갱신 의미 보존 객체-지향 뷰

나 영 국†

요 약

데이터 모델링 능력이 한정되고 뷰 갱신 모호성 문제에 기인하여, 관계형 뷰는 공학 응용에 제한적으로 사용되어 왔다. 반면에 객체지향 데이터베이스의 뷰는 관계형 뷰의 이 두 가지 단점을 극복하기 때문에, 공학 응용을 위한 맞춤 인터페이스를 정의하는데 중요한 역할을 할 것이다. 특히 공학 응용을 위한 데이터베이스 인터페이스는 갱신을 충분히 지원하여 한다. 좀더 자세히 말하면, 인터페이스에 대한 갱신이 모호성이 없이 정의되어야 하며 이 정의는 베이스 스키마에 대한 갱신 행동과 일치하여야 한다. 이를 위하여 객체지향 뷰가 베이스 데이터 모델과 같은 갱신 행동을 보이기 위한 제반 조건 - 갱신 의미 보존(update semantic preserving) - 을 정의하였다. 그리고 이 갱신 의미 보존 특성의 실현 가능성을 보이기 위하여 CAD에 특화된 객체지향 뷰 시스템, 멀티 뷰(MultiView), 을 선정하여 그 시스템의 뷰 모델에 대한 갱신 의미 보존 갱신 행동을 정의하고 구체적인 구현 알고리즘을 제시하였다. 이 연구는 객체지향 데이터베이스에서 가상 클래스를 모았을 때 단순한 클래스의 모임이 아니라 isa계층을 갖는 '스키마'가 될 수 있게하기 위해서는 가상 클래스에 대한 갱신 의미가 클래스간 isa 관계를 위반해서는 안된다는 것을 발견하였다. 그리고 이의 충분조건으로 '뷰 스키마가 베이스 스키마처럼 보이도록' 하는 가상 클래스의 갱신 의미와 가상 클래스간 스키마 형성 가능 조건을 발견하였다. 이는 객체 지향 데이터베이스에서 뷰를 클래스 수준에서 스키마 수준으로 정의하는 충분조건을 발견하고 구현한 최초의 논문이다.

키워드 : 갱신 의미 보존, 객체-지향 뷰, 데이터 인터페이스, 객체-지향 데이터베이스, 뷰 갱신

Update Semantic Preserving Object-Oriented View

Young-Gook Ra†

ABSTRACT

Due to the limitation of data modeling power and the view update ambiguity, relational view is limitedly used for engineering applications. On the contrary, object-oriented database view would play a vital role in defining custom interface for engineering applications because the above two limitations of the relational view are overcome by the object-oriented view. Above all, engineering application data interface should fully support updates. More specifically, updates against the data interface needs to be unambiguously defined and its semantic behavior should be equal to base schema updates'. For this purpose, we define the notion of update semantic preserving which means that view updates displays the same semantics as base schema. Besides, in order to show the feasibility of this characteristics, specific and concrete algorithms for update preserving updates are presented for a CAD specialized object-oriented database view - MultiView. This paper finds that in order that virtual classes coudld form a schema with 'isa' relationships rather than just a group of classes, the update semantics on the virtual classes should be defined such that the implied meaning of 'isa' relationships between classes are not to be violated. Besides, as its sufficiency conditions, we derived the update semantics and schema constituable conditions of the virtual classes that make view schemas look like base schemas. To my best knowledge, this is the first research that presents the sufficiency conditions by which we could defined object-oriented views as integrated schemas rather than as separate classes.

Key word : update semantic preserving, object-oriented view, data interface, object-oriented database, view update

1. 서 론

CAD와 같은 공학용 응용의 데이터 저장소는 글로벌 데이터 모델뿐만 아니라 각 도구에 디자인 데이터의 도구-특화된 표현을 제공할 필요가 있다. 그럼으로써 복잡한 디자인 정보에서 관련이 없는 부분이 걸러진다. 즉, 공학 응용의 데이터 저장소는 도구-특화된 인터페이스를 제공할 수 있는 데이터베이스 뷰 기능을 제공하여야 한다[1-3].

기존의 관계형 뷰는 공학적인 소프트웨어 도구들이 필요로 하는 정도로 데이터베이스에 대한 맞춤(customization)을 충분히 제공하지 못하는 단점이 있다. 더욱이, 대부분의 상용 데이터베이스 뷰는 공학적인 어플리케이션이 요구하는 갱신 기능을 허락하지 않고 질의에만 초점을 두고 있다. 객체지향 뷰는 기존의 관계형 뷰에 차별화 하여 (1) 맞춤 능력이 월등하여 다양한 공학 응용 프로그램의 데이터 요구사항을 만족시켜야 하며 ; (2) 갱신이 잘 정의되어 공학 응

† 정 회 원 : 국립한경대학교 컴퓨터공학과 교수
논문접수 : 2000년 4월 17일, 심사완료 : 2001년 1월 9일

용 프로그램의 관점에서 뷰에 대한 갱신와 베이스 스키만에 대한 갱신를 구별할 수 없어야 한다.

첫번째 이슈, 즉 뷰의 모델링 능력에 대하여 많은 연구가 이루어져 왔다. 초기에는 객체지향 모델의 타입과 쿼리를 이용하여 객체를 변환하여 이를 뷰로 정의하였다[4-7]. 객체지향 모델을 위한 뷰 의미(emantic)는 Abiteboul과 Bonner[4]에 의해 본격적으로 탐구되었다. 객체지향 모델에서 뷰 정의가 완전해지기 위해서는 가상 클래스가 클래스 계층구조에 통합되어야 한다. 이러한 통합 이슈를 포함한 최초의 완전한 뷰 메카니즘 멀티뷰(MultiView)가 Rundenstein-er[8-10]에 의해 제안되었다.

그러나 비록 객체지향 뷰가 CAD 응용 프로그램이 요구하는 복잡한 데이터 인터페이스를 성공적으로 제공한다 하여도 이 인터페이스에 갱신이 되지 않는다면 그 효용성이 상당히 저하된다. CAD 응용 프로그램의 특성상 데이터를 조회하는 시간보다 디자인을 변경하는 데 사용자는 더 많은 시간을 사용하고 따라서 CAD 데이터에 대한 빈번한 갱신이 이루어진다. 그러므로 객체 뷰는 관계형 뷰와는 달리 필수적으로 갱신를 지원하여야 한다.

더욱이 주의할 점은 객체지향 뷰 인터페이스에 대한 갱신는 CAD 응용 프로그램의 관점에서는 베이스 데이터에 대한 갱신로 보여야 한다는 것이다. 예를 들어 자신의 뷰에 있는 클래스 A에 새로운 객체 a를 생성하라고 명령하였는데 이 객체가 클래스 A와 상관이 없는 클래스 B에도 같이 생성된다면 이 응용 프로그램은 개발자가 예상하지 못하는 데이터베이스 반응을 접하게 되고 이로 인해 응용 프로그램 자체가 예기치 못한 행동을 사용자에 보일 수도 있다.

이처럼 객체지향 뷰는 그 활용 영역의 요구사항 특성상 베이스 스키마와 같은 갱신 행동을 보여야한다. 이를 데이터베이스 뷰가 베이스 스키마의 갱신 의미를 보존하였다고 한다. 우리는 객체지향 뷰에 대하여 세 가지 연산자 생성, 변경, 제거를 정의한다. 그리고 객체지향 뷰가 갱신 의미 보존을 하려면 위의 세 가지 연산자 생성, 변경, 제거의 행동이 베이스 스키마와 같게 정의되어야 한다.

이처럼 갱신 의미 보존이 실현된 객체 지향 뷰로 CAD 응용 프로그램의 데이터 인터페이스를 생성하면 이 응용 프로그램은 특수한 목적에 가장 적합한 데이터 포맷을 가지고 있을 뿐만 아니라 모든 갱신가 가능한 자신만의 데이터 발자는 데이터베이스의 경직된 구조에 얽매이지 않고 데이터 호환성을 염려하지 않으면서 원하는 도구를 만들어 낼 수 있게 된다.

본 논문은 첫 번째 이슈인 뷰 모델링 능력에 관한 연구에 비해 그 중요성에도 불구하고 상대적으로 관심이 적은 두 번째 이슈인 뷰 갱신에 집중한다. 갱신 의미 보존을 객체지향 데이터 모델과 뷰 모델에 관점에서 엄밀하게 정의한다. 공식 표준의 객체지형 데이터 모델과 뷰 모델이 없기 때문에 데이터 모델을 정의하였으며 뷰 모델은 CAD용 뷰로써 미시간 대학에서 오랫동안 연구되어온 멀티뷰를 근간으로 재 정의하였다.

특히 이 논문은 갱신 의미 보존을 클래스 수준이 아니라 계층구조를 가진 객체지향 스키마 수준에서 달성하였다는 것을 주목할 필요가 있다. 우리는 객체지향 데이터베이스에서 가상 클래스를 모았을 때 단순한 클래스의 모임이 아니라 isa계층을 갖는 '스키마'가 될 수 있게 하기 위해서는 가상 클래스에 대한 갱신 의미가 클래스간 isa 관계를 위반해서는 안된다는 것을 발견하였다. 그리고 이의 충분조건으로 '뷰 스키마가 베이스 스키마처럼 보이도록' 하는 가상 클래스의 갱신 의미와 가상 클래스간 스키마 형성 가능 조건을 발견하였다. 이는 객체 지향 데이터베이스에서 뷰를 클래스 수준에서 스키마 수준으로 정의하는 충분조건을 발견하고 구현한 최초의 논문이다.

