능동적 객체지향 데이타베이스에서 사용자 정의 제약조건의 역방향 전달에 관한 연구

도남철*·최인준*

Backward Propagation of User-Defined Integrity Constraints On Active Object-Oriented Database

C. Do and I. Choi

Abstract

The trigger mechanism in active object-oriented database systems is known to be a good tool for describing user-defined integrity constraints. It cannot adequately support, however, certain integrity constrains specified on the objects in class composition hierarchy. Those are the cases where the constraints must be maintained in the forward direction along the composition hierarchy as well as in the backward direction We call theses kinds of problems "backward propagation problem" and investigate several ways to resolve them using the currently available techniques. Based on them, a new constructor, called CONSTRAINT^{cch}, is proposed. The constructor can be realized with enhanced facilities for active OODBMS which we recommend in this paper.

1. 서 론

능동적 객체지향형 데이타베이스 관리시스템 (Active Object-Orieted Database Management System)은 능동적인 자료처리 기능(active facility)을 이용하여 자료의 무결성 제약조건 (integrity constraint)을 유지할 수 있도록 하고 있다. 트리거(trigger)를 기반으로 하는 능동적 기능은 다양한 무결성 제약조건을 사용자가 직접 정의할 수 있게 하여준다. 그러므로 객체 지향적 데이타베이스의 능동적 기능

* 포항공과대학교 산업공학과

은 차세대 데이타베이스의 중요한 기능의 하나로 부상하고 있다[Stonebraker 1990].

능동적 객체지향형 데이타베이스의 트리거가 무결성 제약조건을 표현하는 좋은 도구이지만 특정한 경우에는 적절하게 제약조건을 지원하지 못한다. 클래스 구성계층(class composition hierarchy)[1]을 이루는 객체들에 무결성 제약조건을 정의할 경우 현재의 트리거를 포함한 데이타베이스의 능동적 기능으로는 객체들의 수정(update)에 따른 적절한 무결성 제약조건 검사가 불가능하며, 이로 인하여 무결성 제약조건은 유지되지 못한다.

클래스 구성계층은 많은 응용분야에서 사용되므로 이에 대한 무결성 제약조건의 완벽한 지원이 필요하다. 클래스 구성계층은 클래스 계승계층(class inheritance hierarchy)과 더불어 대상을 모델링할 수 있는 객체지향형 데이타베이스의 가장 중요한 구조이다. 클래스 구성계층을 사용하여 모델링할 수 있는 예로, 복잡한 기계의 부품관계, BOM(Bill Of Material),그리고 전자기 판에서 각 전자부품 사이의 관계를 들 수 있다. 특히 객체지향형 데이타베이스의 주요 적용 분야인 CAD, CAM, 사무자동화등에 클래스 구성계층이 많이 응용되므로 이에 대한 무결성 제약조건의 지원은 매우 중요하다.

클래스 구성 계층을 이루는 객체에 대한 무결성 제약조건을 지원하기 위해서는 3가지 분야에 대한 연구가 필요하다. 첫째는 현재의 트리거를 이용하여 제약조건을 정의하였을 때 생기는 문제에 대한 규명이다. 이를 위해 본 연구에서는 역방향 전달(backward propagation)이라는 개념을 설명한다. 둘째로는 정의된 문제에 대한 해결 방법이다.본 연구에서는 문제를 해결하기 위한 3가지 역방향 전달 방법을 제안하며, 각각의 특징에 대해서도 언급한다. 마지막 연구는 클래스 구성계층을 이루는 객체에 대한 제약조건을 충실하게 지원할 수 있는 새로운 생성자 (constructor)에 대한 연구이다. 본 연구에서는 이 무결성 제약조건 생성자의 문법과 의미를 정의하였으며 기존의 데이타베이스의 확장된 능동적 기능을 이용하여 이 생성자를 구현하는 방법을 제안한다.

현재까지 역방향 전달기능을 지원하는 능동형 데이타베이스 관리시스템은 보고되지 않고 있다. 이는 기존의 능동형 데이타베이스 관리시스템은 클래스 구성 계층을 이루는 객체에 대한 무결성 제약조건을 적절히 지원하지 못하는 것을 뜻한다. 아울러 기존의 데이타베이스를 이용하여 역방향 전달을 지원하기 위해서는 트리거를 위시한 능동형 기능의 일부를 보완해야 한다. 본 연구에서는 현재 상용화된 능동적 객체지향형 데이타베이스의 능동적 기능을 이용하여 역방향 전달개념과 역방향 전달 방법을 설명하고, 새로운 제약조건 생성자를 구현한다. 아울러 기존의 능동적 객체지향형 데이타베이스가 역방향 전달을 지원하기 위해서 보완해야 할 사항을 제안한다.

이 논문은 다음과 같이 구성된다. 2장에서는 클래스 구성계층상의 객체에 무결성 제약조건을 기존의 트리거를 이용하여 정의하였을 경우 생기는 문제를 살펴본다. 문제에 대한 예와 일

1) 클래스 구성 계층(class composition hierarchy)은 일정 클래스의 애트리뷰트의 정의역(domain)이 임의의 클래스가 될 때 형성되는 계층구조이다[Kim 1990].

반화된 문제에 대한 고찰을 통하여 역방향 전달의 개념을 구체화 시킨다. 3장에서는 여러가지 역방향 전달을 지원할 수 있는 방법을 나열하고 이들의 특징을 간략하게 나열한다. 아울러 이들 방법간의 여러가지 측면에서의 비교도 첨부한다. 4장에서는 앞에서 언급한 방법 중 가상 클래스를 이용한 방법을 확장한 수평가상 클래스(Horizontal Virtual Class: HVC) 개념을 소개한 다. 5장에서는 HVC 방법을 위하여 기존의 능동적 기능을 확장하는 방안을 제안한다. 아울러 역방향 전달을 지원하는 무결성 제약조건 생성자의 문법과 의미를 정의하고 이를 구현하는 방법을 제안한다. 6장에서 본연구와 관련된 연구사례를 살펴본다. 7장에서는 연구의 결과를 정리하고 추후 연구과제를 언급하며 논문을 마친다.

2. 역방향 전달

2.1 트리거를 이용한 사용자 정의 무결성 제약조건

능동적 객체지향형 데이타베이스의 트리거는 사용자 정의 무결성 제약조건을 표현하는 좋은 도구로 알려져 있다. 특히 트리거의 조건절(condition clause)이나 실행절(action clause)에 메쏘드를 사용하므로써 복잡한 제약조건을 표현할 수 있다고 알려져 있다. 이 절에서는 능동적 객체지향형 데이타베이스의 트리거가 어떻게 무결성 제약조건을 정의할 수 있는지를 예를 통하여 보인다.

다음의 예는 현재 시판되는 능동적 객체지향형 데이타베이스 관리시스템 [UniSQL 1993]의 트리거[2]를 이용하여 공학분야에서 많이 발생하는 부품의 무게에 관한 무결성 제약조건을 표현한 것이다.

```
CREATE TRIGGER part_w_constraint
AFTER UPDATE ON Part
IF PartWeight() ON obj > 100
EXECUTE INVALIDATE TRANSACTION;
```

위의 part_w_constraint 트리거 정의에 따르면 Part 클래스의 한 인스탄스(instance)가 수정되면 데이타베이스는 자동적으로 수정된 객체에 대하여 무게를 계산하는 PartWeight() 메쏘드를 적용시켜 무게를 구하며, 만일 그 값이 100 이상이면 해당 트랜잭션(transaction)을 무효(roll back)화 시키고 100보다 적으면 무결성에 일치하는 객체라고 인정하여 이 수정 트랜잭션을 완료(commit)시킨다. 결과적으로 이 트리거 정의는 Part 클래스의 수정에 대한 무게의 무결성 제약조건을 유지시키게 된다.

