-타입 측면인 'graduate'에 의해 객체타입 'gradstudent'로 특수화된 예를 보여준다.

G-타입 측면에서는 존재종속성[2,14]이 존재한다.

(2) P-타입 측면

P-타입 측면은 상위의 객체 클래스들을 하위의 객체 클래스들의 모임으로 나타내는 추상화로 PART-WHOLE 또는 A-PART-OF의 의미를 가진다[2]. 즉, P-타입 측면은 주어진 객체타입이 다수의 상호배제적인 객체 타입들의 합성에 의해 인식될 수 있음을 나타내는 것이다.

P-타입 측면에 의해 주어진 객체 타입과 관련되는 각 하위 객체 타입을 주어진 객체에 대한 성분 객체(component object)라고 한다. 반대로 성분객체 타입을 합성한 객체타입을 복합객체(composite object)라고 한다.

P-타입 측면은 existence 속성과 exclusive 속성에 의해 특정지워질 수 있다. 그림 2에서 성분객체 'arms', 'legs', 'body'는 복합객체 'student'를

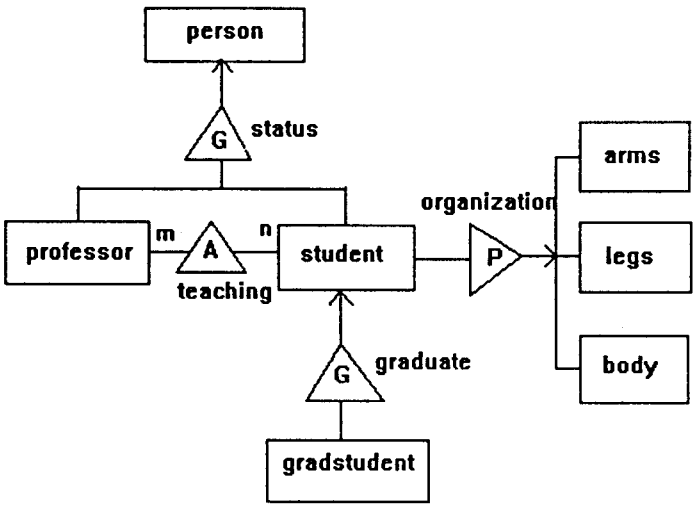

그림 2. 객체중심측면 모델의 예

구성한다.

(3) A-타입 측면

A-타입 측면은 주어진 객체타입의 객체들이 다른 객체타입의 객체들과의 연관성(association)에 의해 인식되는 측면이다[6]. MAO 모델에서는 R-타입 측면과 연관성 타입에 의해 연관성 추상화 개념을 표현하였으나 OOAM에서는 A-타입 측면 하나로 표현한다.

실세계의 엔티티 사이에는 많은 종류의 연관성이 존재하며 일반적으로 연관성은 참여하는 엔티티의 수에 따라 이항연관성(binary associatioin)과 N

항 연관성(N-ary association)으로 분류되고, 엔티티간의 함수성에 따라 일대일, 일대다, 다대다 등으로 구분된다[4,6]. 기존의 객체 중심 데이타베이스 시스템에서는 연관성을 직접적으로 표현할 수 있는 구문과 시맨틱을 제공하지 못하고 있다. 즉, 기존 방법은 특정 연관성을 특별한 방법으로 표현하거나 또는 연관성의 서로 다른 면을 특정 클래스의 메소드 또는 속성들로 구현하고 있다[2,4].

OOAM에서는 연관성을 클래스로 표현한다. 연관성에서 관련된 객체들은 sub에 다중성과 함께 표현된다. 그림 2에서 객체 타입 'professor'와 'student'은 각각 A-타입 측면을 갖고 있다.

A-타입에 의해 스키마는 비계층적으로 구성되어 다이어그램 표현이 복잡하게 된다.