2장은 이 연구의 배경 설명으로 지금까지의 객체지향 뷰에 관한 주요 연구 활동이 소개된다. 3장에서는 이 연구가 가정한 객체 모델과 뷰 모델이 각각 정의된다. 4장에서는 뷰 스키마의 갱신 의미 보존이 정의된다. 5장에서 이론을 바탕으로 구체적인 갱신 의미 보존 뷰 모델이 정의된다. 6장이 이 논문을 결론짓는다.

2. 객체지향 뷰

관계형 데이터베이스의 뷰 메카니즘에 관한 중요성을 인식하여, 많은 연구자가 객체지향 데이터베이스를 위한 뷰를 제안하였다. 객체지향 데이터 모델이 관계형 데이터베이스에 비해 풍부하고 복잡하기 때문에, 제안된 뷰 메카니즘은 관계형 뷰의 메카니즘과는 매우 다르다. 뷰를 분류하는 하나의 주요한 기준은 뷰를 하나의 가상 클래스로 보느냐 아니면 베이스 클래스와 가상 클래스로 구성된 스키마로 보느냐 하는 것이다.

Abiteboul과 Bonner[4]는 O_2 객체지향 모델을 위한 뷰의 여러 가지 의미를 탐구하였다. 그들은 계산된 속성(저장이 아니라)을 정의하고 새로운 클래스(가상 클래스)를 생성하는 메카니즘을 소개하였다. 가상 클래스의 객체들은 조인 또는 다른 복잡한 객체로써 *가상 객체*라 불린다. 그들은 클래스 계층구조에 가상 클래스의 통합을 언급하였다. 그러나 그들이 제안하는 통합 해결책은 통일된 위 방향 상속보다는 선택적인 아래/위 방향 상속을 필요로 한다[5].

Heiler[6]는 비젼(Vision) 데이터베이스[7]의 확장인 퓨즈 객체 모델에 기초한 뷰 메카니즘을 제안하였다. 단지 모델의 타입 시스템과 질의 언어가 객체 표현을 변환하는데 사용된다. 이러한 뷰의 개념은 이 시스템이 각 타입 정의를 하나의 릴레이션으로 다루는데 있어서 관계형 뷰의 개념과 매우 유사하다. 이 시스템은 뷰를 지원하기 위하여 추가의

메카니즘이 필요 없다는데서 특이하다. 뷰의 타입을 계층 구조에 통합하는 문제는 다루지 않고 있다.

Scholl을 포함한 몇 명의 연구자[11] 커쿤(Cocoon) 객체 모델[12]의 컨텍스트(Context)에서 뷰 시스템을 소개한다. 관계형 데이터베이스 관리 시스템처럼, 뷰는 질의에 의해 정의된다. 그러나, 관계형의 경우와는 달리, 그들은 가상 클래스와 속성의 각 타입 별로 특수한 갱신 메소드(method)를 통해 특정한 갱신 의미를 구현함으로써 뷰 갱신 의미의 모호성 문제가 해결된다고 제안한다. 요구 사항으로, 그들은 갱신 가능한 객체 뷰를 허용하기 위해서 객체지향 질의 언어가 *객체 보유*여야 한다고 얘기한다. 저자들은 뷰를 타입과 클래스 *격자에 포함시키기 위해 분류의 필요성을 확인하였다. 그러나, 완전한 분류를 위한 알고리즘은 제시되지 않았다.*

Danaka외 몇 명[13]은 사용자에게 데이터베이스에 대한 여러 개의 뷰를 제공하기 위하여 스키마 가상화(virtualization) 기술의 개념과 구현을 설명한다. 첫째, 가상 클래스와 가상 스키마의 개념이 소개되고 다음에, 스키마를 가상 스키마로 변환하는 과정이 토의된다. 스키마 버추얼라이제이션은 뷰 클래스를 글로벌 스키마에 통합하는 것과 유사하나, 아래 방향 상속이 어떤 클래스에 요구되고 다른 타입의 가상 클래스에는 위 방향 상속이 요구된다.

Rundensteiner[8]는 클래스 분류[9]와 뷰 스키마 생성[10]의 문제가 완전히 다루어지고 자동화된 해결책이 개발됐다는 의미에서 다소 완전한 뷰 메카니즘을 제안하였다. 첫째, 가상 클래스는 객체지향 질의에 의해 정의되고 그 가상 클래스가 베이스 클래스와 함께 글로벌 스키마를 형성한다. 뷰 스키마는 글로벌 스키마로부터 원하는 클래스를 선택하고 isa 관계가 그 선택된 클래스들 사이에 자동 생성됨으로써 생성된다. 이 시스템의 초기 프로토타입[14]이 미시간 대학에서 젬스톤(GemsStone) 객체지향 데이터베이스를 이용하여 구축되었다.

3. 배경 모델

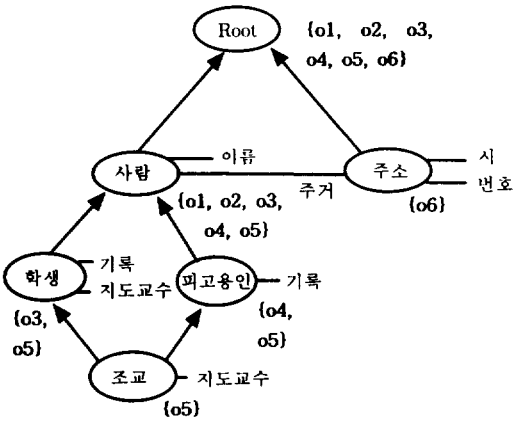
이 장에서 이 연구의 기본이 되는 객체 모델과 뷰 모델이 각각 정의된다.

3.1 객체 모델

이 장은 이 연구에서 가정한 기본적인 객체 데이터 모델을 정의한다. 객체 모델은 고유의 갱신 오퍼레이션을 포함하여 상당히 표준적인 객체지향 객체 모델이다. 아래에, 우리는 객체지향 데이터베이스 모델의 기본 개념을 소개한다.

클래스 C는 유일한 클래스 이름을 갖고, 타입 기술(set description)과 소속 집합(set membership)을 갖는다. 우리는 두 개의 클래스를 서로 연관시키는 특별한 목적의 속성을 릴레이션쉽(relationship) 이라 부르고 이를 r로 표시한다. 두 개의 클래스(같을 수 있음) C_1과 C_2가 릴레이션쉽 r로 연결될 때, 그 연결을 $r(C_1, C_2)$로 표시한다. 클래스 C에 속하는 객체의 집합을 *외연*(C) = {o | o ∈ C}으로 표현하고 이때 소속 조건 ∈ 는객체 인스턴스의 객체 식별자(identifier)에 기반 한다. 객체 o, o ∈ 외면(C), 에 정의된 성질의 집합을 *타입*(C)라 부른다.

(그림 1)의 스키마는 사람, 학생, 피고용인, 조교, 주소 클래스를 포함하고 있다. 이중 학생 클래스는 '기록'과 '지도교수' 속성을 가지고 피고용인 클래스는 '기록' 속성을 갖는다. (그림 1)의 스키마에서 '사람'과 '주소' 클래스 사이에 릴레이션쉽 '주거(사람, 주소)'가 정의되어 있다. 그럼으로써 '주거' 릴레이션쉽의 인스턴스들이 '사람'과 '주소' 클래스 인스턴스들을 관련짓는다. (그림 1)에서 위의 정의에 따라 외연(사람) = {o1, o2, o3, o4, o5} 등으로 각 클래스의 외연이 집합으로 보여진다. 그리고 위의 클래스 타입 정의에 의하면 타입(사람) = {기록, 지도교수}임을 알 수 있다.



(그림 1) 객체 스키마

비공식적으로, (1) C_1의 모든 멤버(member)가 C_2의 멤버(부분집합 관계)이고 (2) C_2에 정의된 모든 성질이 C_1에도 역시 정의될 경우 (서브타입 관계)에 C_1이 C_2에 isa 관계에 있다고 한다. (그림 1)에서 클래스간의 isa 관계는 화살표로 표시되어 있다. 예를 들어 '조교' 클래스에서 '학생' 클래스로 화살표가 그려져 있으면 이는 '조교'가 '학생'의 서브클래스임을 나타낸다. (그림 1)에서 외연(학생) ⊂ 외연(조교)이고 타입(학생) ⊃ 타입(조교)임을 확인할 수 있고 이는 우리의 서브클래스 정의에 부합된다.

3.2 고유의 갱신 (Generic Updates)

우리는 데이터 모델을 정련하여 고유의 갱신를 포함하도록 한다. 객체지향 데이터베이스에서는, 갱신는 일반적으로 타입-특수한 갱신 메소드로 수행된다. 그러나, 우리는 다른 뷰 시스템[8]에서 제안한 것처럼 타입-특수한 갱신를 확장

하는 고유의 갱신 오퍼레이션의 집합을 제공하길 원한다. 그 고유의 갱신 오퍼레이션은 객체를 생성하고 소멸하는 *생성*과 *제거*를 포함하고 속성을 새로운 값으로 만드는 *갱신*을 포함한다.

