객체지향 데이터베이스의 집단화 관계를 위한 뷰 스키마 생성과 구현

차 현 주† · 윤 종 필††

요 약

본 논문에서는 객체지향 데이터베이스에서 클래스 스키마를 도출하고 도출된 스키마들을 통합하여 하나의 뷰 스키마를 생성하는 방법을 제시한다. 특별히, 객체지향 데이터베이스에서 필요로 하는 part_of 관계에서 적합한 클래스 유도 연산자를 개발하고, 뷰 생성 개념을 프로토타이핑한다. 이와 같은 개념은 다양한 응용 프로그램에 적합하도록 여러 뷰를 생성하므로 데이터베이스에 대한 응용 프로그램들의 효과적인 접근과 관리가 가능하게 된다.

View Schema Generation and Implementation for Part_of Hierarchies in Object-oriented Databases

Hyun-Joo Cha† · Jong-Pil Yoon††

ABSTRACT

This paper describes a method of creating class schemas and generating a view defined over those class schemas. Particularly, new operators essentially needed for the part_of hierarchy in object-oriented databases are proposed and the concept proving of view generation is provided. This concept makes it possible to generate one or more views for various different application programs from object-oriented databases, and therefore those application programs can access and manage databases efficiently.

1. 서 론

관계형 데이터베이스 시스템은 개념이 간단하고 사용하기 용이하여 널리 사용되어 왔다. 그러나 컴퓨터 기술의 발전과 함께 등장한 CAD/CAM, CASE, OIS (office information system)와 같은 새로운 데이터베이스의 응용 분야에서는 기존의 관계형 데이터베이스 시스템이 더 이상 적합하지 않게 되었다. Nested 테이블이 지원되지 않는 등 표현상의 제약성을 해결하기 위하여 1980년대 중반부터 객체지향 개념이 소개되고 그것에 대한 연구가 활발히 진행되어 왔다[2,4,5,11,13,14,15,18].

데이터의 재구성을 지원하기 위해서 관계형 데이터베이스에서처럼 뷰 개념을 객체지향 데이터베이스 시스템에 적용하는 것이 필요하다[3]. 뷰는 각 응용 프로그램과 관련되는 데이터베이스 정보의 일부를 의미하므로 각 응용 프로그램의 요구에 적합한 형태로(customized) 이루어져 있다. 뷰를 이용하기 위해서 뷰에

※ 본 논문은 1996년도 숙명여자대학교 교내연구비 지원에 연구되었음.
† 준 회 원 : 숙명여자대학교 대학원 전산학과
†† 정 회 원 : 숙명여자대학교 전산학과 교수
논문접수 : 1998년 1월 16일, 심사완료 : 1998년 7월 6일

대한 연산이 데이터베이스에 대한 연산으로 변환되어야 하는 부담이 있더라도 데이터베이스 시스템에서 뷰를 지원해야 하는 이유는 (1) 데이터베이스 시스템의 확장성의 제약, (2) 데이터베이스 응용 도메인 재구성의 비효율성과 같은 문제를 뷰 개념을 통하여 효과적으로 해결할 수 있기 때문이다.

지금까지 객체지향 데이터베이스 시스템에서의 뷰에 대한 많은 연구가 있었다[7,9,10,13,16]. 대부분의 연구들이 가상 클래스를 유도하여 뷰를 생성하지만 일반화 관계를 다루었고, 유도된 클래스들을 데이터베이스 스키마에 통합하거나 완전한 뷰 스키마를 생성하는 것에 대한 연구는 미비하였다. 이를 보완하기 위해, 본 논문은 집단화 관계에서의 연산자 개발과 구현을 기술한다. 본 논문의 공헌은 기존 일반화 관계를 수용하여 (1) 새로이 집단화 관계에서의 뷰 생성 방법제시와 (2) 집단화 관계에서 특별히 필요한 새로운 연산자 refer의 개발과, (3) 집단화 관계에서의 클래스 통합과 뷰 생성 방법의 구현이다.

본 논문의 구성은 다음과 같다. 2절에서는 객체지향 모델과 관련된 개념들과 기존의 연구들에 대해 서술한다. 3절에서는 뷰 생성을 위해 필요한 클래스의 유도를 다루고, 유도된 클래스를 클래스 계층구조로 통합하는 것을 설명하며 클래스간의 관계가 일반화 관계와 집단화 관계로 표현될 때의 고려 사항들이 제시된다. 4절에서는 뷰 생성에 대해 살펴본다. 5절에서는 프로토타입을 보인다. 그리고 6절에서 결론을 맺는다.

2. 연구 배경

2.1 기본 개념

객체 지향 모델에서 클래스들의 관계로는 일반화(generalization)와 집단화 관계가 있다. 일반화 관계는 속성과 메소드의 상속이 허락되는 관계로, is_a관계로 표현될 수 있다. 집단화(aggregation) 관계는 여러 개의 구성 요소(component)들이 모여서 복합 객체를 형성하는 경우, 복합 객체와 구성 요소들간의 관계를 말한다. 구성 객체들의 클래스가 단순히 일반 함수에 의해 참조되거나 클래스의 속성의 정의영역이 되는 경우와 구성 객체들이 복합 객체의 필수적인 부분이 되어 의미적으로 전체/부분의 관계를 갖는 경우를 구분하지 않고 모두 집단화 관계라 말할 수 있다.