예와 같이 트리거는 대상이 되는 객체의 상태를 변화시킬 수 있는 트랜잭션(예에서는 객체 수정)이 일어날 때마다 이 트랜잭션 결과가 주어진 조건을 만족하는지를 검사하여 상태변화를 허락하거나 무효화시킨다. 이 트리거의 조건부에 사용자가 원하는 객체의 무결성 제약조건을 정의하면 대상 객체는 항시 원하는 상

2) 이 논문에서는 역방향 전달(backward propagation) 개념과 이를 지원하는 생성자(constructor)를 설명하기 위하여 UniSQL/X의 트리거 생성자, 질의어, 그리고 가상 클래스 생성자를 사용한다. UniSQL/X는 UniSQL, Inc.의 등록상표이다.

태를 유지하며, 만일 제약조건을 어기는 변화가 이루어지면 트리거에 의하여 무효화 되게 된다.

2.2 클래스 구성 계층상에서 무결성 제약조건 정의의 문제점

비록 앞 절의 예와 같이 능동적 객체지향형 데이타베이스 관리시스템의 트리거 기능이 사용자정의 무결성 제약조건 표현에 효과적인 도구로 사용될 수 있지만 이 트리거 기능을 이용하여 클래스 구성 계층을 구성하고 있는 객체들에 대한 무결성 제약조건을 적절히 지원할 수 없다. 무결성 제약조건 정의의 문제점을 보다 자세히 살펴보기 위하여 다음의 예를 고려해 보자.

```
p:Part              m:Material
p.volume=30         m.density=2
p.material_type=m
p.PartWeight
{return(p.volum*p.material_type.density)}

TRIGGER part_w_constraint
AFTER UPDATE ON Part
IF PartWeight() ON obj>100.
INVALIDATE TRANSACTION;
```

그림 1 : 트리거를 이용한 Part 클래스의 무게에 관한 무결성 제약조건 정의

사용자는 Part와 Material 클래스에 속하는 객체 p와 m을 정의한다. 이는 그림 1에서 p:Part, m:Material로 표시된다. 특히 Part 클래스는 material_type 애트리뷰트 통해 Material 클래스 와 클래스 구성 계층을 유지하고 있다. 이 클래스 구성 계층을 통하여 객체 p의 material_type 애트리뷰트는 객체 m의 객체 식별자를 그 값으로 가지고 있다. 동시에 객체 p는 PartWeight()라는 무게를 계산하는 메쏘드도 가지고 있다. 이 메쏘드는 객체 p의 무게를 p.volume로 표현되는 객체의 부피값과 p.material_type.density로 표현되는 객체 p의 재질인 객체 m의 밀도를 곱하여 구한다. 앞에서 언급한 객체들 관계에서 Part객체의 무게가 100 이하라는 무결성을 정의하기 위하여 part_w_constraint 트리거를 이용하여 무결성 제약 조건을 정의하였다. part_w_constraint 트리거는 조건절에 포함하고 있는 PartWeight() 메쏘드를 수정된 객체에 적용하여 그 값을 기준으로 해당 트랜잭션이 제약조건에 만족하는가를 검사한다.

예에서 객체 수정에 의한 데이타베이스 상태(database state)의 변화를 살펴보기 위하여 다음과 같이 현재의 데이타베이스 상태를 표현할 수 있다.

S = {p.volume=30, m.density=2 ...}

현재의 데이타베이스 상태 S에서 p.volume의 값을 30에서 60으로 수정하면 다음과 같은 새로운 S′상태로 전환된다.

S′= {p.volume=60, m.density=2 ...}

데이타베이스가 S′상태로 변환되었을 때 part_w_constraint 트리거 정의에 의해서 객체 p에 그림 1의 PartWeight() 메쏘드를 적용하게 된다. PartWeight() 메쏘드는 p.volume 와 m.density 의 값을 곱한값인 120 되돌려주게 되고 이 값은 무결성 제약조건을 위배하므로 이 트랜잭션은 트리거의 조건절에 의하여 무효화 된다. 즉, S′ 데이타베이스 상태로 전환되지 못하고 이전의 S 상태로 되돌아가게 된다(roll back). 이와 같이 part_w_

constraint 트리거 정의로 사용자는 모든 Part 객체가 100 이하의 무게를 가질 것을 의도한다. 이제 다른 데이타베이스 수정의 예로 S 상태의 m.density를 2에서 5로 바꿀 경우를 고려하여 보자. 이 수정은 데이타베이스 상태를 S 에서 다음과 같은 S″로 전환시키게 된다.

$S'' = \{p.volume = 30, m.density = 5\}$

데이타베이스 상태 S″의 객체 p에 트리거 정의에 의하여 PartWeight() 메쏘드를 적용시키면 무게로 120이라는 값이 되돌려 진다. 이 값은 역시 사용자가 의도한 100 이하의 무게라는 무결성 제약조건에 위배되게 된다. 하지만 Material 클래스에는 무결성을 검사할 트리거를 정의하지 않았기 때문에 이 트랜잭션은 성공적으로 완료된다.

예의 무결성 제약조건은 p와 m객체가 모두 참여하고 있으므로 개체 p나 객체 m의 수정 모두가 무게 무결성에 영향을 줄 수 있다. 하지만 p에만 정의된 part_w_constraint 트리거는 m 객체의 수정으로 인하여 발생한 데이타베이스 전환에 대한 무결성 검사를 할 수 없게 된다. 결과적으로 S″ 상태의 데이타베이스는 part_w_constraint 트리거가 Part 클래스에 정의한 무결성 제약조건을 만족시키지만 사용자가 의도한 무게에 대한 무결성 제약조건을 만족시키지 못하는 상태가 된다.

2.3 역방향 전달 및 이의 체계적 지원 필요성

앞절의 예에서 볼 수 있듯이 클래스 구성 계층에 참여하는 객체에 정의된 무결성 제약조건 생성자(integrity constraint constructor)는 계층에 포함된 객체가 수정됨에 따라 무결성 제약 조건을 계층의 순방향으로 유지하는 기능뿐만 아니라 계층의 역방향으로 유지하는 기능도 있어야 함을 알 수 있다. 우리는 클래스 구성 계층의 역방향으로 무결성 제약조건을 유지시키는 것을 무결성 제약조건의 역방향 전달(backward propagation)이라고 부른다.

그림 2는 역방향 전달을 예의 인스탄스 수준이 아닌 클래스 수준에서 고찰한 것을 보여준다. 우선 역방향 전달이 발생하는 클래스 구성 계층이 클래스 Part와 Material 사이에 존재한다. 즉, Part 클래스의 material_type 애트리뷰트의 정의역이 Material 클래스이다(그림의 화살표). 아울러 이 계층구조는 Part 클래스를 루트(root)로 하고 Material 클래스를 종점으로 하는 방향성 그래프를 생성하며 그래프는 객체간의 패스(path) 표현을 통하여 다음과 같이 나타 낼 수 있다.

Part.material_type

이 패스 표현은 Part에서 Material 클래스의 density 애트리뷰트를 참조하는 표현에 사용될 수도 있으며 이를 참조패스(referencial path)라 부른다.