- 생성 연산, (<클래스> 생성)으로 정의, 은 <클래스>의 인스턴스를 생성하여 그 클래스에 더한다.
- 제거 연산, (<객체> 제거)로 정의, 은 객체를 소멸시킨다.
- 갱신 연산, (<오브젝트> 갱신 [값 할당])로 정의, 은 <객체>의 속성에 새로운 값을 할당한다. 예를 들면, (<객체> 갱신 [사람.월급 = 3,000])은 사람 클래스의 <객체> 의 월급 속성 값을 3,000으로 할당한다.

3.3 뷰 모델

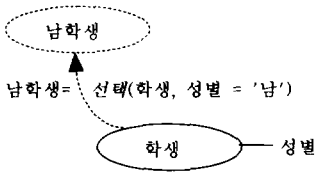
우리는 베이스와 가상 클래스를 구분한다. 베이스 클래스는 초기 스키마 정의할 때 생성된다. 객체 인스턴스는 베이스 객체로 명시적으로 저장되는 베이스 클래스의 회원들이다. 가상 클래스는 객체지향 질의를 사용하여 데이터베이스 운용 중 정의된다. 가상 클래스는 데이터베이스의 상태에 기반하여 정확한 회원 자격을 판별하는 회원 자격 유도 함수가 관련되어 있다. 가상 클래스의 외연은 일반적으로 명시적으로 저장되지 않고 필요시에 계산된다.

3.3.1 가상 클래스

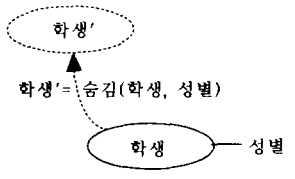
우리 뷰 모델의 객체 대수(Object Algebra)는 문헌[3, 6, 8, 13, 17]에서 발견되는 객체-지향 뷰뿐만 아니라 SQL 뷰도 포함하기 충분할 정도로 일반적이다. 우리 뷰 모델에서 지원되는 객체 대수 연산자는 *선택*(select), *숨김*(hide), *정련*(refine), *합집합*(union), *교집합*(intersection), *차집합*(difference)은 아래에 자세히 정의되어 있다[11].

① C = *선택*$(A, pred)$, A는 클래스이고 *pred*는 조건일 때, 은 클래스 C를 돌려준다. C의 외연은 조건 *pred*를 만족시키는 A의 모든 객체들로 구성되어 있고 C의 타입은 A의 것과 같다. (그림 2-a)를 보면 *남학생* 가상 클래스가 *학생* 클래스로부터 *선택* 연산자를 사용하여 생성됨을 보여준다.

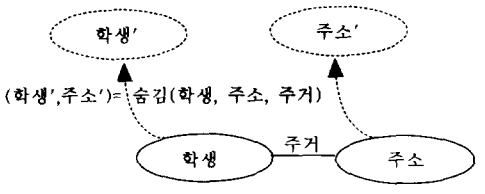
② C = *숨김*$(A, attr)$, A는 클래스이고 *attr*은 숨겨지는 속성일 때, 는 클래스 C를 돌려준다. C의 타입은 *attr* 속성을 제외한 모든 A의 성질을 포함하며, C의 인스턴스는 A에 속한 인스턴스들이다. 변화를 주면, 우리는 클래스 C_1과 C_2간의 관계 r을 숨기는 대수 연산자를 정의할 수 있고, 이는 *숨김*(C_1, C_2, r)로 표시된다. 출력 가상 클래스 C_1'은 C_1의 슈퍼클래스이나 C_2'은 C_2와 isa 관계에 있지 않는다. (그림 2-b)는 *학생'* 가상 클래스가 *학생* 클래스로부터 성별 속성을 *숨긴*다는 것을 알 수 있다. (그림 2-c)는 *학생'*과 *주소'* 클래스가 *학생*과 *주소* 클래스 사이의 *주거* 릴레이션쉽을 *숨김*으로써 생성됨을 보여준다.



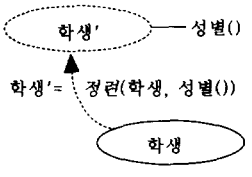
(a) 선택 연산자에 의한 가상 클래스 생성



(b) 숨김 연산자에 의한 가상 클래스 생성



(c) 릴레이션쉽 숨김 연산자에 의한 가상 클래스 생성



(d) 정련 연산자에 의한 가상 클래스 생성

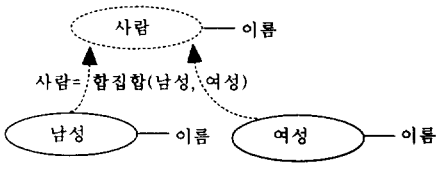
(그림 2) 선택, 숨김, 정련 연산자에 의한 가상 클래스 생성

③ C = *정련*(A, m), A는 클래스이고 m은 새롭게 정의된 메소드일 때, 는 출력 클래스 C를 돌려준다. C의 타입은 A의 것에 정련하는 메소드 m을 더한다. A와 C의 클래스는 그 들의 인스턴스로써 같은 객체 집합을 갖는다. (그림 2-d)는 *학생'* 클래스는 *학생* 클래스를 *성별()* 메소드(method)로 *정련*하여 생성됨을 보여준다.

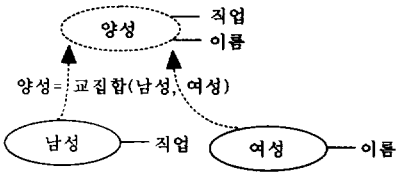
④ C = *합집합*(A, B), A와 B는 클래스일 경우, 은 클래스 C를 돌려준다. C의 타입은 입력 타입의 가장 작은 공통 슈퍼타입이며, 외연은 입력 외연들의 합집합이다. (그림 3-a)는 *사람* 클래스가 *남성*과 *여성* 클래스의 합집합으로 정의됨을 보여준다. 이때 *사람* 합집합 클래스는 *남성*과 *여성* 클래스의 공통 속성 *이름*을 타입으로 가진다.

⑤ C = *교집합*(A, B), A와 B는 클래스일 경우, 는 클래스 C를 돌려준다. C의 타입은 입력 타입의 최대 공통 서브타입이며, 외연은 입력 외연들의 교집합이다. (그림 3-b)는 *양성* 가상 클래스가 *남성*과 *여성* 클래스의 *교집합*으

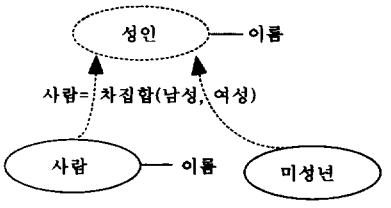
로 정의됨을 보여준다. 이때 *양성* 가상 클래스는 *남성*과 *여성* 클래스의 속성 *직업*과 *이름* 전부를 타입으로 갖는다는 것을 보여준다.



(a) 합집합 연산자에 의한 가상 클래스 생성



(b) 교집합 연산자에 의한 가상 클래스 생성



(c) 차집합 연산자에 의한 가상 클래스 생성

(그림 3) 합집합, 교집합, 차집합 연산자에 의한 가상 클래스 생성

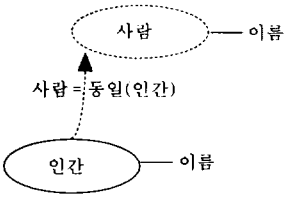
⑥ *C= 차집합(A, B)*, *A*와 *B*는 클래스일 경우, 는 클래스 *C*를 돌려준다. *C*의 타입은 첫번째 변수 클래스 *A*의 것과 같으며, 외연은 *A*의 모든 인스턴스에서 *B*의 인스턴스를 뺀 것에 해당하는 *A* 외연의 부분집합이다. (그림 3-c)는 *성인* 가상 클래스는 *사람* 클래스와 *미성년* 클래스의 *차집합*으로 정의됨을 보여준다.

이상이 멀티뷰[11]의 모델에서 제공하는 객체 대수 연산자이고 우리의 뷰 모델은 더 나아가 다음의 연산자들을 더 제공한다.

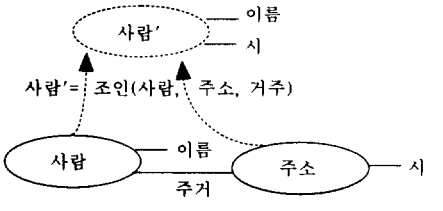
⑦ *C= 동일(A)*, *A*는 클래스일 경우, 는 클래스 *C*를 돌려준다. *C*의 타입과 외연은 *A*의 그 것들과 같다. (그림 4-a)는 *사람* 가상 클래스가 *인간* 클래스로부터 *동일* 연산자를 사용하여 생성됨을 보여준다.