데이터베이스 시스템의 모델이 변경되거나 데이터베이스 시스템이 확장되어야 하는 경우, 데이터베이스 시스템에서 실행되는 모든 응용 프로그램들은 변화된 데이터베이스에서도 실행될 수 있어야 한다. 이를 위해 기존의 관계형 데이터베이스 시스템은 뷰를 이용하였다.

뷰는 저장된 데이터베이스 테이블이나 또 다른 뷰를 근거로 정의되는 것으로, 연산 표현법과 materialization법[3]이 있다. 본 논문에서 제안하는 객체지향 데이터베이스 뷰는 단순히 필요한 데이터만을 추출하는 것이 아니라 데이터베이스에서 추출한 데이터에 대해 적용할 수 있는 연산을 제한할 수 있고 새로운 연산을 정의해 추가할 수도 있으므로 같은 데이터에 대해 응용 프로그램마다 융통성있게 연산을 지원할 수 있다. 객체지향 데이터베이스 시스템에서의 뷰는 같은 데이터에 대해 다양한 관점을 제공하게 된다[17]. 클래스, 상속등의 객체지향 개념을 고려한다면, 뷰의 의미가 관계형 데이터베이스 시스템보다 객체지향 데이터베이스 시스템에서 더 적합하다고 볼 수 있다.

2.2 관련 연구

지금까지 뷰의 적절한 적용을 위해 관계형 데이터베이스 시스템에서 뷰를 정의하는 것과 관련된 많은 연구가 있었다[1,6,9,10,17].

이미 존재하는 클래스에 대해 또는 유도된 가상 클래스로부터 새로운 가상 클래스를 유도하고[6], 이를 뷰로서 제공할 수 있다[17]. 여러 개의 가상 클래스가 유도될 수 있으며, 뷰 형성에 필요한 데이터베이스의 클래스들을 추출하여 유도된 가상 클래스들과 함께 하나의 뷰를 형성할 수 있다[1,10,11]. 여러 개의 클래스가 뷰에 포함되므로 하나의 가상 클래스로서 정의되는 뷰에서보다 다양한 의미를 표현할 수 있고 뷰 적용 범위가 넓어지게 된다.

O2모델을 기반으로 한 [1]에서는 데이터베이스에 있는 실제 객체(existing object)로 가상 클래스(virtual class)를 populate하거나 실제로 존재하지는 않지만 뷰 내에서만 의미있는 상상의 객체(imaginary object)로 가상 클래스를 populate한다.

[1]로부터 확장된 [11]에서는 가상의 스키마로써 뷰가 정의된다. 가상 클래스는 다른 클래스를 기초로 정의되거나 imaginary class로서 정의될 수 있으며 일반화와 전문화를 통해 다른 클래스에서 가상 클래스를 유도할 수 있다.

[10]에서는 하나의 뷰 스키마로서 뷰가 정의된다. 우선 가상 클래스의 유도에 적용될 수 있는 연산을 제안한 후, 제안된 연산으로 가상 클래스를 유도한다. 클래스들의 일반화 관계는 스키마와 뷰 생성 과정에서 다루어지지만 객체지향에서의 다른 중요한 관계인 part_of 관계가 고려되지 않았다.

3. 가상 클래스 생성

3.1 기본 용어

객체지향 데이터베이스에서 정의된 모든 클래스의 집합을 C라 하자. 객체지향 데이터베이스 스키마 S는 vertices의 유한집합 V와 에지의 유한집합 E로 이루어진 방향성 비순환 그래프이며, S = (V, E)로 표현된다. V의 원소들은 C에 속하고, E는 V×V에 대한 이진관계이다. C1에서 C2로의 방향성 에지 e1 = <C1, C2>은 C1이 C2의 직접적인 하위 클래스라는 것을 의미한다(C1 is_a C2). S에는 지정된 루트 노드가 있다.

데이터베이스의 스키마를 처음 정의할 때 함께 정의되는 클래스를 베이스 클래스(base class)라 하고, 그것의 인스턴스는 기본적인 객체로서 데이터베이스에 저장되어 있다. 객체지향 데이터베이스에 주어지는 질의를 통해 유도되는 클래스를 가상 클래스(virtual class)라 하며, 가상 클래스의 내용은 데이터베이스에 저장되어 있는 것이 아니라 요구에 따라 계산된다. 뷰를 정의하는 과정에서 필요에 의해 가상의 클래스가 유도되는 것이다. 가상 클래스는 뷰의 정의자가 클래스 유도 연산을 통해 직접 유도한 가상 클래스인 명시적 가상 클래스(explicit virtual class)와 스키마의 구조를 유지하면서 가상 클래스를 스키마로 통합하는 과정에서 유도되는 새로운 가상 클래스인 암시적 가상 클래스(implicit virtual class)로 구분할 수 있다.

객체지향 데이터베이스 스키마 S = (V, E)에서, V의 모든 클래스가 베이스 클래스이고 E가 데이터베이스에서 유도할 수 있는 베이스 클래스들간의 관계를 나타낸다면, S는 베이스 스키마(base schema)이다. 기본적인 데이터베이스에 대해 유도된 가상 클래스들과 그들간의 관계를 표현하는 에지가 추가되어 베이스 스키마가 확장된(augmented) 것을 전역 스키마(global schema)라 한다. 유도된 가상 클래스들이 통합된 결과를 전역 스키마에서 볼 수 있다. 전역 스키마 GS = (GV, GE)에 대해 정의되는 뷰 스키마(또는 뷰) VS = (VV, VE)는 다른 뷰와 식별할 수 있는 유일한 식별자를 가진다. VS = (VV, VE)에서, VV는 GV의 부분집합이고, VE는 GE에서 유도할 수 있는 에지들로 이루어진다. 뷰 스키마내의 클래스들을 뷰 클래스(view class)라 하며, 이들은 베이스 클래스일 수도 있고 유도된 가상 클래스일 수도 있다.