Part.material_type.density

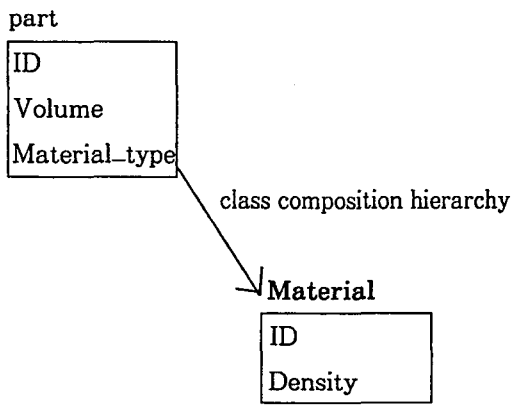


그림 2 : Part와 Material 클래스의 클래스 구성계층

역방향 전달은 위의 클래스 구성 계층을 구성하는 복수의 클래스에 대하여 무결성 제약조건을 정의하였을 때 발생한다. 그림 2의 클래스 계층구조에서 Part 클래스에 속한 인스탄스의 수정으로 인하여 관련된 Material 클래스의 인스탄스을 계층구조의 순방향으로 참조하여 무결성 제약조건을 만족시킬 뿐만 아니라 Material 클래스의 인스탄스가 수정되었을때 관련된 Part 클래스의 인스탄스를 계층의 역방향으로 찾아 제약조건을 유지시켜야 하는 것을 뜻한다. 예의 클래스 구성 계층에서 순방향 전달(forward propagation)은 Part.material_type.density 패스에 의하여 가능하지만 역방향 전달을 위한 Material.Part.volume와 같은 패스가 쉽게 생성되지 못함을 알 수 있다.

예에서와 같이 간단한 클래스 구성 계층에서는 사용자가 무결성 정의의 역방향 전달 문제와 이 를 해결하는 방법을 쉽게 인식 할 수 있다. 하지만 다음의 그림 3은 객체간의 클래스 구성계층이 너무 복잡하여 사용자가 기존의 무결성 제한 조건 정의 기능을 이용하여 역방향 전달 문 제를 쉽게 해결할 수 없는 경우를 보여준다.

그림 3에서 Machine 클래스 객체인 c(c:Machine)가 components 애트리뷰트 값으로 클래스 Part 의 p1, p2 객체를 가지고 있다. 이 때 Part 클래스는 그림 1에서의 Part와 동일한 스키마 구조 를 지닌다. 객체 c도 역시 자신의 무게를 계산하는 MachineWeight() 메쏘드를 가지고 있으며 이 메쏘드를 이용한 machine_w_constraint 트리거를 통하여 무게에 대한 무결성 제약 조건을 정의하고 있다.이 예에서 클래스 구성 계층의 복잡도를 살펴보기 위하여 Machine 클래스의 c 객체에서 Material 클래스의 m1, m2 객체의 density 애트리뷰트를 참조하는 참조 패스를 살펴보자. 이 패스는 c 객체의 components 값이 되는 p1, p2 객체와, 다시 이 객체의 material_type이 되는 m1, m2 객체를 모두 통과하여야 한다. 그러므로 이 패스는 다음과 같이 표현된다.

c.compoents.material_type.density =
{m1.density,m2.density}

이 패스상의 모든 객체의 수정은 c 객체의 무게에 대한 무결성 제약 조건을 위반할 수 있

```
c:Machine
c.components={p1,p2}
c.MachineWeight()
{(for i=0;i<(cardinal(components);i++)
        sum = sum + (PartWeight()  on
 components[i]); return(sum)}

TRIGGER machine_w_constraint
AFTER UPDATE ON Machine
IF MachineWeight() ON obj>1000
INVALIDATE TRANSACTION

p1:Part              p2:Part
p1.volume=20         p2.volume=10
p1.cost=2300         p2.cost=1400
p1.material_type=m1  p2.material_type=m2
p1.PartWeight()      p2.PartWeight()
{return(p1.volume*p1.material_type.density)}

m1:Material          m2:Material
m1.density=1.        m2.density=2.
```

그림 3 : 트리거를 이용한 Machine 클래스의 무게에 관한 무결성 제약조건 정의

다. 그러므로 이들 객체에 대하여 역방향 전달이 지원되어야 한다. 이 패스를 통과하는 객체들이 많아질 수록 한 객체의 수정이 다른 객체들에 미치는 영향이 복잡해 진다. 아울러 이 패스에 집합 형(set type)을 가진 클래스가 포함될 때 (예: c.components)는 이 패스가 집합 원소의 개수만큼 분기하게 되어 더욱 복잡한 역방향 전달 절차가 필요하게 된다. 그러므로 클래스 구 성 계층이 복잡해지면 계층에 포함된 한 객체의 수정에 따른 역방향 전달을 지원할 수 있는 체계적 지원 방법이 필요하게 된다.

복잡한 클래스 구성 계층을 가지는 객체들간의 역방향 전달을 지원하기 위해서는 우선 수정된 객체의 객체 식별자를 자신의 애트리뷰트의 값으로 가지는 객체를 찾을 수 있는 함수가 마련되어야 한다. 이 함수를 통하여 역방향 전달을 지원할 수 있는 세가지 방법을 고려할 수 있다. 첫째는 원래 사용자가 정의한 순방향 전달을 지원하는 트리거에 사용된 메쏘드와 같은 기능 (functionality)을 가진 메쏘드를 이를 실행시킨 트리거 정의와 함께 역방향 전달이 지원되어야 할 객체에 정의하여 주는 방식이다. 둘째는 역방향 전달이 필요한 객체에 트리거를 정의하되 이 트리거에는 순방향 전달에서 사용되었던 메쏘드를 실행시킬 수 있는 메시지를 포함하게 하는 방법이다. 역방향 전달을 지원하는 함수는 필요한 메쏘드를 적용할 객체를 찾는데 사용된다. 마지막 방법은 관계형 데이타베이스의 사용자 뷰(view)에 해당하는 가상 클래스를 사용하여 역방향 전달을 지원하는 방법이다. 가상 클래스는 무결성 제약조건에 관계된 모든 클래스를 기저 클래스(base class)로 가지며 기저 클래스에서 객체의 수정이 발생하였을 때 무결성 제약 조건을 검사할 수 있는 메쏘드를 실행시키는 역할을 한다. 이 세가지 방법은 다음 절에서 자세히 고찰한다.

3. 역방향 전달 방법

3.1 메쏘드 중복을 이용한 역방향 전달 방법

가장 간단한 역방향 전달 문제 해결 방법은 역방향 전달이 발생하는 객체에 원래 정의된 무결성 검사 메쏘드와 같은 기능을 가진 또 다른 메쏘드를 정의해주는 것이다. 이 메쏘드는 원래 정의된 메쏘드와 같이 수정된 객체의 애트리뷰트 값과 이 객체와 클래스 구성계층을 이루는 객체의 애트리뷰트 값을 참조하여 무결성을 검사하게 된다. 하지만 원래의 메쏘드와 다른 점은 수정된 객체를 애트리뷰트 값으로 가지는 객체에 접근하기 위하여 클래스 구성계층을 역으로 찾아가는 기능이 필요하다는 점이다. 이와 같은 클래스 구성계층을 역으로 찾아가는데 필요한 함수인 Backward()를 데이타베이스 질의어로 나타내면 다음과 같다.

```
OBJECT Backward(updated, att_domain, attribute)
{
SELECT Object FROM att_domain Object
    WHERE Object.attribute=updated;
return(Object);
}
```

이 함수는 클래스 구성계층에서 수정된 객체(updated)를 자신의 attribute 애트리뷰트로 값으로 가지고 있는 객체의 식별자(Object)를 돌려준다. att_domain은 Object가 속한 클래스의 이름이다. 즉, 이 함수에서 되돌려 지는

객체 식별자는 인수로 포함된 객체의 무결성 제약조건을 역방향으로 전달할 대상이다.

한 객체에 여러 개의 객체가 클래스 구성계층 관계를 가지고 있는 경우 Backward()함수는 여러 개의 객체를 되돌려 줄 수 있다. 이 경우에는 시스템 스택(system stack)을 이용하여 되돌려지는 객체 집합의 각 원소에 대하여 필요한 메쏘드를 반복해서 적용할 수 있게 하여 주면 된다. 문제를 간단히 하여 이해를 돕기 위하여 Backward()함수가 객체 집합을 되돌리는 경우는 앞으로의 예에서 제외하고 1개의 객체만을 되돌리는 경우만 예로 들겠다.