3. 데이타베이스 모델링 공리

OOAM에 근거한 공리들을 보여주고 이 공리들이 실세계를 모델링할 때 어떻게 해석되는지 설명된다.

먼저 데이타베이스의 모델링 시작은 실세계의 모델링하고자 하는 모든 대상체인 데이타 객체들을 기술하는 것이다. 이들은 심볼릭 형태의 이름과 그에 대한 값들로 나타낸다. 데이타베이스에서 심볼 형태의 이름은 보통 property와 연관되는데 property는 각 객체들을 다른 객체들과 구분시키는 특성이다. 값들은 원자값 집합의 부류들로 구성되며 각 원자값의 집합은 하나의 시맨틱 개념을 표현한다.

보통 property 이름과 그 값을 속성(attribute)이라 한다. 속성은 실세계로부터 추출해 낸 쪼갤 수 없는 하나의 정보이다. 속성내에서 property 이름은 특정 시맨틱의 역할 값을 가지며 역할 값의 혼돈을 피하기 위해 속성은 하나의 원자값의 집합으로부터 하나의 요소를 취한다. 이는 다음과 같은 공리를 이끌어낸다.

(공리 1) 속성

각 속성은 쪼갤 수 없는 시맨틱 해석을 가진다.

객체는 실세계에서 독립적이고 개별적인 하나의 대상체로 표현된다. 객체의

property는 속성들에 의해 서술되고 속성의 한 부분이 그 객체를 유일하게 식별할 수 있게 해 준다. 즉 객체는 속성들의 집합으로 정의될 수 있다. 객체의 이름 자체는 어떠한 시맨틱 정보를 가지고 있지 않다. 객체들 중 같은 속성들의 집합을 갖는 것들이 있다. 이는 다음의 공리를 유도한다.

(공리 2) 클래스

데이타 객체 x에 대한 시맨틱을 S(x)라 할 때, S(x)를 만족하는 모든 데이타 객체들로 구성되는 클래스(class)가 존재한다.

만약에 속성의 값 부분을 제외하고 추상화한다면, 즉 property 집합에만 중점을 둔다면 클래스 타입(클래스 구조)를 얻을 수 있다. 데이타베이스 설계자는 각 클래스 타입에 대한 심볼 이름을 정의하며 데이타베이스의 모든 클래스에 대한 클래스 타입을 구축해야 한다. 만약에 두개의 클래스 타입이 동일한 property 집합을 가진다면 위의 공리에 의해 이는 동일한 클래스 타입으로 볼 수 있다. 데이타베이스 설계자가 동일한 property 집합을 가진 두개의 클래스 타입이 같은 시맨틱을 갖는다고 볼 수 없으면 어느 하나의 클래스 타입을 선택하여 속성들을 첨가하거나 삭제하여야 한다. 그러므로 다음과 같은 공리가 데이타베이스 설계에서 만족되어야 한다.

(공리 3) 클래스 타입

어떤 두 개의 클래스 타입도 같은 property 이름의 집합을 가질 수 없다.

개념적 모델 설계중에 클래스는 고립된 개념이 아니고 다른 클래스들과 관계를 갖는다. 이러한 관계성의 형태는 다양하다. 예를 들면, 클래스들은 속성들을 공유할 수도 있고, 속성들간에 함수적 관계가 있을 수도 있고, 어떤 클래스는 다른 클래스 구조의 구성요소가 될 수도 있다. 모든 경우에 관계성은 존재하는 클래스의 합집합으로 생각될 수 있으며 관계성의 property를 나타내기 위해 속성들이 추가될 수 있다. 그러므로 관계성은 별개의 개념이 아니라 클래스 타입과 동일한 개념으로 볼 수 있다.

(공리 4) 관계성

관계성은 클래스 타입이다.