⑧ *C= 조인(C1, C2, r)*, *C1*과 *C2*는 클래스이고 *r*은 관계일 경우, 는 클래스 *C*를 반환한다. *C*의 타입은 관계 *r*을 제외하고 *C1*과 *C2*의 성질들을 포함한다. *C*의 외연은 $\{o | o_j = (o_1, o_2)$, o_1과 o_2는 각각 *C1*과 *C2*에 속하고 $\exists r(o1,o2)\}$이다. 속성 *b*가 *C*에 정의되었다고 하면 O_j의 *b* 값은, $o_j \rightarrow b$로 표시, $o_1 \rightarrow b$ (*b*가 *C1*에 정의)이거나 또는 $o_2 \rightarrow b$ (*b*가 *C2*에 정의) 이다. 만약 *b*가 *C1*과 *C2* 클래스 양쪽에 정의되어 있다면, $o_j \rightarrow b$는 $o1 \rightarrow b$와 같다, 즉, 조인 객체 o_j의 왼쪽 원천 객체 o_1이 오른쪽 원천 객체 o_2에 비해 높은 우선 순위를 가진다. (그림 4-b)는 *사람'* 가상 클래스가 사람과 주소 클래스를 *주거* 릴레이션쉽을 기반하여 *조인*함으로써 생성됨을 보여준다. *사람'* 가상 클래스는 원천 클래스인 *사람*과 주소 클래스들의 속성 *이름*과 *시*를 전부 가짐을 알 수 있다.

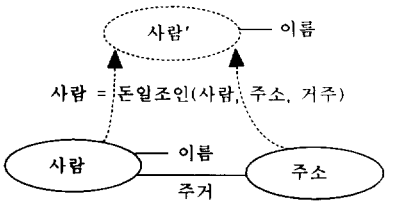
⑨ *C= 동일조인(C1, C2, r)*, *C1*과 *C2*는 클래스이고 *r*은 *C1*의 서브클래스(*C1* 포함)와 *C2*간에 정의, 는 클래스 *C*를 돌려준다. 이는 다음과 같은 점을 제외하고는 *조인(C1, C2, r)*과 같다. 첫째, 동일조인 클래스 *C*의 타입은 첫 번째 변수 클래스 *C1*의 것과 같다 - 조인 클래스의 타입은 두 변수 클래스의 성질들의 합집합이다. 둘째, 첫 번째 변수 클래스 *C1*의 직/간접 인스턴스들은 *C2* 인스턴스들과 *r*로 연결되지 않으면 이들은 동일조인 클래스에 속한다. (그림 4-c)는 *사람'* 가상 클래스가 *사람*과 주소 클래스를 *동일조인*하여 생성됨을 알 수 있다.



(a) 동일 연산자에 의한 가상 클래스 생성



(b) 조인 연산자에 의한 가상 클래스 생성

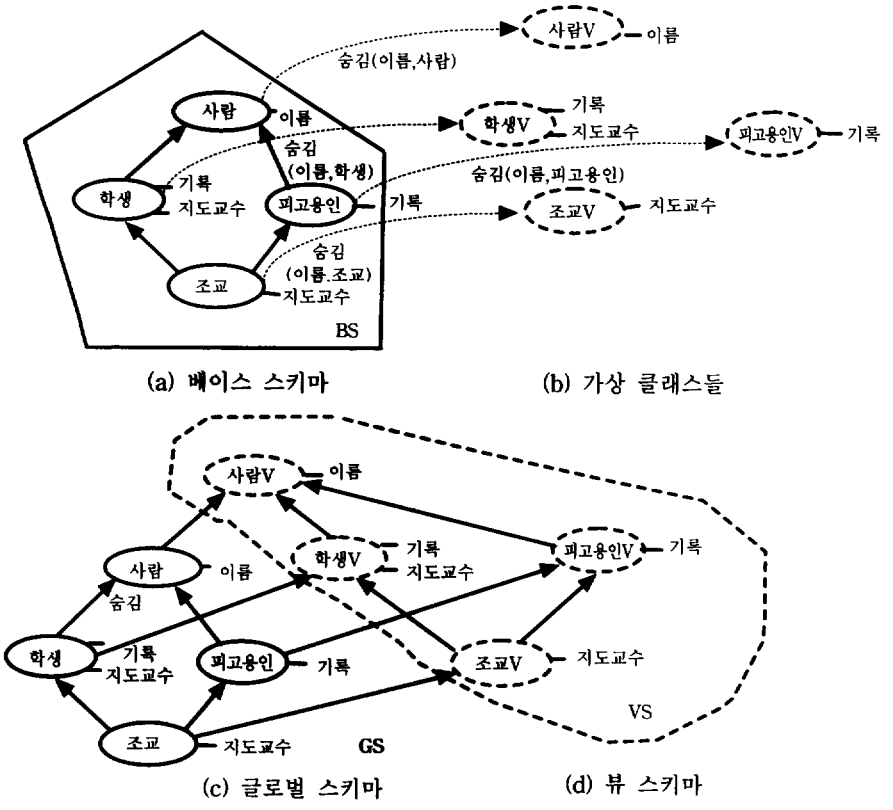


(c) 동일조인 연산자에 의한 가상 클래스 생성

(그림 4) 동일, 조인, 동일조인 연산자에 의한 가상 클래스 생성

3.3.2 뷰 스키마 (View Schema)

(그림 5-a)는 사람, 학생, 피고용인, 조교의 베이스 클래스를 포함하는 베이스 스키마 BS를 보여준다. 그러면 우리는 (그림 5-b)에서 보듯이 사람V, 학생V, 피고용인V, 조교V



(그림 5) 베이스 스키마(BS), 글로벌 스키마(GS), 뷰 스키마(VS)

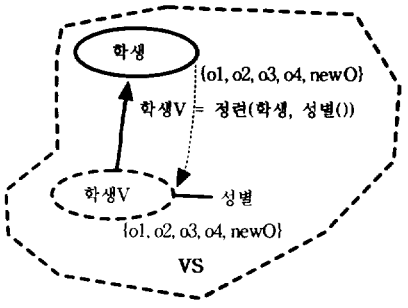
가상 클래스를 가질 수 있다 - 이때 이들 가상 클래스는 사람, 학생, 피고용인, 조교 베이스 클래스 각각으로부터 속성 이름[사람]을 숨긴다. (그림 5-c)에서 보듯이 우리가 선정한 CAD에 특화된 객체-지향 뷰인 MultiView는 글로벌 스키마(Global Schema)를 도입한다. 글로벌 스키마 GS = (GV, GE)는 베이스 스키마 BS와 정의된 모든 가상 클래스를 포함한다. 이때 클래스 사이의 isa 관계인 에지는 베이스 스키마 정의된 에지들에 기반하여 자동으로 생성될 수 있다[11, 12]. (그림 5-d)에서 보듯이 우리는 글로벌 스키마에서 원하는 클래스들만 선택하여 뷰 스키마를 정의한다. 이때 뷰 스키마의 클래스들 사이의 에지는 글로벌 스키마에 정의된 에지에 기반하여 자동으로 생성된다[13].

4. 뷰 스키마의 갱신 의미(Semantic)

데이터베이스 뷰를 사용하여 CAD와 같은 공학 응용 프로그램의 데이터 인터페이스(interface)를 만들려는 우리의 목표에 의하면, 뷰 스키마는 사용자에게 베이스 스키마로 보여져야 하기 때문에 베이스 스키마와 같은 성질(property)을 가져야한다. 그렇지 않다면 베이스 스키마에서 개발된 공학 응용 프로그램이 객체지향 뷰에서 동작할 때 비정상적인 행동을 보일 수 있다. 이는 뷰 스키마가 베이스라면 허용되어야 할 모든 질의와 갱신가 똑같이 뷰에서 허용되어야함을 의미한다. 그러나, 뷰에 대한 갱신이 베이스 스키마에 대한 갱신으로 변환될 때 모호한 의미를 가진다는 것은 잘 알려진 사실이다. 이 문제는 뷰 갱신 문제[19]로 알려져 있다. 이 문제를 해결하기 위해, 우리는 뷰 갱신 의미를 다음과 같이 정의하여 한다 : (1) 뷰에 대한 갱신은 모호함 없이 기저의 베이스 스키마에 대한 갱신으로 번역된다 ; (2) 기저의 베이스 스키마로부터 뷰에 대한 (2)기저의 베이스 스키마로부터 뷰에 대한 재-전파는 그 갱신가 뷰에 직접 수행되는 것과 동일한 효과를 가진다. 이러한 갱신 의미가 이 장에서 공식화되고 그 성질들이 조사된다.

4.1 소 개

뷰 갱신 의미(semanctics)를 논의하기 위해 (그림 6)의 간단한 뷰 스키마 VS를 고려해보자. 우리가 생성 갱신을 사용하여 학생의 새로운 인스턴스 newO를 생성하였다 하자. 그러면 정련 객체 대수의 정의에 따라 newO는 자동으로 학생V의 인스턴스가 된다. 만약 VS가 공학 응용 프로그램의 데이터 인터페이스로 사용된다면 그 프로그램은 그 인터페이스를 더 이상 통상적인 베이스 스키마로 볼 수는 없을 것이다. 왜냐하면 상위 클래스(*학생*)에 새 인스턴스(newO)를 생성하면 이 인스턴스가 하위 클래스(*학생V*)에서 보이는 베이스 스키마에서 발생할 수 없는 기현상이 나타나기 때문이다.