메쏘드 중복 방법의 예로 1장의 그림 1의 Part 클래스 무게에 대한 무결성 정의를 사용하여 보자. 메쏘드 중복 방법을 이용하여 Part 클래스의 무게에 대한 무결성을 위한 역방향 전달 기능을 정의하기 위하여 Material 클래스의 객체 m에 대하여 다음과 같은 메쏘드와 트리거가 정의된다.

```
m.PartWeight2()
{return(Backward(m,Part,material_type).volume * m.density)}

TRIGGER trigger_2
AFTER UPDATE ON Materia
IF PartWeight2() ON obj>100.
INVALIDATE TRANSACTION;
```

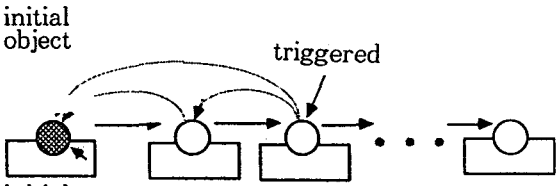
예에서 m 객체를 수정함으로 발생하는 역방향 전달을 지원하기 위하여 PartWeight()와 기능이 같은 PartWeight2()를 정의한 것을 보여주고 있다. 이 메쏘드의 기능은 유사하나 PartWeight2()에서는 클래스 구성계층의 하위 계층에 속한 m 객체에서 역으로 상위 계층에 있는 p 객체를 찾기 위해 Backward(m, Part, material_type) 함수가 사용된 것이 다르다는 점을 알 수가 있다.

기능이 유사한 메쏘드를 클래스 구성계층에 참여한 객체에 정의함으로써 이 객체를 수정하므로써 발생하는 무결성 제약조건을 검사할 수 있는 역방향 전달이 가능하게 되었다. 하지만 이 방법은 새로운 메쏘드를 무결성 정의에 참여한 모든 클래스에 정의해야 하기 때문에 코드의 중복이 필연적이다. 아울러 역방향 전달을 자동으로 지원할 경우 원래의 메쏘드로부터 새로이 정의해야 할 메쏘드를 생성해야 하는 문제를 해결해야 한다. 예의 경우에는 간단한 규칙을 적용하여 메쏘드를 생성해 낼 수 있지만 클래스 구성계층이 객체의 집합과 관계를 가지고 있을 경우나 다층구조의 계층을 가지고 있는 경우 복잡한 규칙을 적용해야 하는 문제가 있다. 아울러 무결성 제약조건에 참여하는 객체가 늘어날수록 각 객체를 참고해야 하는 메쏘드의 크기가 증가하는 문제도 발생한다.

메쏘드 중복 방법에 의하여 역방향 전달이 지원될 경우 객체를 수정하는 사용자는 비교적 단순한 권한만이 필요하다. 사용자는 단지 무결성에 관련된 애트리뷰트를 읽을 수 있는 권한만 있으면 된다. 하지만 권한의 단위가 클래스이고 관계형 데이타베이스의 뷰(view)와 같은 기능을 통해 권한의 조정을 하지 않는다면 무결성에 관계된 모든 객체에 대하여 읽기 권한을 가져야 무결성 제약조건을 검사할 수 있다. 앞의 그림 3에서 Material 객체를 수정하면서 무게에 대한 무결성 검사를 하기위해서는 Part와 Machine 클래스에 대한 읽기 권한이 필요하다. 만일 현재 객체를 수정하는 사용자가 Part 클래스에 대한 읽기 권한이 없고 뷰를 통해 선별적 읽기가 불가능하다면 Machine의

무게에 대한 무결성 검사는 하지 못하게 된다.

그림 4는 메쏘드 중복의 방법을 적용하여 N개의 객체가 집합 자료형식(set datatype)을 사용하지 않는 클래스 구성계층으로 연결되어 있는 경우 클래스 구성계층상의 한 클래스에 속한 객체의 수정에서 생긴 무결성 검사시 필요한 참조 횟수를 나타낸 것이다. 메쏘드 중복 방법은 무결성 검사시에 Backward() 함수를 이용한 역방향 전달과 일반적인 클래스 구성계층 관계를 이용한 순방향 전달을 한다. 즉 자신을 클래스 구성계층상의 상위 클래스에 속하는 객체는 역방향 전달하고 자신이 하위 클래스에 속하는 객체는 일반적인 순방향 전달을 하게 된다. 그러므로 주어진 가정하에서 무결성 검사에 필요한 모든 클래스 구성계층을 통한 접근 횟수는 N−1회가 된다.



$$\left\{\begin{matrix}\text{forward propagation}+\\ \text{vackward propagation}\end{matrix}\right\}=N-1$$

그림 4 : 메쏘드 중복 방법 사용시 타 객체 접근 횟수

3.2 메쏘드 호출을 이용한 역방향 전달 방법

메쏘드 중복 방식의 비효율적인 코드 사용 문제를 해결한 것이 메쏘드 호출 방식이다. 메쏘드 호출방식은 객체지향적 개념중의 하나인 메시지 전달(message passing)을 이용하여 기존의 정의된 무결성 검사용 메쏘드를 재사용하는 것이다. 즉 Backward() 함수를 이용하여 수정된 객체를 애트리뷰트 값으로 가지는 객체를 찾은 다음, 이 객체에 무결성 제약조건을 위하여 정의된 메쏘드를 실행하라는 메시지를 보내는 것이다. 이 메시지를 받은 객체는 무결성을 검사하기 위하여 이 메쏘드를 실행시키게 된다.

메쏘드 호출 방식을 설명하기 위하여 위에서 예로 들었던 1절 그림 1의 Part 클래스의 무게 무결성에 대한 역방향 전달 문제를 다시 살펴 보겠다. 역방향 전달 문제를 해결하기 위하여 메쏘드 호출 방식은 Part 클래스에 정의된 PartWeight()메쏘드를 호출하면 되므로 다음과 같은 트리거를 Material 클래스에 정의하면 된다.

```
TRIGGER trigger_2
AFTER UPDATE ON Material
IF PartWeight() on Backward(obj,Part,
        material_type)>100.
INVALIDATE TRANSACTION
```

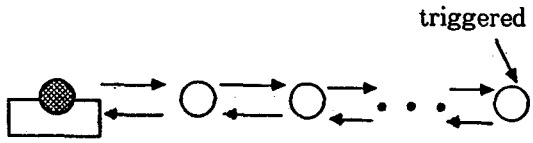
만약 m 객체를 수정할 경우 Backward() 함수를 사용하여 m을 material_type 애트리뷰트의 값으로 가지는 Part 클래스에 속한 객체를 찾게 된다. 이렇게 검색된 객체(예에서 p)에 PartWeight()을 적용하라는 메시지를 보내므로써 현재 일어난 객체수정에 대한 무결성을 검사하게 된다. 이 방법은 앞의 메쏘드 중복 방법에 비하여 다시 메쏘드를 정의하지 않으므로 코드를 재사용할 수 있다. 아울러 역방향 전달을 자동으로 지원할 경우 단지 사용자가 정의한 메쏘드를 호출하면 되므로 간단한

규칙으로도 가능하다.

하지만 메쏘드를 이용하여 객체를 검사하는 과정에서 p 객체는 수정하려는 m 객체의 값이 아닌 과거의 객체를 클래스 구성계층 관계로 지칭할 수 있다. 그러므로 적절한 조치를 해주지 않는다면 이 메쏘드는 과거의 객체값을 참고로 무결성을 검사하게 된다. 이 문제는 5장의 기존의 능동적 개체지향형 데이타베이스의 문제점에서 다시 다루기로 한다.

메쏘드 호출방법은 다른 객체의 메쏘드를 호출하므로 해당 객체에 대한 읽기 권한 뿐만 아니라 해당 메쏘드에 대한 실행 권한도 필요하게 된다. 그러므로 이 객체를 수정하는 사용자는 클래스 구성계층에 참여하는 객체에 대하여 메쏘드 중복 방법에 비하여 보다 높은 수준의 권한을 가져야 한다. 아울러 메쏘드 중복 방법과 마찬가지로 무결성 검사에 관계된 모든 객체에 대한 읽기 권한도 있어야만 무결성 검사를 할 수 있다.