OOAM에서 데이타 객체들은 그 객체를 이해하는 모델링 관점에 따라 여러 측면에서 기술될 수 있다. 즉 실세계는 다중 측면 객체들의 모임이다. 이는 모델 구성자의 이해 관점에 따라 실세계를 여러 측면으로 객체들을 서술하는 것이다. 이는 각 클래스 타입에 대해 여러개의 관계성을 허락한다.

(공리 5) 측면

측면은 관계성으로 클래스 타입에 대해서만 정의되며, 하나의 클래스 타입에 대해 여러개의 측면이 존재할 수 있다.

위에서 유도된 공리들은 데이타베이스 설계 과정에서 다음과 같은 데이타베이스의 간략한 기술을 얻는데 사용될 수 있다.

가. 설계하고자 하는 데이타베이스에서 발견할 수 있는 모든 클래스 타입들을 선정한다. 클래스 공리에 따라 두개의 클래스 타입은 동일한 property를 가질 수 없다. 만약에 두개의 클래스 타입이 동일한 것으로 간주된다면 다중 역할로 간주하여 역할에 대한 property를 첨가한다. 즉 각 클래스 타입들은 다른 클래스 타입에 대해 독립적으로 인식되는 모델링 대상의 주요 개념, 사물 등이 된다. 클래스 공리를 만족하도록 클래스 타입의 개념들은 상호 배제적이어야 한다.

나. 각 클래스 타입에 대한 property 이름 집합, 원자 값 집합, 그에 따른 속성들을 선정한다. 속성 공리에 따라 각 속성에 대한 원자값 집합이 모호하지 않도록 한다.

다. 클래스 타입에 대해 측면을 설계한다. 이는 측면 공리에 따라 클래스 타입에 대해서만 정의되며 여러개의 측면이 존재할 수 있다. 측면이 속성을 가질 때 이 속성들은 측면에 대한 클래스에 표기된다.

4. 데이타베이스의 intension

위에서 제시한 설계 과정에서의 공리를 기반으로 하여 데이타베이스의 intension, 즉 OOAM의 데이타베이스 스키마의 형식론을 제시한다.

정의 1. KOA

임의의 서로 다른 두개의 클래스는 세가지 측면 관계 중 하나를 이용하여 연결될 수 있는데, 이 세가지 측면을 KOA(Kind of Aspect)라 한다. 즉, 다음과 같은 세가지 집합을 KOA라 한다.

KOA = { N_typeA, N_typeP, N_typeG }

예. 그림 2에서 나타나는 측면은 4가지이다.

KOA = { teaching_typeA, organization_typeP,
status_typeG, graduate_typeG }

정의 2. DB 스키마 S

다음과 같은 형식의 클래스 스킴들의 집합을 데이타베이스 스키마 S라 한다.

$\{C_1, C_2, C_3, \ldots, C_n\}$ KOA $\{C_1', C_2', C_3', \ldots, C_m'\}$
$C_n\{P_1, P_2, P_3, \ldots, P_i\}$
$C_m'\{P_1', P_2', P_3', \ldots, P_j'\}$
$KOA\{P_1'', P_2'', P_3'', \ldots, P_k''\}$

여기서, $\forall i,j,k \in N$,
$\forall m,n \in N$, C_n 과 C_m' 는 클래스 이름 ,
$\forall i,j \in N$, P_i 와 P_j'는 클래스 C_n과 C_m'이 가지고 있는 속성,
$\forall k \in N$, P_k''는 측면이 가지고 있는 속성

예. 그림 2에 대한 데이타베이스 스키마 S는 아래와 같다.