(그림 6) 간단한 뷰 스키마

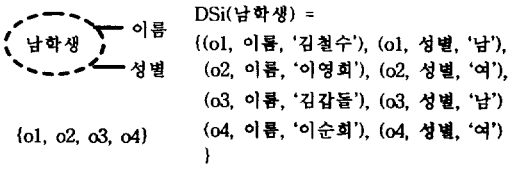
1장에서 언급한대로 공학 응용 프로그램은 데이터베이스를 활용함으로써 생산성을 높일 수 있다. 그리고 공학 응용 프로그램의 다양한 데이터 요구를 만족시키기 위해 뷰를 생성한다. 그런데 대부분의 공학 응용 프로그램은 그 데이터 인터페이스로 베이스 스키마를 상정하기 때문에 뷰에 대하여 동작할 경우 오류를 발생시킬 수 있다. 이러한 오류를 방지하기 위해서는 뷰에 대한 갱신이 마치 베이스 스키마에 대한 것처럼 보여야 한다. 즉, 뷰 갱신 의미(semantic)이 베이스 스키마의 갱신 의미(semantics)을 보존하여야 한다.

4.2 이 론

우리는 가상 클래스 하(下)의 저장 데이터베이스에 대한 정의를 내린다.

[정의 1] *VC*가 가상 클래스라 하자. 그러면, $DB_i(VC)$는 두 집합에 의해 정의된다. 하나는 시각 i 에 *VC*에 속한 인스턴스들의 *OID* 들의 집합, $ODS_i(VS)$로 표시, 이며 또 하나는 시각 i의 *VC*의 데이터 집합으로 $DS_i(VC) = \{(o, attr, v) |$ 시각 i의 $o \in ODS_i(VC)$, $attr \in type(VC)$, 그리고 $v = o \rightarrow attr\}$로 표시된다.

(그림 7)에서 가상 클래스 *남학생*의 속성은 *이름*, *성별*이다. 이때 시각 i의 VC의 인스턴스는 {o1, o2, o3, o4}이라 하자. 그러면 이 클래스의 상태는 (그림 7)의 오른쪽에 보이는 것과 같다.



(그림 7) 가상 클래스의 상태

위의 정의를 바탕으로, 가상 클래스 *VC*의 세 가지 고유 갱신의 의미 보존(update semantic preserving : usp) 특성이 정의된다.

[정의 2] *VC*가 가상 클래스라 한다. 그러면 *VC*의 고유의 갱신가 다음과 같이 정의될 때만 *VC*를 usp라 한다:

- *생성(VC)*는 새 객체 o, $o \in ODS_{i+1}$를 돌려준다. 그 부대효과로 $ODS_{i+1}(VC) = ODS_i(VC) \cup \{o\}$ 이고 $DS_{i+1}(VC) = DS_i(VC) \cup \{(o, attr, nil) | attr \in type(VC)\}$이다.
- *제거(o)*, $o \in ODS(VC)$, 는 *nil*을 돌려준다. 부대효과로 $ODS_{i+1}(VC) = ODS_i(VC) - \{o\}$ 이고 $DS_{i+1}(VC) = DS_i(VC) - \{(o, attr, v) |$ 시각 i의 $attr \in type(VC)$ 이고 $v = o \rightarrow attr\}$ 이다.
- *갱신(o, attr, newValue)*, $o \in DS(VC)$, $attr \in type(VC)$, 그리고 $newValue \in domain(attr)$, 은 o를 돌려준다.

위 정의의 첫번째 조건은 어떤 클래스 인스턴스를 생성하면 객체 식별자 (object identifier) 'o'와 각 속성 값들을 저장할 장소 '(o, attr, nil)'가 확보됨을 의미한다. 두번째 조건은 어떤 클래스 인스턴스를 제거하면 그 객체 식별자 'o'와 속성 값 저장소 '(o, attr, nil)'가 함께 삭제됨을 의미한다. 세 번째 조건은 어떤 인스턴스 'o'의 속성 'attr'의 값을 'newValue'로 변경하면 그 속성 값 저장소가 '(o, attr, newValue)'로 됨을 의미한다.

[정의 3] VC_1 과 VC_2 둘 다 가상 클래스라 하자. 그러면 다음 조건이 만족될 때(VC_1, VC_2) 짝을 usp 뷰 형성 가능이라 한다.

- VC_1이 VC_2의 하위 클래스이면, 시각 i에 VC_1에 적용되는 생성 오퍼레이션은 그것이 $DB_i(VC_1)$를 변경하는 것과 같은 방법으로 $DB_i(VC_2)$를 변경한다. VC_1이 VC_2의 하위 클래스가 아니면, VC_1에 적용되는 생성은 $DB_i(VC_2)$를 바꾸지 않는다.
- 인스턴스 o, $o \in ODS(VC_1)$, 의 $attr$에 적용되는 갱신 갱신는, 속성 $attr$이 VC_2에서 정의되고 인스턴스 $o \in ODS(VC_2)$이면, $DB_i(VC_2)$를 $DB_i(VC_1)$ 바꾸는 것과 같이 바꾼다. 그렇지 않으면, 즉, attr이 VC2에서 정의되지 않거나 o가 $ODS(VC_2)$에 속하지 않으면, $DB_i(VC_2)$는 변하지 않는다.
- 인스턴스 o, $o \in ODS(VC_1)$, 에 적용되는 제거 갱신는 $o \in ODS_i(VC_2)$이면 $DB_i(VC_2)$를 $DB_i(VC_1)$이 바뀌는 것과 똑같이 바꾼다. 그렇지 않다면, $DB_i(VC_2)$가 변하지 않는다.

위 정의의 첫번째 조건은 한 클래스 VC_1의 인스턴스를 생성하면 그 인스턴스는 단지 그 하위 클래스 VC_2에서만 보인다는 것을 의미한다. 두번째 조건은 한 인스턴스 o의 $attr$ 값을 바꾸면 이 변화가 하위 클래스에서도 보인다는 것을 의미한다. 세번째 조건은 어떤 인스턴스 o를 클래스 VC1에서 제거하면 이 인스턴스는 하위 클래스 VC_2에서도 제거됨을 의미한다.

VC_1이 VC_2와 usp 뷰 형성 가능을 $usp(VC_1, VC_2)$로 표시하기도 한다. 클래스 VC_1과 VC_2가 뷰 형성가능이면 두

클래스로 구성된 뷰 스키마는 항상 갱신 의미 보존이다. 즉, 이 뷰 스키마에 대한 갱신은 베이스 스키마에 대한 갱신과 동일한 행동을 보인다.

생성, 갱신, 제거와 같이 usp 뷰 형성 가능에는 세 가지 조건이 있지만, 대부분의 경우 단지 첫번째 조건만 만족시키면 충분하다. 이는 다음 정리에서 서술된다.

[정리 1] 가상 클래스 VC_1과 VC_2가 두 개의 (가상) 클래스 집합 $K_1 = \{C_{11}, C_{21}, C_{31}, \cdots, C_{n1}\}$ 과 $K_2 = \{C_{12}, C_{22}, \cdots, C_{m2}\}$ 각각에서 유도되었다고 하자. 그러면, K_1과 K_2의 (가상) 클래스들이 서로 usp 뷰 형성 가능 관계이며 K_1과 K_2의 (가상) 클래스로부터의 VC_1과 VC_2를 유도하는 함수 각각이 조인이나 동일조인을 포함하지 않으면, VC_1과 VC_2는 usp 뷰 형성 가능이다.

증명 : 인스턴스 o, $o \in ODS(VC_1)$, 의 $attr$ 값을 v로 갱신하고 o가 $ODS(VC_2)$에도 속하고 $attr$이 VC_2에도 정의되었다고 하자. 그러면 o는 적어도 하나의 클래스 $C_{i1}, C_{i1} \in K_1$ 이고 $attr$이 C_{i1}에서 정의, 에 속한다. 이는 왜냐하면 VC_1의 유도 함수가 조인이나 동일조인을 포함하지 않아 새로운 객체를 생성하지는 않기 때문이다. 비슷한 이유로 o는 $attr$이 정의된 적어도 하나의 클래스 C_{j2}, $C_{j2} \in K_2$, 에 속하여야 한다.

*갱신(o, attr, v)*는 $DB(C_{i1})$의 상태를 바꿔서 $o \rightarrow attr = v$가 되게 한다. 왜냐하면 $ODS(C_{i1})$과 C_{i1}이 $attr$ 속성을 가지기 때문이다. 그러면 이 데이터베이스의 변화는 C_{j2}의 데이터베이스를 같은 방법으로 바꾼다. 왜냐하면 C_{i1}은 C_{j2}와 usp 뷰 형성 가능이기 때문이다. 다시 이 데이터베이스 변화는 VC_2의 데이터베이스를 같은 방법으로 바꾼다. 왜냐하면 $o \in VC_2$ 이고 $attr$이 VC_2에 정의되어 있기 때문이다. 이로서 위의 usp 뷰 형성가능 조건 중 갱신 부분이 만족됐다. 제거 갱신는 데이터베이스에서 객체를 제거하는 글로벌(global) 오퍼레이션이다 -- *제거(o)*는 인스턴스 o를 $o \in ODS(C)$인 어떤 클래스 C로부터도 제거한다. 이는 usp 뷰 형성가능의 조건중 세 번째 부분 즉, 제거 갱신에 관한 조건을 만족한다. 이로써 우리는 첫째 조건이 만족되고 (이 정리의 조건) 조인이나 동일조인이 관련되지 않으면 둘째와 셋째 조건이 자동으로 만족함을 보였다.