메쏘드 호출 방식은 N개의 객체가 무결성 검사에 참여하고 집합 형식의 클래스 구성계층 관계가 포함되지 않을 경우 무결성 검사 메쏘드를 찾는 역방향 전달에서 N−1회, 그리고 무결성 검사에서 N−1회 이상의 타 객체에 대한 접근이 필요하다. 그러므로 그림 7과 같이 객체가 클래스 구성계층의 제일 하위 클래스에 속할 경우 무결성 검사를 위해서 2N−2의 회수의 클래스 구성계층 관계를 통한 접근이 필요하다.



N objects in class composition hierarchy

forward propagation = N − 1

bachward propagation = N − 1

그림 5. 메쏘드 호출 방법 사용시 타객체 접근 횟수

3.3 가상 클래스를 이용한 역방향 전달 방법

마지막 방법은 무결성 검사에 관련된 모든 객체를 기저 클래스(base class)로 하는 가상 클래스를 정의한후 이 클래스를 통하여 역방향 전달을 지원하는 방법이다. 이 방법에서 사용하는 가상 클래스는 일반적인 객체지향적 데이타베이스에서 사용하는 가상 클래스와는 다르게 클래스 계승계층에 관심을 두는 것이 아닌 클래스 구성계층 관계에 중점을 두고 있다. 아울러 이 가상 클래스는 무결성에 관계된 메쏘드도 같이 가지고 있게 된다. 그러므로 객체수정이 일어날 경우 이 가상 클래스를 통하여 무결성 검사에 필요한 메쏘드를 실행시키게 된다.

역시 앞에서 사용한 그림 1의 Part 클래스의 무게에 관한 무결성 문제를 해결하기 위하여 이 방법을 적용하여 보자. 우선 관계된 모든 클래스(Part, Material 클래스)를 기저 클래스로 하는 가상 클래스를 정의하면 다음과 같다.

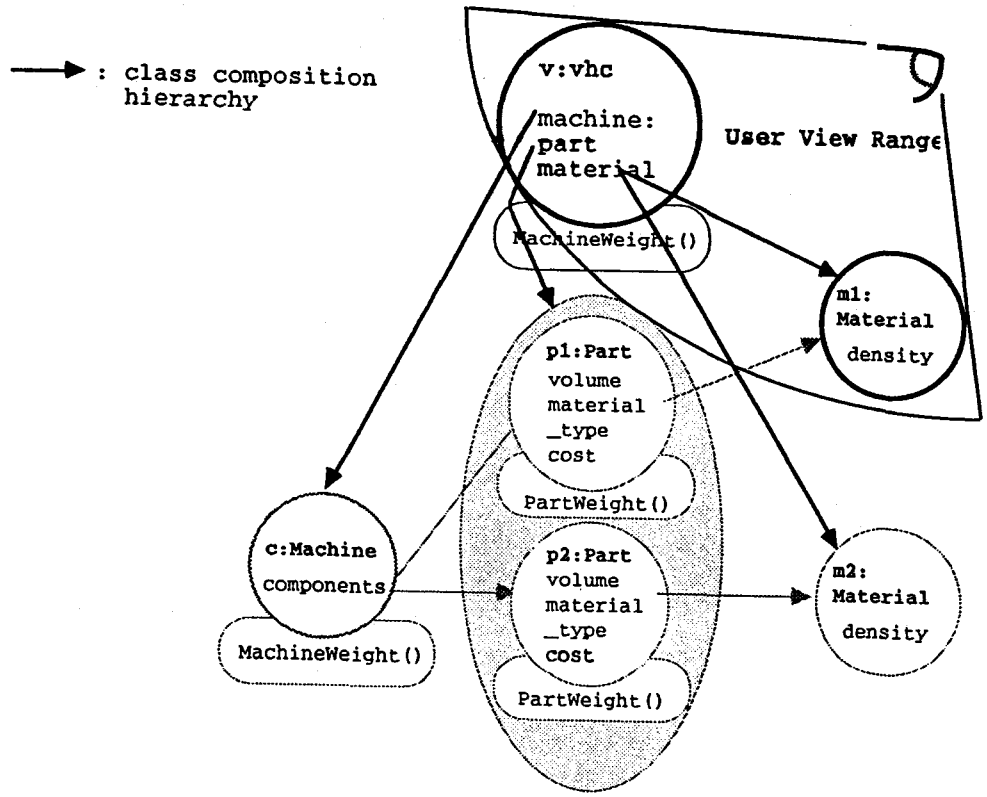


그림 6 : 가상 클래스를 이용한 역방향 전달 방법에서의 사용자 권한

```
CREATE VIRTUAL CLASS hvc
{part Part,
 material Material}
AS SELECT p, p.material_type FROM Part p
METHOD PartWeight();
```

이 정의로 다음과 같은 객체가 hvc 클래스의 인스탄스로 생긴다.

```
h:hvc
h.part = p
h.material = m
h.PartWeight()
```

역방향 전달을 위해서는 m에 다음과 같은 트리거를 정의한다.

```
TRIGGER trigger_2
AFTER UPDATE ON Material
IF PartWeight() ON Backward(obj,hvc,
        material).part>100.
INVALIDATE TRANSACTION;
```

이 방법은 클래스 구성계층에 관계된 객체의 식별자를 가상 클래스를 이용하여 준비한 후에 수정이 일어난 객체는 트리거의 Backward() 함수를 이용하여 자신과 클래스 구성계층을 이루는 hvc 클래스의 객체를 찾아 이 객체에 정의되어 있는 메쏘드를 실행 시키는 것이다. 예에서 Material 클래스의 한 객체에 수정이 일어나면 Backward() 함수를 이용 h를 찾고 h

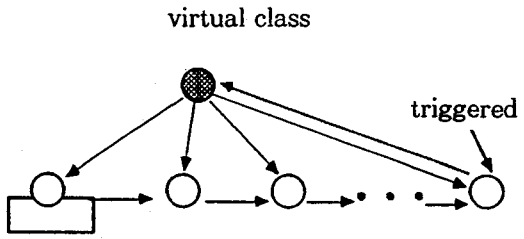
에 존재하는 p 객체(hvc.part)에 대하여 무결성을 검사하는 PartWeight() 을 적용하여 무결성을 검사하게 된다.

가상 클래스를 사용하는 방법은 가상 클래스를 새로 정의할 뿐 기존의 객체를 그대로 사용하므로 코드나 객체의 중복은 없다. 아울러 자동화시에도 새로운 메쏘드를 중복시킬 필요가 없고 비교적 간단한 규칙으로 역방향 전달을 지원할 수 있다. 가상 클래스를 정의하는 방법은 5장에서 다룬다.

권한부여에서는 메쏘드 호출 방법과 마찬가지로 사용자에게 무결성 제약조건을 위해 정의된 메쏘드(예에서 PartWeight())를 실행 시킬 수 있는 권한이 필요하다. 하지만 가상 클래스는 관계형 데이타베이스의 뷰 기능에서와 같이 무결성 기능에 관계된 정보외에는 접근을 허용하지 않을 수 있다. 예로 Part 클래스에 부품의 가격을 기록한 애트리뷰트가 있고 이 정보는 무결성을 검사해야 하는 사용자에게 공개할 수 없다면 가상 클래스를 통하여 가격을 공개하지 않고 무결성을 검사하도록 할 수 있다.