{person} status_typeG {professor, student}
{student} graduate_typeG {gradstudent}
{professor} teaching_typeA {student}
{student} teaching_typeA {professor}
{arms, legs, body} organization_typeP {student}
person{snumber}
student{id, department}
professor{major}

gradstudent{term}

arms{}

legs{}

body{}

teaching_typeA{course, room}

organization_typeP{}

status_typeG{}

gradstudent_typeG{}

정의 3. Props(C), Classes(S), Props(KOA)

데이타베이스 스키마 S를 구성하는 모든 클래스 이름을 Classes(S)라 하고, 클래스 C와 KOA가 갖는 속성들의 집합을 각각 Props(C)와 Props(KOA)라 한다. 즉,

$$\forall n \in N, \quad \cup C_n = Classes(S)$$

$$\forall m \in N, \quad \cup P_m = Props(C)$$

$$\forall K \in N, \quad \cup P_k = Props(KOA)$$

예. 그림 2 에서 클래스들의 집합은 아래와 같다.

Classes(S) = {person, student, professor, gradstudent,
arms, legs, body}

클래스 student에 대한 속성들의 집합은

Props(student) = {id, department}

이고 측면 teaching_typeA에 대한 속성들의 집합은

Props(teaching_typeA) = {course, room}

이다.

정의 4. immediate referclass

정의 2.의 클래스 스킴에서 $\{C_1, C_2, C_3, \ldots, C_n\}$ 을 측면 KOA 에 대한 $\{C_1', C_2', C_3', \ldots, C_m'\}$의 immediate referclass 라 하고 다음과 같이 표기하기로 한다.

$$\{C_1', C_2', C_3', \ldots, C_m'\} \xrightarrow{KOA} \{C_1, C_2, C_3, \ldots, C_n\}$$

예. 그림 2에서 클래스 person은 측면 status_typeG에 대해 클래스

professor와 student 의 immediate referclass이다.

$$\{professor,\ student\} \xrightarrow{status\text{-}typeG} \{person\}$$

정의 5. referclass

다음과 같은 조건을 만족하는 $\{C_1, C_2, C_3, \ldots, C_n\}$ 를 측면 KOA에 대한 $\{C_1', C_2', C_3', \ldots, C_m'\}$의 referclass 라 한다.

1) $\{C_1', C_2', C_3', \ldots, C_m'\} = \{C_1, C_2, C_3, \ldots, C_n\}$

2) $\{C_1', C_2', C_3', \ldots, C_m'\} \xrightarrow{KOA} referclass\{C_1, C_2, C_3, \ldots, C_n\}$

이때, 측면 KOA에 대한$\{C_1', C_2', C_3', \ldots, C_m'\}$의 referclass $\{C_1, C_2, C_3, \ldots, C_n\}$를 다음과 같이 표기하기로 한다.

$$\{C_1', C_2', C_3', \ldots, C_m'\} \overset{KOA}{\Rightarrow} \{C_1, C_2, C_3, \ldots, C_n\}$$

예. 그림 2에서 클래스 gradstudent의 typeG에 대한 referclass는 person, student, gradstudent이다.

$$\{gradstudent\} \overset{typeG}{\Rightarrow} \{person,\ student,\ gradstudent\}$$

위에서 정의한 referclass에 의하여 우리는 클래스가 참조할 수 있는 모든 속성들을 정의할 수 있는데 이 때 필요한 표기법 두가지를 다음과 같이 약속하기로 한다.

(1) $[props(C) \times props(KOA)]^{n+1}$

$\equiv [props(C) \times props(KOA)] \times$

$[props(C_1) \times props(KOA)] \times$

....

$[props(C_n) \times props(KOA)]$

(2) $[props(KOA) \cup props(C)]^{n+1}$

$\equiv [props(KOA) \cup props(C)] \cup$

$[props(KOA) \cup props(C1)] \cup$

....

$[props(KOA) \cup props(C_n)]$

단, $C \xrightarrow{KOA} C_1$, $C_1 \xrightarrow{KOA} C_2$, $C_2 \xrightarrow{KOA} C_3$, ... $C_{n-1} \xrightarrow{KOA} C_n$ 이고

$[props(C) \times props(KOA)]^0 \equiv [props(KOA) \cup props(C)]^0 \equiv \emptyset$ 이다.