위 정리에 의하면 갱신 의미 보존 스키마의 임의의 클래스로부터 가상 클래스 VC_1과 VC_2를 유도하면 - 조인이나 동일조인을 사용하지 않고 - 이 두 클래스는 usp 뷰 형성 가능임을 알 수 있다.

[정리 2] VC_1과 VC_2가 가상 클래스라 하자. (VC_1, VC_2)가 usp 뷰 형성 가능이면, (VC_2, VC_1) 역시 usp 뷰 형성 가능이다 (교환법칙).

증명 : 정리 10의 뷰 형성 가능의 세가지 조건은 대칭이다. 즉, 조건을 기술한 문장에서 VC1과 VC2를 바꾸어도 여전히 옳다. 그러므로 usp(VC1, VC2)이면 usp(VC2, VC1)이다.

[정리 3] VC_1, VC_2, VC_3가 가상 클래스라 하자. (VC_1, VC_2)과 (VC_2, VC_3)가 usp 뷰 형성 가능이고 (VC_1 isa VC_2) 이고 (VC_2 isa VC_3) 이면, (VC_1, VC_3) 역시 usp 뷰 형성 가능이다 (이행성 법칙).

증명 : VC1에 대한 생성 연산은 VC2에 전파되고 결국 VC3에 전파된다. VC3에 대한 생성은 DB(VC2)에 영향을 미칠 수 없다. 왜냐하면 VC3가 VC2의 상위 클래스이고 서로 usp 뷰 형성 가능이기 때문이다. 그리고 ODS(VC1)은 새롭게 형성된 객체를 가질 수 없다. 왜냐하면, 그렇지 않으면, ODS(VC2)가 영향을 받기 때문이다. 이는 VC2가 VC1의 상위클래스이고 서로 usp 뷰 형성 가능이라는 것을 고려하면 자명하다.

ODS(VC1)의 객체 o 의 attr 속성에 적용되는 갱신연산은 DB(VC1)이 변화하는 것과 동일하게 ODS(VC2)에 영향을 준다. 이는 VC1과 VC2가 usp 뷰 형성 가능이기 때문이다. 그러면 DB(VC3)는 DB(VC2)가 이 갱신 연산에 영향을 받는 것과 동일하게 변한다. 이는 VC2와 VC3가 usp 뷰 형성 가능이기 때문이다. 사실, 갱신에 대한 usp 뷰 형성 가능 조건은 VC1, VC2, VC3 사이의 isa 관계에 무관하게 이행성이 성립된다.

제거 연산에 관해서는 이 정리는 자명하다. 이는 객체의 제거는 글로벌 연산이기 때문이다.

[정리 4] VC_1, VC_2, VC_3는 가상 클래스이다. (VC_1, VC_2)와 (VC_2, VC_3)가 usp 뷰 형성 가능이고 (VC_1 isa VC_2), (VC_3 isa VC_2), VC_1이 VC_3와 isa 관계가 없으면, (VC_1, VC_3) 역시 usp 뷰 형성 가능이다.

증명 : VC1에 대한 생성은 VC1이 VC2와 usp 뷰 형성 가능이고 VC1이 VC2의 하위 클래스이므로 VC2에 전파된다. 그러나 VC2가 VC3와 뷰 형성 가능이고 VC3가 VC2의 상위클래스이므로 위의 생성 연산은 VC3에 전파되지 않는다. 그러므로 (VC1, VC3)은 정리 10의 뷰 형성 가능 첫번째 조건을 만족시킨다. 이는 DB(VC3)이 VC1에 대한 생성 연산에 영향을 받지 않기 때문이다. VC3와 VC1은 대칭이기 때문에 동일한 논의가 VC3에 대한 생성 연산에 적용된다. 정리 10의 두 번째 조건은 VC1, VC2, VC3 사이의 isa 관계에 무관하게 이행성이다. 세번째 조건은 VC1이 VC3와 뷰 형성 가능이든 아니든 관계없이 항상 성립한다.

[정의 4] 뷰 스키마 VS가 usp 클래스 $\{VC_1, VC_2, \cdots, VC_n\}$로 구성되어 있다고 하자. 그러면 뷰 스키마 VS가 뷰 형성 가능 스키마라는 것은 VS 클래스의 모든 페어(pair) p, $p \in VV$, 가 usp 뷰 형성 가능이다 는 걸로 정의된다.

*VS*의 모든 클래스가 usp이기 때문에, VC에 대한 생성, 갱신, 제거는 정의 9에 의해서 베이스 클래스에 대한 갱신와 같은 의미를 가진다. 더구나, 정의 11에 의하면 모든 폐어(VC_1, VC_2)는 usp 뷰 형성 가능이다. 이는 VC_1에 대한 모든 변경은 정의 9에 따라 $DB_i(VC_1)$을 변화하고 정의 10에 따라 *VS*의 모든 다른 클래스 C_2의 상태, $DB_i(VC_2)$,에 영향을 준다. 정의 9와 10에 의해 정의된 갱신 의미가 베이스 스키마의 갱신 의미을 보존한다는 것을 고려하면, 다음의 정의와 주장이 가능하다.

[정의 5] *VS*를 스키마라 하자. 그러면 $DB(VS)$로 표현되는 *VS*의 데이터베이스는 *VS*의 모든 클래스 VC_i의 데이터베이스, $DB(VC_i)$, 의 합집합으로 정의된다.

주장 1 : *VS*가 usp 뷰 스키마이고 $DB_i(VS)$가 시각 *i*에 *VS* 하에 저장된 데이터베이스라 하자. 그러면 $DB_i(VS)$에 대한 임의의 갱신는 시각 *i+1*의 스냅샷(snapshot) $DB_{i+1}(VS)$을 결과하는데, 이 결과는 만약 *VS*가 베이스 스키마라 했을 경우의 결과와 같다.

위의 주장은 *VS*가 usp이면 $DB_i(VS)$에 대한 갱신가 기저의 스키마 *S*의 데이터베이스 $DB_i(S)$로 번역되고 이것이 *S*에서 *VS*로 재-전파되어 *VS*의 데이터베이스가 $DB_{i+1}(VS)$로 이전할 때, 이는 만약 *VS*가 베이스 스키마여서 그 갱신가 *VS*에 직접 실행 됐을 때의 결과와 같다는 것을 의미한다. usp 뷰에 대한 비슷한 정의가[21]에서 발견된다.

[정의 7] 스키마 *S1*에서 *S2*로의 매핑으로 표현되는 뷰 유도 함수 $VD_{S2,S1}$ 는 *S1*이 usp 이기만 하면 *S2*가 usp라는 것을 보장하면 usp라 불린다.

[정리 5] 뷰 유도 함수 VD_1과 VD_2를 가정하자. 그러면 이 뷰 유도 함수의 결합 $VD_2 \cdot VD_1$은 역시 usp 이다. 이때 · 은 $VD_2 \cdot VD_1(S) = VD_2(VD_1(S))$와 같이 연속적인 함수 적용으로 정의된다.

증명 : 뷰 $VS_1 (= VD_1(S))$은 *S*가 usp이면 정의 13에 의해서 usp이다. 그리고, 뷰 $VS_2 (= VD_2(VS_1))$는 VD_2가 usp 유도 함수이고 VS_1이 usp 임으로 역시 usp이다. 그러므로 함수 결합 $VD_2 \cdot VD_1$은 usp 뷰 유도 함수이다.

5. 갱신 의미 보존 뷰 모델

우리는 갱신 의미 보존을 위하여 모든 뷰 유도는 단지 6가지 뷰 연산자만 요구하도록 우리의 뷰 모델을 제한한다 : *숨김(hide), 합집합(union), 차(difference), 동일 (ident), 조인(join), 동일조인(identicalJoin)* 이다. 그리고, 우리는 이 여섯 가지 가상 클래스의 갱신 의미를 베이스 클래스의 갱신 의미가 보존되도록 정의한다. 각 가상 클래스의 갱신이 정의될 때 우리는 이 가상 클래스들이 기존의 클래스(원천 클래스 포함)와 usp 뷰 형성 가능인 조건을 함께 도출한다. 그러므로 우리가 뷰 스키마 *VS*를 생성할 때 (1) 위의 6가지 뷰 연산자만 사용하고 ; (2) 생성된 가상 클래스의 갱신이 아래에서처럼 정의되고 ; (3) 가상 클래스와 그 변수 클래스(들)는 서로 usp 뷰 형성 가능일 경우만 한 스키마에 포함되면 *VS*는 갱신 의미 보존임이 보장된다.

5.1 숨김 가상 클래스

숨김 가상 클래스는 *숨김* 연산자에 의해 생성된다. 이는 그 변수 클래스의 상위 클래스이며 타입이 바뀐다.

갱신 의미 정의 이 *숨김* 클래스에 적용되는 *생성* 갱신는 이 것이 마치 변수 클래스에 적용되는 것처럼 동작한다. 그러면, 이는 다시 가상(*숨김*) 클래스가 베이스 클래스인 것인 것과 같은 효과를 낸다. 이는 왜냐하면 생성된 객체가 *숨김* 클래스의 새로운 인스턴스로 나타날 것이기 때문이다. *제거*와 *갱신* 오퍼레이션은 직접적인 방법으로 원천 객체에 전파된다. 이는 오퍼레이션이 숨김 클래스의 객체에 직접 작용하는 것과 똑같은 효과를 낼 것이다. 그러므로 갱신가 위에서처럼 정의된 숨김 클래스는 정의 2에 의하여 usp 가상 클래스이다.