3.2 가상 클래스 유도

데이터베이스에 있는 정보를 기초로 각 응용 프로그램에서 요구하는 형식의 뷰를 만들기 위해서는 먼저 새로운 가상 클래스의 유도가 필요하다. 본 절에서는 가상 클래스 유도를 위한 새로운 연산자 Refer가 정의되고, 그 외에 필요한 연산은 [10]에서 소개된 Hide, Refine, Select등의 개념을 이용한다. 기존 연산자들은 is_a관계만 고려하였으므로 복합객체를 표현하는 뷰를 생성하는 데는 한계가 있었다. 이를 해결하기 위해 본 논문에서는 Refer라는 새로운 연산자를 소개한다. Refer라는 연산자는 단순히 is_a관계 뿐만 아니라 part_of의 관계에서 효율적으로 뷰를 생성하기 위하여 필요하다. 클래스 유도 연산의 대상이 되는 클래스를 소스 클래스(source class)라 하며, 이것은 데이터베이스의 클래스이거나 이미 유도된 가상 클래스일 수 있다.

1 Refer연산은 소스 클래스에 대해 새로운 part_of 관계를 정의하여 추가함으로써 가상 클래스를 유도한다. Refer연산은 다음과 같은 구문을 가진다.

class <virtual-class> = refer (<referred-class>) from (<source-class>) with [<part_of-relationship-defs>] <part_of-relationship-defs>에는 연산에서 추가하려고 하는 part_of 관계에 대한 정의가 레이블과 함께 주어진다. <referred-class>에는 참조되는 클래스 이름이 주어지며, 이 클래스는 연산에서 정의되는 part_of 관계에 의해 구성 객체의 클래스가 된다. Refer연산에서 추가된 part_of 관계는 <virtual-class>에서 정의된 part_of관계로서 스키마에 표현된다. <virtual-class>와 <source-class>의 인스턴스 집합은 같다.

3.3 Part_of 관계에서의 가상 클래스 통합

유도된 가상 클래스들이 독립적으로 존재한다면, 다른 클래스들과의 관계를 파악하기 어렵고, 데이터베이스 스키마와 일관성이 유지되는 뷰를 생성하기 곤란하며, 각 응용 프로그램과 데이터베이스의 모델간의 mapping에서 문제가 생길 수 있다. 또한 유도된 가상 클래

스들간의 관계의 연결이 곤란할 수 있으므로 완전한 의미의 뷰(뷰 스키마)를 생성하기가 어렵다. 그러므로 유도된 클래스는 하나의 전역 스키마로 통합되어 스키마내의 다른 클래스들과의 관계를 분명히 하는 것이 바람직하다.

유도된 클래스를 D라 한다면, 전역 스키마로 D를 통합하기 위해서는 먼저 전역 스키마에서 D의 직접적인 상위 클래스들과 하위 클래스들에 해당하는 것을 발견해야 한다. 이를 위해 전역 스키마 전체에 대한 검색이 이루어지게 되고, D의 property 함수들과 인스턴스를 고려하여 검색이 진행된다. 클래스 유도 연산으로 클래스의 유도를 분명히 지정하지 않았더라도 클래스 D의 통합을 위해 필요하다면 적당한 가상 클래스를 생성할 수 있다. 스키마에서 발견된 상위 클래스들이 클래스 D의 직접적인 상위 클래스가 되도록, 발견된 하위 클래스들이 클래스 D의 직접적인 하위 클래스가 되도록 is_a 관계의 에지를 연결한다. 이전에 연결되어 있던 클래스 D의 직접적인 상위 클래스들과 하위 클래스들간의 직접적인 에지는 제거되며 클래스간의 상속 관계를 고려해 각 클래스의 property 함수를 나타낸다[10].

클래스의 통합을 위한 스키마의 검색은 클래스 유도 연산에 따라 스키마의 검색 방식을 다르게 함으로써 보다 효과적으로 수행될 수 있다.

3.3.1 Hide 연산

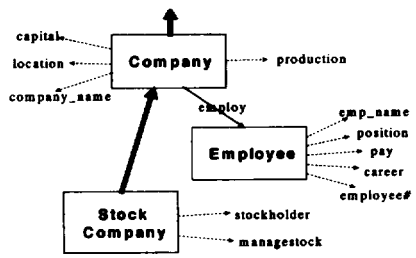
Hide 연산은 연산의 대상이 되는 소스 클래스의 상위 클래스로서 가상 클래스를 유도하는 연산이다.

먼저 소스 클래스에서 정의된 property 함수만을 숨기려는 경우를 고려해 보자. Hide 연산으로 유도된 클래스는 소스 클래스의 직접적인 상위 클래스가 될 수 있고, 기존의 소스 클래스의 직접적인 상위 클래스들이 새로 유도된 클래스의 직접적인 상위 클래스로 된다.