정의 6. RProps*(C)

임의의 클래스 C가 참조할 수 있는 모든 속성들의 집합을 RProps*(C)이라 한다. 즉, RProps*(C)은 아래의 두 조건을 만족하는 집합과 같다.

1. Props(C)
2. $\bigcup_{n=1}^{C \overset{KOA}{\Rightarrow} \exists C'}$ $[props(KOA) \cup props(C')]^n$ (단 $C \neq C'$)

예. 그림 2에서 클래스 student가 참조할 수 있는 속성들은 클래스 student, person, professor, arms, ledgs, body가 가진 속성들과 측면 status_typeG, teaching_typeA, organization_typeP가 가지는 속성들의 합집합이다.

RProps*(student)={snumber, id, department, major, course, room}

정의 7. PF(S) (path function over DB schema S)

데이타베이스 스키마 S가 주어졌을 때, 모든 클래스 C (C∈classes(S)) 에 대해서 다음과 같은 집합을 S에 대한 PF라 한다.

$$PF(S) = \bigcup_{n=0}^{C \overset{KOA}{\Rightarrow} \exists C'} [props(C) \times props(KOA)]^n \times props(c') \quad \cup \quad \bigcup_{n=0}^{C \overset{KOA}{\Rightarrow} \exists C'} [props(C) \times props(KOA)]^n$$

단, $\emptyset \times props(C) \equiv props(C)$ 로 약속한다.

예. 클래스 gradstudent에서 typeG에 대해 나타나는 path function은 아래와 같으며 이와 같은 방법으로 모든 클래스에 대해 path function을 만들 수 있다.

PF = { (term), (term, id), (term, department),

(term, id, snumber), (term, department, snumber) }

위에서 정의한 path function에 의해 데이타베이스 스키마에서 정보들을 찾아가는 경로를 나타낸다. 특히 측면에 의해 유도되는 특성을 규칙으로 나타낼 수 있다.

먼저 이 규칙들을 나타내기 위한 predicate는 아래와 같다.

- typeG(S,G) : 클래스 G가 G-측면에 의해 클래스 S를 가진다.
- typeP(C,P) : 클래스 P가 P-측면에 의해 클래스 C를 가진다.
- typeA(C,A) : 클래스 A가 A-측면에 의해 클래스 C를 가진다.

다음 규칙들은 OOAM의 시맨틱 구조에서 발생하는 추론 규칙들이다.

규칙 1. C(p) & typeG(B,C) → B(p)

이는 G-측면에서 나타나는 property 상속에 대한 메카니즘이다.

규칙 2. typeG(A,B) & typeG(B,A) ↔ A=B

규칙 3. typeG(A,B) & typeG(B,C) → typeG(A,C)

규칙 4. typeG(A,B) & typeG(A,C) → ∃ D(typeG(B,D) & typeG(C,D))

규칙 5. typeP(A,B) → ~typeP(B,A)

규칙 6. typeP(A,B) & typeP(B,C) → typeP(A,C)

규칙 7. typeG(A,B) & typeP(B,C) → typeP(A,C)

규칙7은 항상 성립하나 그의 반대 즉,

typeP(A,B) & typeG(B,C) → typeP(A,C)는 항상 참이 아니다.

다음은 OOAM의 특성인 다중측면을 제공함으로써 발생되는 측면추론들이다.

규칙 8. typeG(A,B) & typeP(A,C) → typeP(B,C)

규칙 9. typeG(B,C) & typeP(A,C) → typeP(A,B)

규칙 10. typeG(A,B) & typeA(A,C) → typeA(B,C)

제시한 규칙들은 클래스들간의 관계에 의해 명확한 시맨틱을 나타낸다. 이 시맨틱은 실세계를 정확하게 모델링할 수 있는 기본이 되고 지식베이스에 구축되어 질의처리에 도움을 줄 수 있다.