변수 클래스와 usp 뷰 형성 가능 그러나, *숨김* 클래스는 그것의 변수 클래스와 뷰 형성 가능이 아니다. 이는 *숨김* 클래스의 인스턴스 생성은 그 변수 클래스에서 보이고, 이는 정의 3의 usp 뷰 형성 가능 정의를 위반하기 때문이다.

5.2 차집합 가상 클래스

차집합 가상 클래스는 첫번째 변수 클래스의 하위 클래스에 위치하며, 첫번째 변수 클래스의 모든 객체 중에서 두번째 변수 클래스의 멤버(member)가 아닌 모든 인스턴스를 포함한다. *차집합* 가상 클래스 인스턴스에 적용되는 *생성, 갱신, 제거* 갱신 오퍼레이션은 첫째 변수 클래스에 전파된다.

갱신 의미 정의 *차집합* 가상 클래스에 대한 *생성*은 첫째 변수 클래스의 새 객체의 *생성*으로 해석된다. 이 새 객체는 두번째 변수 클래스가 usp 뷰 형성 가능이고 두번째 변수 클래스가 첫째 변수 클래스의 상위 클래스가 아니면 차집합 가상 클래스의 인스턴스가 된다. 그렇지 않으면, 새 객체 두번째 변수 클래스에 역시 소속될 것이고 그러면 이는 가상 클래스의 인스턴스가 되지 못한다. 그러므로 우리는 이 경우를 금한다, 즉, *차집합* 클래스의 두 번째 변수 클래스는 첫째 변수 클래스의 상위가 되지 못한다.

차집합 가상 클래스 인스턴스의 제거는 첫째 변수 클래스의 원천 객체에 전파되고 그 원천 객체는 스키마의 모든 클래스에서 제거된다. 이의 효과로, *차집합* 클래스의 인스

턴스는 가상 클래스로부터 직접 지워지는 걸로 보인다. 가상 객체 속성 값의 갱신은 첫째 변수 클래스의 원천 객체에 적용돼도 그 효과는 가상 객체에 반영된다.

변수 클래스와 usp 뷰 형성 가능 그러므로 위 단락의 갱신 의미에 따라, *VC*에 대한 *생성, 갱신, 제거* 갱신는 아무런 부대 효과 없이 그 *차집합* 클래스가 베이스 클래스인 것처럼 실행된다. 즉, 갱신가 위와 같이 정의된 *차집합* 가상 클래스는 정의 2에 의해 usp 가상 클래스이다. 게다가, *차집합* 클래스는 첫째 변수 클래스와 usp 뷰 형성 가능이 아니다. 이는 왜냐하면 첫째 클래스(상위) 인스턴스의 생성이 차집합 클래스(하위)에서 보이기 때문이며, 이는 usp 뷰 형성 가능 정의 10을 위반하기 때문이다.

그러나, 차집합 클래스 C는 정의 3에 의하면 두번째 변수 클래스 C와 usp 뷰 형성 가능이다. 이는 왜냐하면 C의 인스턴스 생성은 C2에 보이지 않으며(C에 대한 생성은 C1에 대한 생성으로 번역되고 이는 C2가 C1의 슈퍼 클래스가 될 수 없기 때문에 C2에서 보일 수가 없다), C/C2 인스턴스의 속성 값 갱신은 서로의 데이터베이스에 영향을 끼치지 않으며(C와 C2는 공통의 속성이 없다), 각 클래스 인스턴스의 제거는 전체 데이터베이스에서 완전한 제거를 낳기 때문이다.

5.3 합집합 가상 클래스

합집합 가상 클래스는 두 변수 클래스의 상위 클래스이다.

갱신 의미 정의 이 경우는 다음의 갱신에서 모호함을 가져온다 : *합집합* 가상 클래스의 객체를 생성시키려면, 이것이 적어도 하나의 원천 클래스에 전파되어야 한다. 일반적으로, 세 가지 선택이 있다. *생성*은 둘 중의 하나 클래스에 또는 둘 다에 전파된다. 우리는 *합집합* 클래스의 생성이 단지 첫째 변수 클래스에 전파되기로 제안한다.

*제거*와 *갱신*의 고유 갱신는 명백한 방법으로 정의된다 : 그들은 항상 양쪽의 변수 클래스에 전파된다. *합집합* 클래스 인스턴스의 제거는 변수 클래스의 원천 인스턴스들에 적용된다. 이는 데이터베이스로부터 원천 인스턴스들을 제거하는 결과를 낳는다. *합집합* 인스턴스 값의 갱신은 변수 클래스에서 보이거나 변수 클래스 인스턴스 값의 갱신은 역시 *합집합* 클래스에 보인다. 그래서 갱신 의미가 위와 같이 정의된 *합집합* 클래스는 정의 2에 의해 usp이다.

변수 클래스와 usp 뷰 형성 가능 *합집합* 클래스는 두번째 변수 클래스와 usp 뷰 형성 가능이다 - 생성이 전파되지 않는 클래스이다. 이는 왜냐하면 두번째 변수 클래스가 첫번째 변수 클래스의 슈퍼 클래스가 아니면 합집합 클래스에 대한 생성은 합집합 클래스의 하위 클래스인 두번째 변수 클래스에 전파되지 않는다. 그래서, 이 경우를 제외한다, 즉, 합집합 클래스의 두번째 변수 클래스는 첫번째 변수 클래스의 슈퍼 클래스가 될 수 없게 하면, 합집합 클래스는 두번째 변수 클래스와 usp 뷰 형성 가능이다.

5.4 동일 가상 클래스

갱신 의미 정의 *동일* 클래스 인스턴스의 생성은 그 변수 클래스 인스턴스 생성으로 정의된다. *동일* 인스턴스의 갱신은 직접적으로 원천 인스턴스에 전파된다. *동일* 클래스의 인스턴스 제거는 전체 데이터베이스에서 완전히 제거되는 결과를 낳는다. 그러면 *동일* 클래스에 대한 *생성, 갱신, 제거*는 그것이 베이스 클래스인 것처럼 실행된다. 그래서 갱신 의미가 위와 같이 정의된 *동일* 클래스는 정의 2에 의해 usp이다.

변수 클래스와 usp 뷰 형성 가능 그러나, *동일* 클래스는 그 변수 클래스와 usp 뷰 형성 가능이 아니다. 이는 우리의 객체 모델이 같은 계층도에서 타입과 외연(extent)이 같으면서 두 개의 서로 다른 클래스를 허용하지 않기 때문이다.

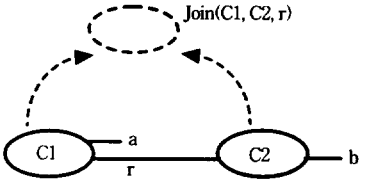
5.5 조인 가상 클래스

갱신 의미 정의 우리는 *조인* 클래스가 usp 가상 클래스가 되기 위하여 *조인* 클래스에 대한 *생성, 갱신, 제거*의 세 가지 고유 갱신를 아래와 같이 정의한다 :

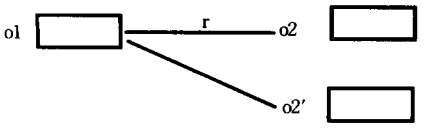
- (그림 8-a)와 같은 *조인(C1, C2, r)* 클래스에 대한 생성은 (그림 8-b)에서 볼 수 있듯이 C_1과 C_2 각각의 클래스에 대해 o_1과 o_2 두 인스턴스의 생성으로 번역되고, o_1은 o_2에 r로 연결된다. 이는 조인 클래스의 새로운 인스턴스 o_j의 생성으로 결과한다.
- 속성 b가 o_1에 정의될 때, 조인 인스턴스 o_j의 b 값을 v로 갱신함은 다음과 같이 번역된다 :

  ① $\exists r(o_1, o_2'), o_2' \neq o_2$, 이면, o_1의 b 값을 v로 갱신한다.
  ② 그렇지 않은 경우, 즉, $\exists r(o_1, o_2'), o_2' \neq o_2$, 이면, C_1의 인스턴스 o_1'을 생성하고 o_1의 값을 o_1'에 복사하고, o_1'의 b 값을 v로 갱신한다. 이는 $r(o_1, o_2')$에 기반한 조인 인스턴스가 이 갱신 오퍼레이션에 의해 의도하지 않게 갱신되는 것을 방지한다. 마지막으로, $r(o_1', o_2)$에 기반한 새로운 조인 인스턴스가 생성되고 그것의 *OID*는 옛 조인 인스턴스 o_j의 *OID*로 만든다. 그래서 조인 인스턴스 o_j의 *OID*는 o_1의 값이 갱신 오퍼레이션에 의해 변할 때 변하지 않게 된다.
  ③ b가 o_2에 정의된 경우, o_1이 o_2로 바뀌는 것 빼고는 위와 똑같은 프로시져가 실행된다. 속성 b가 o_1과 o_2 둘 다에 정의된 경우, 왼쪽 변수 클래스가 더 높은 우선순위를 갖는다. 그래서, 위의 프로시져가 이 경우에 실행된다.