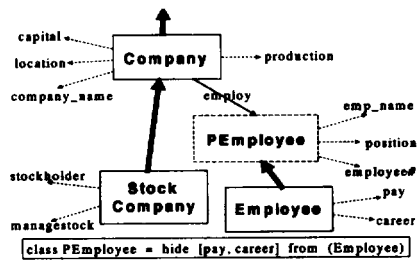
소스 클래스가 상속받은 property함수가 hide 연산에서 숨겨진 경우를 고려해보자. 유도된 클래스의 직접적인 상위 클래스로 적당한 클래스를 발견하기 위해서 계층구조를 따라서 소스 클래스의 직접/간접적인 상위 클래스들을 검색하게 된다. 검색 도중 유도된 클래스의 상위 클래스로서 적당한 클래스를 발견하면 검색은 종료된다. 적절한 상위 클래스를 발견할 때까지 스키마의 검색이 이루어지며, 소스 클래스의 모든 직접/간접적인 상위 클래스들이 검색될 수도 있다. 스키마 검색이 종료된 후에도 유도된 클래스의 직접적인 상위 클래스로 적당한 클래스를 발견하지 못했다면, 적절한 상위 클래스가 있을 것이라고 추측할 수 있는 계층구조상의 위치에 임의의 가상 클래스를 생성하게 된다. 이렇게 생성되는 클래스를 암시적 가상 클래스라 한다. 클래스 통합 과정에서 필요에 의해 생성되는 이런 클래스도 통합되어야 하며, 때로는 연쇄적인 클래스의 생성과 통합이 일어날 수 있다.

유도된 클래스의 상위 클래스를 발견하는 것은 hide 연산에서 숨겨진 property 함수들(속성, 메소드 등), 클래스간의 part_of관계, 클래스들의 상속 등을 고려하여 이루어진다. 각 part_of 관계에 주어진 레이블은 property 함수명처럼 연산에서 사용되어진다.

그림 1은 간단한 예를 들어 part_of관계가 관련된 hide 연산의 실행과 클래스 통합을 보인다. 연산으로 유도된 가상 클래스는 점선의 사각형으로 표현되며 클래스와 property함수의 연결은 점선의 화살표로 나타낸다. 클래스 Company는 복합 객체의 클래스로 Employee의 인스턴스를 구성 객체로 가진다. 그 두 클래스간의 관계는 part_of관계로 가는 실선의 화살표와 레이블 employ로 표현된다.



(a) 연산 실행 전



(b)유도된 클래스 PEmployee의 통합

(그림 1) hide 연산
(Fig.1) hide operation

그림 1.(a)에서 레이블 employ로 표현되었던 Company와 Employee간의 관계는 그림 1.(b)의 hide 연산 실행 후에도 계속 유지될 수 있어야 하고, PEmployee는 Employee의 상위 클래스가 되므로 Company와 유도된 클래스 PEmployee사이에도 employ로 표현된 part_of관계가 유지될 수 있어야 한다. PEmployee와 Company를 레이블 employ인 part_of의 에지로 연결함으로써 Company와 PEmployee간의 part_of 관계가 형성되고, Employee는 상위 클래스인 PEmployee에서 employ로 표현된 관계를 상속받을 수 있으므로 그림 1.(b)와 같이 나타낼 수 있다.

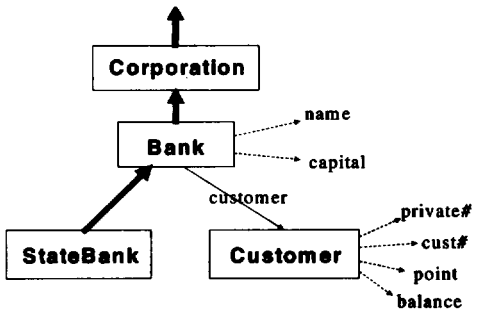
3.3.2 Refine 연산

Refine 연산으로 유도된 클래스는 소스 클래스에서 상속받은 property 함수 외에 연산에서 정의된 새로운 임의의 property 함수를 가지므로 유도된 클래스는 소스 클래스의 하위 클래스가 될 수 있다. Refine연산은 클래스의 인스턴스들에 대해서는 아무런 변화를 주지 않으므로 연산으로 유도된 클래스는 스키마상에서 소스 클래스의 하위 클래스로 통합된다. 소스 클래스가 유도된 클래스의 직접적인 상위 클래스가 되므로 유도된 클래스를 통합하기 위한 스키마의 검색은 불필요하다.

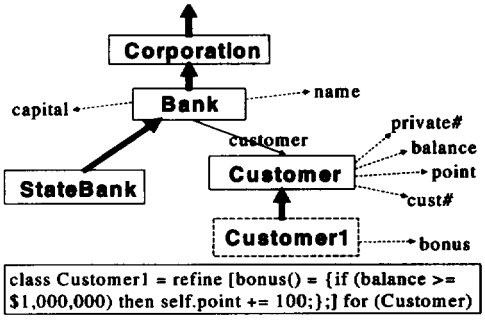
Part_of관계가 refine 연산과 관련되는 경우 유도된 클래스가 어떻게 통합되는지를 그림 2에서 보인다. 소스 클래스 Customer와 연결되어 있던 part_of에지는 refine 연산으로 아무런 영향을 받지 않는다.