효율적인 권한의 예가 그림 6에 잘 나타나 있다. 이는 그림 3의 Machine의 무게 무결성 제약조건 예를 가상 클래스를 이용한 역방향 전달 방법을 사용하여 표현하고 있다. 한 사용자에게는 Material에 대한 수정 권한이 있고 Part나 Machine에 대한 읽기 권한이 없다고 가정하였을 때 앞의 두가지 역방향 전달 방법(메쏘드 호출, 메쏘드 중복)으로는 무결성 검사가 불가능하게 된다. 하지만 가상 클래스에 대한 읽기 권한을 무결성 제약조건 검사를 위해서 허용하면 사용자 는 무결성에 꼭 필요한 c.components, p.material_type, 그리고 이들과 클래스 구성계층을 이루는 Machine 클래스의 객체 식별자만을 알게 된다. 이때 사용자에 대하여 Part의 클래스 구조가 알려지지 않았다면 p 객체의 cost 애트리뷰트에 접근할 수 없고, 동시에 무결성 검사는 가능해 진다.

가상 클래스를 이용한 역방향 전달 방법은 메쏘드 호출 방식에 비하여 보다 적은 클래스 구성 계층을 통한 타 객체에 대한 접근 횟수를 보인다. 그림 9는 이 방식을 이용하였을 때 접근 횟수를 나태내고 있다. 그림에서 보는 바와 같이 무결성 검사에 참여하는 N개의 객체 어디에서 수정이 일어나도 1 회의 역방향 전달을 통하여 가상 클래스에 존재하는 무결성검사 메쏘드를 호출할 수 있다. 아울러 호출될 메쏘드 상에서 무결성을 검사하기 위하여 N−1회의 접근이 필요하다. 이때는 클래스 구성계층을 순방향으로 전파하게 된다. 이 방법을 메쏘드 중복방법과 비교하여도 1회 더 가상 클래스의 해당 인스탄스를 찾기위해 역방향 전달하는 차이만을 볼 수 있다.



N objects in class composition hierarchy

→ forward propagation = N − 1

← backward propagation = 1

그림 7 : 가상 클래스를 이용한 방법에서의 타 ㄴ객체 접근 횟수

4. 수평가상 클래스

4.1 가상 클래스를 이용한 역방향 전달의 장점

우리는 역방향 전달을 지원할 수 있는 3가지 방법을 제안하고 이를 분석하여 보았다. 각각의 방법을 분석하여 비교한 결과 가상 클래스를 이용한 역방향 전달이 여러가지 측면에서 많은 장점을 가지고 있음을 알았다. 가상 클래스를 이용한 역방향 전달의 장점을 정리하면 아래와 같다.

- 가상 클래스를 사용하므로 자료의 중복을 피할 수 있으며 메쏘드 코드도 재 사용할 수 있다. 특히 메쏘드를 재사용 하므로써 데이타베이스 관리 시스템이 자동으로 역방향 전달을 지원할 때 메쏘드 중복 방법보다 쉽게 이를 지원할 수 있다.
- 무결성 제약조건에 연관된 객체가 수정되었을 때 제약조건을 검사하기 위한 클래스 구성계층상의 타 객체 접근횟수(그림 7)가 메쏘드 호출 방법(그림 5)보다 적고, 메쏘드 중복(그림 4)과 는 1회의 차이가 날 뿐이다.
- 가상 클래스는 관계형 데이타베이스의 뷰 기능과 같이 무결성 제약조건에 필요한 정보만을 선택적으로 제공하므로 무결성 제약조건을 검사하려는 사용자의 불필요한 접근을 효과적으로 제한할 수 있다.
- 무결성 제약조건 검사에 참여하는 객체와 메쏘드를 하나의 클래스 단위로 관리하므로 무결성 제약조건 정의의 의미적 단위를 표현할 수 있고, 무결성 제약조건 관리도 효과적으로 할 수 있다.

4.2 수평가상 클래스로의 확장

우리는 "가상 클래스를 이용한 역방향 전달 방법"에서 사용된 가상 클래스의 이름을 특별히 수 평가상 클래스(Horizontal Virtual Class;HVC)로 정하였으며 이 HVC는 객체지향적 데이타베이스의 가상 클래스의 일종이지만 몇가지 특별한 성질을 가지고 있다.

첫째로 HVC를 형성하는 기저 클래스는 단일 클래스 구성계층에 포함되는 클래스로만 구성되어 있다. 일반적으로 객체지향데이타베이스의 가상 클래스가 상위클래스와 하위클래스의 클래스 계승계층 구조를 지닐 수 있는 반면, HVC는 계승계층구조가 아닌 구성계층구조에 포함된 클래스를 기저클래스로 사용한다. 특히 HVC의 H(Horizontal)의 의미는 가상 클래스의 기저 클래스가 모두 객체의 수직적 계승계층구조가 아닌 수평적인 구성계층구조 관계에 있음을 뜻한다.

둘째, HVC는 모든 애트리뷰트의 자료형이 객체 식별자로 이루어진다. 이 애트리뷰트는 무결성 제약조건 검사에 참여하는 각 객체의 식별자를 가지게 되며, 만일 기저 객체간의 클래스 구성 계층이 클래스 집합의 참여로 이루어졌다면 HVC에서도 해당 애트리뷰트의 자료형이 객체 식별 자들의 집합으로 구성된다. 이는 HVC 가 무결성 제약조건 검사의 파생을 연결하기 위하여 관계 된 객체들의 식별자를 관리하는 역할을 하기 때문이다. 특히 이러한 객체 식별자중 무결성 제약조건 검사가 처음으로 시작되는 객체 식별자(예: 그림 3의 Machine 클래스 무게 무결성 제약조건 예에서는 m, 그림 1의 Part 무게 무결성 제약조건

예에서는 p)를 초기 객체(initial object)라 하며 이 객체는 뒤에서 언급하는 초기 메쏘드(initial method)가 적용되는 객체이다.

셋째, HVC는 무결성 제약조건을 위한 초기 메쏘드를 포함한다. 무결성 제약조건을 검사하기 위한 메쏘드는 1개의 메쏘드로 구성되어 있거나 여러개의 메쏘드로 구성되어 있을 수 있다. 초기 메쏘드를 시작하면 관계된 메쏘드가 다시 이 메쏘드로부터 호출되어 전체 검사가 진행된다. 이들 메쏘드는 다시 다른 메쏘드를 호출하여 필요한 무결성 제약조건을 검사하게 된다. 이와 같이 제약조건 검사의 시작에 필요한 초기 메쏘드를 HVC가 가지고 있게 된다.

넷째, HVC는 관련된 객체와 메쏘드를 가지고 무결성 제약조건을 표현하는 추상적 데이타형 (abstract datatype)이다. HVC는 자신의 애트리뷰트에 무결성 제약조건에 참여된 모든 객체들의 식별자를 가지고 있다. 즉, 무결성 제약조건에 관계된 자료(객체의 식별자로 대표되는)를 가지고 있는 것이다. 아울러 초기 메쏘드로 대표되는 자료의 프로시듀어(procedure)를 가지고 있다. 그러므로 HVC는 클래스 구성계층에 포함된 객체에 정의된 무결성 제약조건을 나타내는 추상적 데이타형이다. 이 추상적 데이타형은 정의된 무결성 제약조건을 의미적으로 정리해 줄 뿐만 아니라, 실질적으로 무결성 제약조건을 관리하는 기본 단위가 될 것이다.

5. 현재 트리거 기능의 제한성과 무결성 제약조건 생성자

앞절에서 HVC와 트리거를 이용하여 역방향 전달 문제를 해결할 수 있음을 보였다. 그러나 기존의 능동적 객체지향형 데이타베이스에 본 논문에서 제안한 역방향 전달기능을 지원하기 위해서는 몇가지 기능을 확장하거나 추가하여야 한다. 이 절에서는 역방향 전달을 지원하기 위하여 해결해야할 기존 데이타베이스의 문제점과 이를 해결하기 위한 확장방안을 제시한다. 아울러 확장된 능동적 기능을 이용하여 클래스 구성계층상의 객체에 정의된 무결성 제약조건을 지원하는 생성자를 정의하고, 이의 구현 방법을 제시하겠다.