5. 데이타베이스 extensions

이 장에서는 데이타베이스의 extensions, 즉 실제 저장되는 데이타에 대한 형식적인 정의를 내린다. 특히 클래스 타입 스페이스가 extension 스페이스로 잘 대응될 수 있도록 클래스와 클래스 타입의 관계를 보여준다.

속성이란 property 이름(심볼)과 그에 대한 값(원자값)의 관계이다. 즉, 속성은 객체의 한 모델링 특성을 나타내는 것으로 객체들은 여러 속성들을 가진다. 심볼에 대한 값은 원자값의 집합으로 부터 취한다고 가정하면 속성값은 유한 집합의 원소이다.

도메인은 한 속성이 가질 수 있는 가능한 모든 값들의 집합이다.

정의 8. 도메인(Domain)

d_a를 속성 a의 도메인이라 하면 클래스 타입 C의 도메인 D_c은 다음과 같다.

$$D_c = \prod_{a \in A} d_a$$

클래스 타입 C의 인스턴스들의 집합은 클래스 I라 하고 이는 $P(D_c)$의 원소이다. 왜냐하면 속성이나 인스턴스들이 null이거나 불완전한 정보를 가질 수 있기 때문이다. 또 클래스 타입 C의 인스턴스 i는 I의 원소이다.

클래스 공리에 따라 클래스 타입은 속성에 의해 결정된다. 이는 다음과 같은 extension mapping 정의를 이끌어낸다.

정의 9. 프로젝션 함수 π_c

클래스 I에 속해 있는 모든 인스턴스 i를 D_c로 프로젝션(projection) 하는 함수를 π_c 라고 하고 다음과 같이 표기한다.

$$\pi_c : I \rightarrow P(D_c)$$

다음으로 클래스 타입과 인스턴스의 관계를 형식적으로 정의한다. 각 extension은 도메인의 부분집합으로 나타내기 때문에 각 클래스 타입에 대한 함수의 집합이 있어야 한다. 그러므로 클래스 타입에 대한 extension은 다음과 같이 정의할 수 있다.

정의10. 함수 E_c

스키마 S에 속해 있는 각 클래스 타입 C 를 $\pi_c(I)$ 로 프로젝션 시키는 함수를 E_c 라 하고 다음과 같이 표기한다.

$$E_c : Classes(S) \rightarrow P(D_c)$$

데이타베이스의 extensions은 위의 정의에 의한 각 클래스 타입의 extension의 집합이다. 데이타베이스의 intension을 extension으로 mapping하는 것은 데이타베이스의 상태를 나타내는 무결성 제약조건 기능을 한다. 즉 mapping은 타당한 도메인으로의 extension만 허용한다.

이에 유도되는 규칙을 서술하기 위해 다음과 같은 predicate를 사용한다.

- instance(e,C) : e는 클래스 C의 인스턴스이다.
- C(p,x,y) : 클래스 C의 인스턴스 x가 property p에 대해 y값을 가진다.

규칙 11. instance(e,C) & typeG(C,D) → instance(d, D)

인스턴스 d는 인스턴스 e의 부분집합이다.

규칙 12. C(p,x,y) & typeG(B,C) → B(p,x,y)

이는 property값이 G-측면에서 상속됨을 의미한다.

6. 결론

본 논문에서는 OOAM에 의해 구축되는 데이타베이스 스키마의 시맨틱을 분석하고 서술하기 위해 OOAM을 형식적으로 정의하였다. 먼저 데이타베이스 설계 과정에서 필요한 공리들을 설정하고 OOAM을 intension과 extension으로 각각 정의한 후 그들 사이의 관계를 정의하였다.

클래스란 객체들의 집합으로 속성들의 집합에 의해서만 특정 시맨틱을 갖는다. 이와 같은 접근 방법으로 OOAM에 의해 구축되는 스키마의 시맨틱을 분석하기가 용이하며 그에 관련된 공리들(속성, 클래스, 클래스 타입, 측면 등)을 설정했다. 이는 데이타베이스 설계 과정에서 데이타베이스 설계의 간략한 기술을 얻는데 사용된다.