3.3.3 Select 연산

Select 연산으로 유도된 클래스는 소스 클래스의 인스턴스들 중 주어진 조건을 만족하는 인스턴스들로 이루어진 클래스이므로 유도된 클래스의 인스턴스들은 소스 클래스의 인스턴스들의 일부이다. 연산의 실행으로 클래스의 property함수들에는 변화가 없지만 인스턴스들의 부분 집합 관계를 고려하면 유도된 클래스는 소스 클래스의 하위 클래스가 된다. 소스 클래스가 유도된 클래스의 직접적인 상위 클래스가 되므로 유도된 클래스의 직접적인 상위 클래스를 찾기 위한 스키마의 검색은 필요하지 않다. 소스 클래스가 part_of에지와 직접 연결되어 있을 때, 그 part_of에지는 select연산으로 아무런 영향을 받지 않는다.



(a)연산 실행 전



(b)유도된 클래스 Customer1의 통합

(그림 2) refine 연산
(Fig. 2) refine operation

3.3.4 Union, Intersect, Diff 연산

Union연산은 두 소스 클래스에 속하는 인스턴스들의 합집합이 유도된 클래스의 인스턴스들의 집합이 되므로 유도된 클래스는 두 소스 클래스의 상위 클래스가 된다. Intersect에서는 두 소스 클래스에 공통적으로 있는 인스턴스들이 유도된 클래스의 인스턴스가 된다. 유도된 클래스는 두 소스 클래스의 직접적인 하위 클래스가 되므로 유도된 클래스의 통합위치를 찾기 위한 스키마 검색은 불필요하다. Diff 연산에서는 첫번째 소스 클래스에만 속하고 두번째 소스 클래스에는 속하지 않는 인스턴스들로 가상 클래스가 유도된다. 유도된 클래스는 첫번째 소스 클래스의 직접적인 하위 클래스가 되므로 유도된 클래스의 통합 위치를 발견하기 위한 스키마의 검색은 불필요하다. 소스 클래스가 part_of에지와 직접 연결되어 있을 때, intersect,diff연산은 그 part_of에지에 아무런 영향을 주지 않는다.

3.3.5 Refer 연산

Refer연산으로 유도되는 가상 클래스는 연산의 대상이 되는 소스 클래스의 하위 클래스가 된다. Refer연산에서 유도되는 클래스는 스키마내의 다른 클래스와 새로운 part_of관계를 형성하며, 그 part_of에지에는 refer연산에서 사용된 레이블이 주어진다. 유도된 클래스는 refer연산으로 새로 정의된 part_of관계를 통해 다른 클래스의 객체를 구성 객체로 가지는 복합 객체의 클래스가 된다. Refer연산으로 유도되는 클래스는 소스 클래스에서 상속 받는 property함수를 가지면서 연산에서 새로 정의된 part_of관계를 추가적으로 가지므로 소스 클래스의 하위 클래스가 될 수 있다. Refer연산은 클래스의 인스턴스에 대해서 영향을 주지 않으므로 유도된 가상 클래스는 스키마상에서 소스 클래스의 하위 클래스가 된다. Refer연산의 소스 클래스가 유도된 가상 클래스의 직접적인 상위 클래스가 되므로 유도된 클래스를 통합하기 위한 스키마 검색은 필요하지 않다.

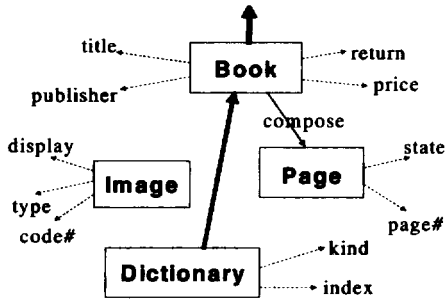
Refine연산은 property 함수 정의를 추가하여 가상 클래스를 유도하고 refer연산은 새로운 part_of관계를 정의하여 가상 클래스를 유도한다는 점에서 구별할 수 있다. Refine은 이미 정의된 property 함수를 약간 변형하여 새로운 property함수를 정의하고 추가하여 가상 클래스를 유도하지만 refer는 기존의 관계가 없을 때에도 새로운 part_of관계를 정의할 수 있고 새로운 part_of관계를 추가함으로써 가상 클래스를 유도한다. 클래스 유도 연산과 통합과정에서 part_of관계가 일반적인 property 함수처럼 사용되고 다루어지므로 refine연산이 기존의 property함수를 변형하는 것뿐만 아니라 완전히 새로운 property함수를 정의할 수 있도록 연산의 의미를 확장하여 생각한다면, refer연산은 refine연산의 특수한 경우로 볼 수도 있다.

Part_of관계가 관련되는 refer연산의 실행을 그림 3에서 보인다. 책에서 이미지를 표현할 수 있도록 클래스 Book과 스키마내의 다른 클래스 Image에 대해 새로운 part_of관계를 정의하여 클래스 Book1을 유도하면 그림 3.(b)와 같다.