5.1 문제점

첫째 문제점은 기존의 능동적 객체지향형 데이타베이스가 본 논문에서 제안한 Backward () 함수의 기능을 가지고 있지 않은 점이다. 이 기능은 역방향 전달문제를 해결하기 위해서는 꼭 필요하므로 무결성 제약조건을 지원하려면 이 함수를 3.1 절의 질의어를 이용하여 정의하거나 객체 사이의 중복연결 리스트(double linked list)를 이용하여 구현하여야 한다.

두번째 문제는 역방향 전달 지원할 경우 일반적인 트리거 정의에 대한 불필요한 역방향 전달을 지원할 수 있다는 점이다. 트리거는 무결성 정의외에도 여러 가지 용도로 사용된다. 예로 일정한 시간이 되면 월급을 계산하라는 규칙을 트리거를 이용하여 표현할 수 있다. 이와 같은 상황에서 역방향 전달을 요하는 클래스 구성계층 관계상의 무결성 정의도 트리거를 사용하여 정의 한다면 위의 예와 같은 일반적 트리거와 구분이 되지 않는다. 즉, 이용자가 정의한 일반적 트리거정의와 역방향 전달을 요하는 무결성을 정의한 트리거가 구분되지 않으므로 역방향 전달을 선별적으로 적용해야 하는 문제가 생긴다. 위의 월급계산의 경우는 역방

향 전달 기능을 제공하면 사용자가 원하지 않는 결과를 초래할 수 있다. 그러므로 이 문제를 해결하기 위하여는 트리거와 역방향 전달이 제공되어야 하는 무결성 제약조건을 분리해서 정의할 수 있어야 한다. 다음절에서는 기존의 트리거 생성자를 확장시켜 무결성 제약조건 생성자를 확장하여 보았다. 아울러 사용자가 이 확장자를 사용하였을 때 시스템이 어떻게 HVC를 이용하여 역방향 전달을 지원할 수 있는지를 보이겠다.

마지막 문제는 메쏘드 호출 방식에서 객체 수정시의 검사용 임시 객체(New 객체)가 수정이전 객체의 클래스 구성계층 정보를 가지고 있지 않은 경우이다. New 객체는 새로운 객체가 조건을 만족시키지 못할 경우 다시 무효화시키는 비용을 덜기 위해 수정대상 객체를 임시 복사하여 이 객체에 수정할 값을 써 놓고 대신 검사하는 객체를 말한다. 이 과정에서 대상 객체를 복사할 때 객체가 가지는 클래스 구성계층까지 복사해야만이 역방향 전달에 필요한 검사를 할 수 있다. 만일 클래스 구성계층을 복사하지 않는다면 검사가 되는 객체는 과거에 클래스 구성계층에 연결되어 있는 객체가 될 것이다.

5.2 HVC를 이용한 무결성 제약조건 생성자

새로이 확장되는 클래스 구성계층상의 무결성 생성자는 다음과 같은 문법구조를 가지고 있다.

CONSTRAINT^{och} ON initial_class, target_class
AS condition;
initial_class=CLASS_NAME
target_class=target_class CLASS_NAME
condition=logical_expression

CONSTRAINT^{och} 생성자는 initial_class와 무결성에 관련된 target_class에 대하여 무결성 제약조건을 정의한다. 이때 target_class는 복수개의 클래스가 될 수 있으며 initial_class의 target_class를 루트로 하는 클래스 구성계층에 포함된 클래스라야 한다. condition은 참, 거짓 값을 되돌릴 수 있는 논리표현(logical expression)으로 이 표현을 만족하는 객체만이 데이타베이스에 존재할 수 있다.

이 생성자의 의미(semantics)를 보다 자세히 표현하면 다음과 같다.

{state′}=Transition({state})
Transition=Update
{state′}={ o ∈ state, TYPE(o) ∈ {initial_class} ∪ target_class : condition(o) =TRUE}
ClassCompositionSet(initial_class)⊃ target_class

이 생성자는 데이타베이스의 과거 상태인 {state}를 Update를 통해 새로운 상태인 {state′}로 바꾸는 경우에 적용되며 새로운 상태에서 initial_class와 target_class에 속한 객체는 주어진 condtion을 만족시켜야 한다. 아울러 target_class는 initial_class의 ClassCompositionSet()의 부분집합이어야 하는데 이는 initial_class를 루트로 하는 클래스 구성계층에 포함된 모든 클래스 집합의 부분집합이어야 함을 뜻한다.

생성자의 의미에서 나타난 바와 같이 CONSTRIANT는 클래스 구성계층에 포함된 객체만이 무결성에 관련된 경우에 적용되는 매우 제한적인 생성자이다. 그러므로 후에 연결될

무결성 제약조건 생성자의 한 모듈로써 사용될 수 있다. 만약 initial_class 루트로 하는 클래스 구성계층에 포함되는 모든 클래스가 초기 메쏘드와 이로부터 호출되는 메쏘드에서 참조되는 모든 클래스(target_class)를 포함한다는 것을 데이타베이스 시스템이 자동으로 검증할 수 있다면 이 CONSTRAINT^{cch}를 자동으로 정의할 수 있을 것이다. 아울러 이 경우에는 사용자가 직접 target_class를 입력할 필요도 없다.

새로이 확장되는 CONSTRIANT^{cch}는 앞 절에서 언급한 기능을 확장하면 기존의 트리거와 HVC를 이용하면 구현할 수 있다. 다음은 주어진 무결성 제약조건 정의로부터 역방향 전달을 지원하는 HVC와 트리거를 정의하는 과정(procedure)이다. 단 condition 부분은 초기 메쏘드를 사용하여 정의하였다고 가정한다.

Input : initial_class, target_class, condition(with Initial Method)

(1) initial_class와 target_class를 각기 정의 구역으로 하는 애트리뷰트를 가진 가상 클래스를 만든다. 이때 집합 자료 형태가 클래스 구성계층에 존재하면 대응하는 애트리뷰트부터는 모두 해당 클래스의 집합을 정의역으로 한다.

(2) 가상 클래스에 사용하는 질의는 Initial_class와 target_class에 속한 각 클래스를 질의 대상으로 하며 이는 참조패스 형식으로 표현한다.

(3) condition상의 초기 메쏘드를 HVC에 부착한다.

(4) initial_class와 target_class에 HVC의 초기 메쏘드를 호출하기 위해 Backward(obj, HVC,obj_attribute)함수를 가진 트리거를 붙인다. 이때 obj_attribute는 현재 수정한 객체의 클래스를 가리키는 HVC의 애트리뷰트의 이름이다. 아울러 트리거의 조건절은 CONSTRAINT^{cch}의 조건절과 논리적 보완(complement) 관계에 있다.

output : 역방향 전달을 지원할 수 있는 HVC 및 트리거 정의

다음은 그림 3의 Machine 클래스의 무게 무결성을 CONSTRAINT^{cch}를 이용하여 정의하고 이를 위의 과정을 이용하여 역방향 전달을 지원하기위한 HVC와 트리거로 전환한 예이다.