• o_j의 제거는 $r(o_1, o_2)$의 제거로 번역된다.



(a) 조인 클래스 생성



(b) 조인 오브젝트 생성

(그림 8) 조인 클래스의 업데이트 예

위의 갱신 의미에 따르면, *조인*(C_1, C_2, r) 클래스에 대한 *생성* 오퍼레이션은 r 관계에 의해 연결된 C_1과 C_2 클래스 각각에 두 개의 인스턴스를 생성하는 결과를 낳고, 그러므로 *조인* 클래스의 인스턴스 o_j가 생성된다. *조인* 인스턴스 o_j의 속성 값을 갱신함은 원천 인스턴스 중 하나의 값 *갱신*을 초래하고 이 변화는 위에서 정의된 갱신 의미에 따라 단지 해당 *조인* 인스턴스에만 보인다. 그러므로 정의 2에 의해 *조인* 클래스는 usp이다.

변수 클래스와 usp 뷰 형성 가능 *조인*(C_1, C_2, r) 클래스는 원천 클래스 중 어느 것하고도 usp 뷰 형성 가능이 아니다. 이는 *조인* 인스턴스 o_j의 *생성*이 C_1과 C_2 인스턴스의 생성을 초래하기 때문이다, 즉, C_1과 C_2가 조인 클래스 C와 isa 관계에 있지 않음에도 C에 대한 생성이 C_1과 C_2의 데이터베이스에 영향을 끼친다. 이는 usp 뷰 형성 가능 정의 3의 첫번째 조건을 위반한다.

5.6 동일조인 가상 클래스

갱신 의미 정의 *동일조인*(C_1, C_2, r) 클래스에 *생성* 오퍼레이션은 첫째 변수 클래스 C_1의 인스턴스 o_j의 생성으로 번역된다. 그러면 o는 *동일조인*에도 역시 나타날 것이다. *동일조인*(C_1, C_2, r)의 객체 o_j의 속성 b 값을 갱신하면, 이는 o_j가 *조인* 인스턴스가 아니면 o_j에 직접 적용된다. 그렇지 않으면, 즉, o_j가 *조인* 인스턴스이면, r 관계로 연결된 C_1과 C_2의 인스턴스 o_1과 o_2로부터 구성된, *갱신* 갱신는 첫번째 원천 인스턴스 o_1의 b 값을 갱신하거나(o_1이 o_2에만 r로 연결되었을 때) 또한 o_1의 카피(copy)인 o_c 인스턴스의 b 값을 갱신한다 (이때 o_2는 이제 o_1이 아니라 o_c와 r로 연결된다).

동일조인(C_1, C_2, r)의 인스턴스 o의 제거는 o가 *조인* 인스턴스가 아닌 경우 o 인스턴스가 전체 데이터베이스로부터 제거된다. 인스턴스 o가 C_1과 C_2 각각의 인스턴스 o_1과 o_2의 r-연결로부터 생성된 거라 하면 ; (1) $r(o_1, o_2')$인 C_2의 인스턴스 o_2'이 존재하면 o_1의 카피 o_1'이 생성되고 o_1'은 o_2'과 r-연결된다 ; (2) 연결 $r(o_1, o_2)$이 제거된다 ; 그리고 마지막으로 (3) o_1이 제거된다. 이 제거 프로시져는 동일조인 클래스의 인스턴스에 대한 제거가 같은 클래스의 다른 인스턴스로 전파하는 걸 방지한다. 그러므로 갱신 의미가 위에서처럼 정의된 *동일조인* 클래스는 정의 2에 의해 usp이다.

변수 클래스와 usp 뷰 형성 가능 *동일조인*(C_1, C_2, r) 클래스는 원천 클래스 중 어느 것하고도 usp 뷰 형성 가능이 아니다. 이는 왜냐하면 *동일조인* 클래스에 대한 *생성*은 C_1에 대한 *생성*을 초래하고, 이는 usp 뷰 형성 가능 정의 3의 첫번째 조건을 위반하기 때문이다 - 하나의 클래스 *동일조인*(C_1,C_2,r)에 대한 생성이 이 클래스와 isa 관계에 있지 않은 다른 클래스의 데이터베이스에 영향을 미치기 때문이다. 이는 다른 원천 클래스 C_2가 C_1과 r 연결되어 있어 C_1 없이 어떤 스키마에 포함될 수 없기 때문에 C_2도 역시 *동일조인* 클래스와 뷰 형성 가능이 아니라는 것을 의미한다.

6. 결 론

CAD와 같은 공학 응용 프로그램의 데이터 인터페이스를 위해 관계형 뷰를 활용하면 데이터 호환성에 있어 많은 이점이 있다. 그러나 관계형 뷰의 데이터 모델링 능력의 한계와 갱신 모호성으로 인해 제한적으로 사용되어 왔다. 이 두 가지 단점을 객체-지향 뷰에서 극복하기 위해 많은 연구가 되어왔다. 이 논문은 이 중 뷰의 갱신 문제에 접근한다. 공학 응용 프로그램의 데이터 인터페이스 역할을 하기 위한 충분조건으로 뷰가 베이스 스키마의 갱신 의미를 보존해야 한다는 것을 밝히고 이를 공식화한다. 게다가 객체-지향 뷰 연구원들 사이에 잘 알려진 뷰 시스템인 멀티 뷰를 선택하여 베이스 스키마 의미를 보존하는 갱신 행동을 실제로 구현하여 타당성(feasibility)를 증명한다.

참 고 문 헌

[1] E. A. Rundensteiner, "Research initiation award : an object-oriented extensible view system for computer- aided design applications," NSF project summary, 1993.

[2] T. Neuman, "On representing design information in a common database," Proc. of the Engineering Design Applications at the Data Base Week, San Jose, pp.81-87, 1983.

[3] T. C. Chiueh and R. H. Katz, "Intelligent VLSI Design Object Management, EDAC'92, pp.410-414, Feb., 1992.

[4] S. Abiteboul and A. Bonner, "Objects and Views," SIGMOD, pp.238-247, 1991.

[5] O_2 View User O_2 Technology, version 1 edition, December 1993.

[6] S. Heiler and S. B. Zdonik, "Object Views : Extending the Vision," in IEEE International Conference on Data Engineering, pp.86-93, 1990.

[7] Caruso and Sciore, "The VISION Object-Oriented Database System," 1987.

[8] E. A. Rundensteiner, "MultiView : A methodology for supporting multiple views in object-oriented databases," in 18th VLDB Conference, pp.187-198, 1992.

[9] E. A. Rundensteiner, "A classification algorithm for supporting object-oriented views," in International Conference on Information and Knowledge Management, pp.18-25, November 1994.

[10] E. A. Rundensteiner, "Tools for view generation in OODBs," in International Conference on Information and Knowledge Management, pp.635-644, November 1993[8].

[11] M. H. Scholl, C. Laasch, and M. Tresch, "Updatable views in object-oriented databases," in Proceedings of the Second DOOD Conference, December 1991.

[12] M. H. Scholl and H. J. Schek, "Survey of the cocoon project, Objektbanken fur Experten," October 1992.

[13] K. Tanaka, M. Yoshikawa, and K. Ishihara, "Schema virtualization in object-oriented databases," in IEEE Transactions on Knowledge and Data Engineering, Vol.2, No.1, pp.125-142, March 1990.

[14] H. A. Kuno and E. A. Rundensteiner, "The MultiView OODB view system : Design and Implementation," Technical Report CSE-TR-246-95, University of Michigan, Ann Arbor, July 1995.

[15] H. A. Kuno and E. A. Rundensteiner, "Using object-oriented principles to optimize update propagation to materialized views," in IEEE International Conference on Data Engineering, pp.310-317, 1996.

[16] J. Banerjee, W. Kim, H. J. Kim, and H. F. Korth, "Semantics and implementation of schema evolution in object-oriented databases," SIGMOD, pp.311-322, 1987.

[17] U. Dayal, "Queries and views in an object-oriented data model," in International Workshop on Database Programming Language 2, 1989.

[18] W. Kim and H. Chou, "Versions of schema for OODBs," in Proc. 14th VLDB, pp.148-159, 1988.

[19] C. B. Medeiros and F. W. Tompa, "Understanding the implications of view update policies," in International Conference on Very Large Databases, pp.316-323, 1985.

[20] Y. G. Ra, "Transparent Schema Evolution (TSE) Using Object-Oriented View Technology : Transparency Theory, Methodology and System," Ph. D. Dissertation, University of Michigan, 1996.

[21] V. Vidal and M. Winslett, "Preserving update semantics in schema integration," in International Conference on Information and Knowledge Management, pp.263-271, 1994.

나 영 국

e-mail : ygra@hnu.hankyong.ac.kr

1987년 서울대학교 전자공학과 졸업(학사)

1989년 The Penn. State Univ. 컴퓨터공학과 졸업(석사)

1996년 The Univ. of Michigan(Ann Arbor) Comp. Sci. 졸업(박사)

1997년~1999년 SDS 재직

1999년~현재 국립한경대학교 전임강사

관심분야 : 데이터베이스 시스템, 데이터 모델링 방법론