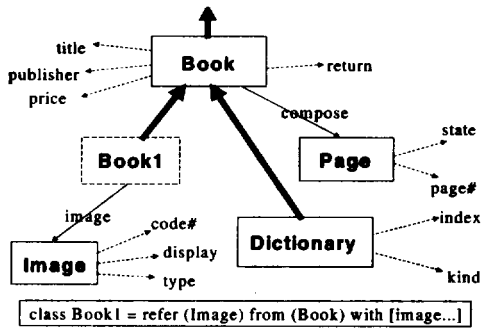
4. 뷰 생성

뷰를 생성하기 위해서 먼저 전역 스키마에서 뷰에 포함될 클래스가 선택된다. 뷰 클래스로 선택되는 클래스는 베이스 클래스일 수도 있고 가상 클래스일 수도 있다. 전역 스키마에서 뷰 클래스로 선택된 클래스가 복합 객체의 클래스인 경우, 구성 객체들의 클래스들도 뷰 클래스로 선택되도록 한다. 뷰 정의 과정에서 구성 객체의 클래스가 뷰 클래스로 선택되지 않았다면 자동으로 뷰내에 포함되도록 한다. 선택된 뷰 클래스들을 하나의 스키마로 연결하기 위해 루트 클래스가 필요하다. 전역 스키마에서의 클래스들의 관계를 고려하여 뷰 스키마의 루트 클래스로 적당한 클래스가 뷰 클래스로 선택되지 않았다면 자동적으로 뷰 스키마에 포함되게 한다.

초기의 뷰는 빈(empty) 상태이다. 전역 스키마에서 응용 프로그램의 뷰를 구성할 클래스들이 선택되어 뷰에 더해지면, 완전한 하나의 뷰를 만들기 위해 전역 스키마에서의 클래스들의 관계에서 추론하여 뷰 클래스들간의 관계를 형성하고 뷰 클래스들의 property 함수가 표현된다. 전역 스키마상에서 뷰 클래스가 상속받는 property함수들은 뷰 스키마에서의 클래스 위치를 고려해 적절히 표현된다.



(a)연산 실행 전

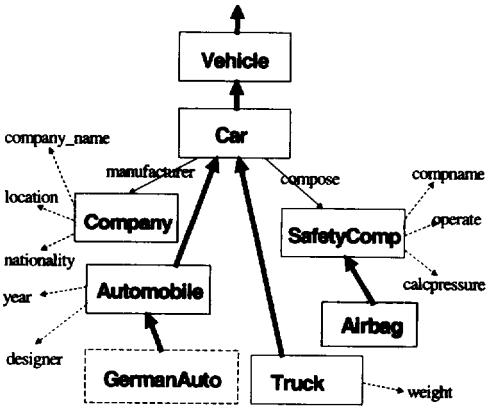


(b)유도된 클래스 Book1의 통합

(그림 3) refer연산
(Fig. 3) refer operation

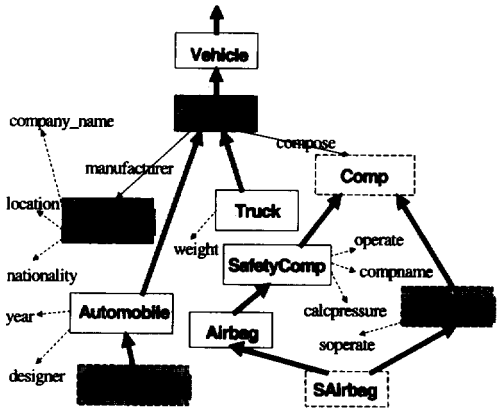
뷰 클래스로 선택된 클래스 A의 상위 클래스 SA가 전역 스키마에서 다른 클래스 B의 인스턴스를 구성하는 구성 객체의 클래스이고 B와 SA사이의 part_of 관계가 레이블 L1으로 표현되었다면 뷰 클래스 A는 L1이 의미하는 관계의 역관계(inverse relationship)[12]를 상속받을 수 있다. A의 상위 클래스가 뷰 클래스로 선택되지 않았을 경우, 뷰 스키마에서 이런 part_of 관계의 표현이 A와 B사이에 레이블 L1으로 이루어질 수 있다.

뷰의 생성을 설명하기 위해 그림 4의 예를 이용한다. 에어백의 성능이 좋은 독일산 승용차에 대한 뷰 AirbagCar의 생성을 그림 5에서 보여 주고 있다. 유도된 가상 클래스의 통합으로 확장된 전역 스키마인 그림 5.(a)에서 어두운 색으로 칠해진 클래스가 뷰 클래스로 선택된 클래스를 나타낸다. 클래스 Vehicle은 뷰 클래스로 선택되지 않았지만 뷰 스키마의 루트 클래스로서 뷰내에 존재하게 된다.

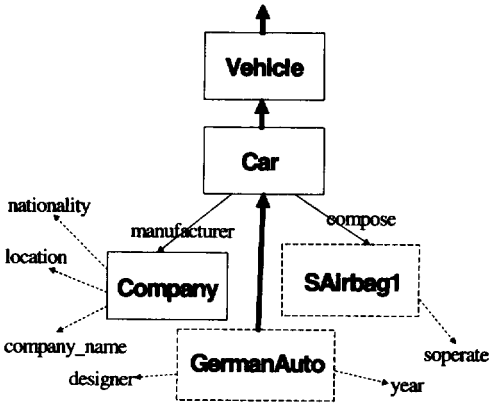


(그림 4) 자동차 스키마
(Fig. 4) Car Schema

뷰 클래스 SAairbag1은 상위 클래스 Comp와 Car 사이에 유지되는 compose라는 레이블이 주어진 part_of관계를 상속받을 수 있다. SAairbag1의 상위 클래스들 중 뷰 클래스로 선택된 것이 없으므로 뷰 스키마에서는 레이블 compose로 표현된 part_of 관계가 Car와 SAirbag1사이에 표현된다. 만약 SAirbag1이 뷰 클래스로 선택되지 않고 Car, Company, GermanAuto만이 뷰 클래스로 선택되었다면, 클래스 Comp가 자동으로 뷰 클래스로 지정되어 뷰 스키마에 포함된다.