그리고 이들 공리를 기반으로 하여 OOAM에 의해 구축되는 데이타베이스 스키

마의 형식론을 정의한 후에 데이타베이스 스키마에서 정보들을 찾아가는 path function을 정의하였다. 특히 OOAM의 측면에 의해 유도되는 여러 개의 규칙들을 보여주고 extension을 정의한 후에 이들 사이의 관계도 나타내었다. 제시한 규칙들은 클래스들간의 관계에 의해 명확한 시맨틱을 가지며 지식베이스에 구축될 수 있다.

추후 연구과제로는 유도된 규칙들을 KB/DB 통합시 실제 KB 설계에 이용할 수 있는 방법론의 개발과 응용, path function에 대한 depenency와 추론 메카니즘 등의 형식적인 접근 등이 있다.

<참고논문>

[1] Arno Siebes, "Using Design Axioms and Topology to Model Database Semantics", Proceedings of the 13th VLDB Conference, 1987.

[2] Banerjee, J., Chou, H. T., Garza, J. F., Kim, W., Woelk, D., and Ballou, N., "Data Model Issues for Object-Oriented Appicatoins", ACM Trans. on Office Information Systems, 5(1), January, 1987.

[3] Chen Weidong, David S. Warren, "C-Logic of Complex Objects", Proceedings of the 8th ACM'S SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, March 1989.

[4] D. N.,Smith, Concepts of Object-Oriented Programming, McGraw-Hill, 1991.

[5] Grant E.Weddell, "A theory of Functional Dependencies for Object-Oriented Data Models", The first Internationsal Conference on Deductive and Object-Oriented Databases, 1989.

[6] J. E., Rumbaugh, et. al., Object-Oriented Modeling and Design, Prentice-Hall, 1991.

[7] Masahiko Tsukamoto, "DOT:A Term Representation Using DOT Algebra for Knowledge-Bases", Second Internatonal Conference on Deductive and Object-Oriented Databases, 1991.

[8] Michael Kifer, Georg Lausen, "F-Logic:A Higher-Order Language for Reasoning about Objects, Inheritance, and Scheme", SIGMOD RECORD, 1989.

[9] Michael Kifer, James Wa, "A Logic for Object-Oriented Logic

Programming", SIGMOD RECORD, 1989.

[10] M. R. Blaha, W. J. Premerlani, and J. E. Rumbaugh, "Relational Database Design using an Object-Oriented Methodology", Comm. of the ACM, Vol.31, No.4, April, 1988.

[11] Stephen W. Clyde, David W. Embley, Scott N. Woodfield, "Tunable Formalism in Object-Oriented Systems Analysis:Meeting the Needs of Both the Theoreticians and Practitioners", OOPSLA'92.

[12] Ullman, Principles of Database Systems, Computer Science Press, 1982.

[13] Ulrich Schiel, "Abstractions in Semantic Networks", ACM SIGART Newsletter, No.107, 1989.

[14] Won Kim, Introduction to Object-oriented Databases, The MIT Press, 1990.

[15] 김형주, "성질 계승 그래프:객체 중심 데이타베이스 스키마를 위한 형식 모델", 한국정보과학회 논문지, Vol.16, No.5, 1989.

[16] 오선영, 백두권, "객체 중심 측면 모델에 의한 개념적 데이타베이스 모델링", 한국정보과학회 학술발표논문집, Vol.19, No.2, 1992.

[17] 오선영, 백두권, "객체 중심 측면 모델에 의한 KB/DB의 설계", 한국정보처리응용학회 춘계 학술발표논문집, 제1권, 제1호, 1994.

[18] 조동영, "다중측면을 고려한 객체중심 데이타베이스 모델링", 고려대학교, 박사학위논문, 1991.