(a) GS에서 뷰 클래스 선택



(b)완성된 뷰 스키마 AirbagCar

(그림 5) 전역 스키마 GS에서 뷰 스키마 AirbagCar의 생성
(Fig. 5) Generation of View Schema AirbagCar from GS

그림 5.(a)에서 클래스 Company가 뷰 클래스로 선택되지 않았더라도 생성되는 뷰 스키마내에는 클래스 Company가 자동으로 포함되어진다. Company는 복합객체의 클래스 Car와 레이블 manufacturer의 part_of 관계를 가지는 구성 객체의 클래스이므로 보다 완전한 의미의 뷰를 생성하기 위해 뷰 스키마에 포함되는 것이다. 뷰 스키마 AirbagCar를 그림 5.(b)에서 볼 수 있다.

5. 프로토타이핑

5.1 프로토타이핑에서의 제한 조건

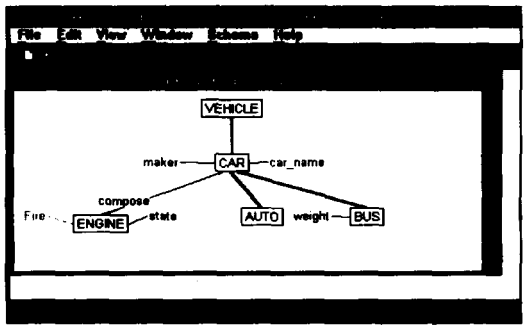
가상 클래스의 유도 단계에서 임의의 한 클래스에

대해 같은 property 함수가 여러 번 hide연산에서 숨겨지는 것은 불필요한 클래스들이 생성될 수 있으므로 피한다. 클래스의 property 함수와 part_of 관계에 주어진 레이블들은 각각 다른 이름을 사용하도록 한다.

프로토타이핑은 스키마 레벨에서 이루어지므로 각 클래스의 인스턴스에 대해서는 표현되지 않고, 클래스의 property함수들만이 표현된다. 새로운 property함수가 정의된 경우 스키마에서 그 함수가 표현되고 다루어지지만 함수의 의미까지는 다루어지지 않는다. 다중 상속(multiple inheritance)이 일어나는 경우 어떤 상위 클래스에서 상속받을 것인지의 문제는 고려되지 않았다.

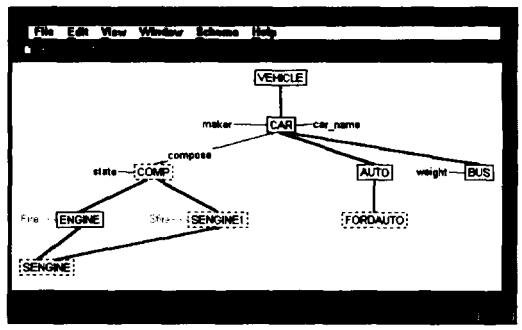
5.2 프로토타입의 적용 예

그림 6은 입력 화일을 읽어 생성된 Car에 대한 베이스 스키마이다. 클래스는 사각형으로 표현되고, 각 클래스의 property 함수들이 클래스를 나타내는 사각형의 측면과 연결되어 있다. Property 함수들 중 실선이 속성을 의미하고 점선이 메소드를 의미한다. 클래스들의 일반화 관계는 상위 클래스의 하단과 하위 클래스의 상단이 연결된 굵은 실선으로 나타낸다. 복합 객체의 클래스와 구성 객체의 클래스 사이의 part_of관계는 복합 객체들의 클래스의 하단과 구성 객체들의 클래스의 상단을 연결하는 가는 실선으로 표현되며, 적당한 레이블이 주어진다.



(그림 6) Car의 베이스 스키마
(Fig. 6) Base Schema of Car

이제 엔진이 정상적인 상태일 때만 연소되는 Ford사의 승용차에 대한 뷰 생성을 위해 필요한 전역 스키마를 만들어 보자. 먼저, 안전성을 보장하기 위해 엔진의 상태가 정상적일 때만 연소되도록 하는 함수 Sfire()를 새로 정의하여 추가하는 refine연산을 클래스 ENGINE에 적용하여 가상 클래스 SENGINE을 유도한다. 그런 다음, 엔진의 상태에 관계없이 연소되는 Fire()를 숨기기 위한 hide연산을 SENGINE에 대해 적용하여 클래스 SENGINE1을 유도한다. 클래스 SENGINE1은 스키마 상에서 직접적인 상위 클래스가 될 만한 클래스를 발견할 수 없으므로 부가적인 클래스 COMP를 생성하고, COMP를 SENGINE1의 직접적인 상위 클래스로 연결한다. 마지막으로 클래스 AUTO에 대해 select연산을 실행하여 Ford사의 승용차들에 대한 클래스 FORDAUTO를 유도한다. 유도된 클래스들이 통합된 최종적인 정역 스키마는 그림 7과 같다.



(그림 7) 클래스 FORDAUTO의 통합
(Fig. 7) Integration of class FORDAUTO

6. 결 론

본 논문에서는 객체지향 데이터베이스에서 다양한 응용 프로그램에 적합한 뷰를 제공하는 방법을 제시하고 구현방법을 기술하였다. 기존의 연구에서는 클래스들의 is_a관계만이 다루어졌지만 본 논문에서는 클래스들의 is_a관계와 part_of관계를 둘 다 고려하여 스키마를 생성하고 뷰 유도의 구현적 방법을 확장하였다.

Part_of관계를 스키마에서 다루기 위해 고려해야 할 사항들을 설명하고, 각 클래스 유도 연산에 따라 part_of관계가 어떻게 다루어지는지 살펴보았다. 가상 클래스를 유도하면 유도된 클래스는 스키마로 통합되며, 스키마에서의 적당한 통합 위치를 발견하기 위한 스키마를 검색하였다. 프로토타입에서는 is_a관계외에 본 논문에서 살펴본 클래스들의 part_of관계까지도 고려한 클래스의 유도와 통합을 보였다.

본 연구는 클래스들간의 part_of관계가 표현될 수 있는 스키마에서 뷰 스키마를 생성하므로, 상위 클래

스/하위 클래스로 표현되는 정보 외에 복합 객체들에 관련된 정보를 뷰에서 표현할 수 있는 효과가 있다. 보다 넓은 범위에 뷰가 적용될 수 있으므로 데이터베이스 정보의 검색뿐만 아니라 CAD/CAM 등의 응용분야에서 필요한 서비스를 포함한 다양한 요구에 보다 적절히 대응할 수 있다.

참 고 문 헌

[1] Serge Abiteboul and Anthony Bonner, "Object and Views", Proceeding of ACM-SIGMOD International Conference on Management of Data , pp. 238-247, May 1991.

[2] O.Deux et al. , "The Story of O2", IEEE Transactions on Knowledge and Data Engineering, Vol. 2, No.1, pp.91-108, March 1990.

[3] Nita Goyal et al. "Preliminary Report on (Active) View Materialization in GUI Programming", Proceeding of the Workshop on Materialized Views : Techniques and Applications, pp.56-64, June 1996.

[4] Won Kim, "Architecture of the ORION Next-Generation Database Systems", IEEE Transactions on Knowledge and Data Engineering, Vol. 2, No.1, pp.109-124, March 1990.

[5] Won Kim, "Object-Oriented Database : Definition and Research Directions", IEEE Transactions on Knowledge and Data Engineering, Vol.2, No.3, pp.327-340, September 1990.

[6] Won Kim and William Kelley, "On View Support in Object-Oriented Database Systems", Modern Database Systems, ACM Press, pp.108-129, 1995.

[7] Won Kim, "Object-Oriented Database Systems : Promises, Reality, and Future", Modern Database Systems, ACM Press pp.255-280, 1995.

[8] Angelika Kotz-dittrich and Klaus R. Dittrich, "Where Object-Oriented DBMSs Should Do Better : A Critique Based on Early Experiences", Modern Database Systems, ACM Press, pp.238-254 1995.

[9] Elke A. Rundensteiner and Lubomir Bic, "Set Operation in Object-Based Data Models", IEEE Transactions on Knowledge and Data Engineering, Vol.4, issue 4, pp.382-398, August 1992.

[10] Elke A. Rundensteiner, "Design Views for Synthesis", Technical Report in Univ. of Michigan, 1993.

[11] Cassio Souza dos Santos, Serge Abiteboul and Claude Delobel, "Virtual Schema and Bases", Proceeding of International Conference on Extending Database Technology, Vol.779, pp.81-94, 1994.

[12] David W. Shipman, "The Functional Data Model and Data Language DAPLEX", ACM Transactions on Database System, Vol.6, No.1, pp.755-771, March 1981.

[13] Richard M. Soley and William Kent, "The OMG Object Model", Modern Database Systems, ACM Press, pp.18-41, 1995.

[14] K. Wilkinson, Peter Lyngbak and Waqar Hasan, "The Iris Architecture and Implementation", IEEE Trasactions on Knowledge and Data Engineering, Vol.2, No.1, pp.63-75, March 1990.

[15] Edward Yourdon, "Object-Oriented Systems Design : An Integrated Approach", Prentice-Hall, 1994.

[16] 김성기, 이석호, "복합 객체의 시맨틱스 분석 및 모델 설계", 한국정보과학회 논문지, Vol.17, No.6, pp.655-664, November 1990.

[17] 안상현, 황수찬, 이석호, "객체중심 데이타 모델에서 뷰 개념의 지원", 한국정보과학회 논문지, Vol. 17, No.6, pp.643-653, November 1990.

[18] 이윤준, "객체 지향 데이타베이스 시스템", 한국정보과학회지 제12권 제3호 pp.9-23, April 1994.

차 현 주

1994년 울산대학교 전자계산학과 졸업(학사)

1997년 숙명여자대학교 전산학과 졸업(석사)

관심분야 : 객체지향 데이터베이스, 분산데이터베이스

윤 종 필

1981년 연세대학교 전기공학 학사

1987년 University of Florida, 컴퓨터공학 석사

1993년 George Mason University 전산학 박사

1994년부터 숙명여자대학교 전산학과 부교수

관심분야 : 데이터마이닝, 분산데이터베이스, 능동데이터베이스 멀티미디어 데이터베이스, 객체지향 데이터베이스