CONSTRAINT^{cch}를 이용한 정의 :

CONSTRAINT^{cch} ON Machine Part Material

AS MachineWeight() on obj〉=1000

(1) initial_class(Machine 클래스)와 target_class(Part, Material 클래스)로 부터 HVC의 애트리뷰트를 정의한다. 이때 m.components가 집합 클래스 구성계층 관계를 가지고 있으므로 다음에 오는 모든 애트리뷰트는 집합 형태의 정의역(setof(Part))을 가진다.

```
CREATE VCLASS hvc
{machineqr Machine,
  part setof(Part),
  material setof(Material)}
```

(2) HVC의 Query를 생성한다.

```
AS SELECT m, m.components, m.components.material_type
FROM Machine m
```

(3) 초기 메쏘드를 HVC에 첨부한다.

```
Method MachineWeight();
```

(4) initial_class와 target_class에 트리거 정의한다. 이때 HVC의 initial_class의 객체를 지칭하는 가상 클래스의 애트리뷰트에 초기 메쏘드가 적용되도록 한다. 아래의 정의는 Mate-

rial 클래스에 정의된 트리거이다.

```
TRIGGER machine_weight_material
AFTER UPDATE ON Material
IF MachineWeight() ON Backward(obj,
    hvc,material).machine>1000
INVALIDATE TRANSACTION;
```

위와 같이 제공한 방법을 따라 Machine 클래스의 무게에 관한 CONSTRAINT[cch] 정의로부터 생성된 HVC와 트리거는 클래스 구성계층상의 역방향 전달을 지원하는 무결성 제약조건을 지원한다.

6. 관련 연구

본 연구와 관련된 분야로 능동적 객체지향형 데이타베이스에 관한 연구를 들수 있다. 현재 객 체지향적 데이타베이스에서의 능동형 기능의 구현은 능동적 관계형 데이타베이스의 기능과 몇가지 점에서 다르게 진행되고 있다[ANWA93]. 특히 관계형 데이타베이스에는 없는 메쏘드를 능동적 객체지향적 데이타베이스에서는 트리거을 선언할 때 사용하려는 움직임이 있다. 하지만 이들 능동적 데이타베이스의 트리거 기능은 무결성 문제에 관하여는 기초적 기능을 제공하고 있으며, 단지 의미상의 차이로 트리거와 무결성 제약조건 생성자를 분리하고 있어서[Gehani & Jagadish 1992] 무결성을 정의하기 위한 능동형 개념의 적용에 관한 연구는 부족한 실정이다.

HiPAC(High Performanc Active database system)은 Smalltalk을 기반으로 하는 능동형 데이타베이스로써 많은 능동형 데이타베이스가 받아들이는 Event-Condition-Action(ECA) 모델을 사용하여 능동형 규칙을 선언하도록 하였다[McCarthy & Dayal 1989, Dayal et al. 1988]. 또 다른 연구로써는 AT&T Ball 연구소에서 개발되고 있는 Ode[Gehani & Jagadish 1991, Gehani & Jagadish 1992]를 들수 있다. 특히 이 연구에서는 기존의 능동형 기능을 트리거 생성자와 무결성을 정의하는 제약 생성자(co-nstraint constructor)로 분류하였으나 그 분류기준은 단지 두가지의 의미적 차이 때문이라고 밝히고 있다. UniSQL/X [UniSQL 1993]는 버전 2.0 부터 능동형 기능을 추가한 객체지향적 데이타베이스이다. 이 데이타베이스에서는 트리거를 자료정의시에 정의함으로써 능동형 기능을 제공한다. 이 능동형 기능에는 사용자가 메쏘드를 사용하여 규칙을 정의할 수도 있다.

위에서 언급한 능동적 객체지향적 데이타베이스에서 연구된 규칙을 데이타베이스 무결성 유지에 사용하고자 하는 연구는 기존의 데이타베이스 무결성 연구에서 현재 시작되고 있는 상태이다[Urban et al. 1992]. 기존의 데이타베이스의 무결성 연구는 논리(predicate logic)을 기반으로 관계형 데이타베이스에서의 무결성 연구가 주로 이루어졌다[Ullman 1988, Urban & Decambre 1988,]. 하지만 데이타베이스 연구의 중심이 객체지향적 데이타베이스로 옮기자 능동적 객체지향형 데이타베이스의 여러 가지 특징을 고려한 무결성 연구가 이루어지고 있다. Urban[1992]은 그의 연구에서 객체지향적 데이타베이스의 인캡슐레이션(encapsulation)을 고려한 무결성 연구를 하였

다. 아울러 몇몇 연구에서는 무결성 유지기능을 일반적 능동형 기능과 분리해야 한다고 주장하고 있다.

기존의 데이타베이스 무결성 연구에서는 클래스 구성계층에서의 무결성 제약조건 문제인 역방향 전달을 다루지는 않았지만 이와 유사한 개념이 객체지향적 프로그램 언어에서는 나타나고 있다. Smalltalk에서 사용자 환경등에서의 상호 관련을 위하여 객체간의 관련(Dependancy) 개념을 가지고 있다[Digitalk 1986]. 이 개념을 이용하므로써 한 객체의 변화가 다른 객체에 영향을 주어야 할 경우를 표현할 수 있다. 하지만 이 경우에는 본 연구에서와 같이 클래스 구성계층에만 국한되지는 않는다.

7. 결 론

현재의 능동적 객체지향형 데이타베이스의 트리거는 사용자 정의 무결성 제약조건을 적절하게 지원할 수 없다. 특히 트리거는 클래스 구성계층을 이루는 클래스들에 대한 제약조건 유지에 부적절하다. 그러므로 본 연구에서는 클래스 구성계층상에 정의된 제약조건을 지원할 수 있는 역방향 전달에 대한 개념과 방법을 제안하였다.

본 연구에서 제안한 역방향 전달방법인 "가상 클래스를 이용한 역방향 전달방법"은 능동적 객체지향형 데이타베이스의 가상 클래스, 메쏘드를 포함한 트리거 정의, 그리고 메시지 전달등의 기능을 통합하여 보다 효과적으로 역방향 전달을 지원하도록 하였다. 이 방법은 여러 가지 장점을 지니고 있다. 이들 장점으로는 코드 재사용, 무결성 검사를 위한 제한된 권한 부여, 효율적인 타 객체 참조등을 예로 들 수 있다. 이 방법을 기본으로 본 연구에서는 클래스 구성계층의 객체에 대한 무결성 제약조건을 정의하는 생성자인 CONSTRAINT[och]를 그 의미와 문법을 포함하여 정의되었다. 이 생성자는 사용자의 제약조건 정의로부터 역방향 전달에 필요한 가상 클래스, 메쏘드, 그리고 트리거를 생성하게 된다. 본 연구에서는 무결성 제약조건 정의시의 문제점, 역방향 전달의 개념, 역방향 전달방법, 그리고 새로운 생성자를 정의하는 도구로 기존의 능동적 객체지향형 데이타베이스[UniSQL 1993]를 사용 하였다. 아직 역방향 전달을 지원하는 데이타베이스는 보고되지 않았으며 본 연구에서 사용된 데이타베이스도 이를 지원하지 않고 있다. 그러므로 본 연구에서는 역방향 전달, 즉 클래스 구성계층에서의 제약조건을 지원하기 위하여 기존의 데이타베이스가 보강해야 할 문제점을 제안 하였다. 트리거를 중심으로 하는 능동적 기능을 무결성 제약조건에 응용하는 분야는 아직 많은 부분에서 연구를 필요로 한다. 이들 중에 앞으로 본 연구와 연관되어 시도되어야 할 분야는 클래스 계승계층을 대상으로 하는 무결성 제약조건의 특징 및 문제점에 관한 연구를 들 수 있다. 이는 클래스 구성계층에 관한 본 연구와 통합되면 객체지향적 데이타베이스의 두가지 대표적 계층구조상의 무결성 제약조건 지원 할 수 있는 완성도 있는 연구가 될 것이다.

참고문헌

Anwar, E., L., Maugis, and S. Chakravarty, "A New Perspective on Rule Support for Object-Oriented Databases," *ACM-SIGMOD 1993 Int'l Conf. Management of Data*,Washington, DC, USA, May 1993.

Dayal, U., B. Blaustein, A. Buch-mann, and U. Chakravarthy, "The HiP-ACProject: Combining Active Databas-e and Timing Constraints," *ACM SIGM-OD RECORD*, Vol. 17, No. 1, March 1988.

Digitalk, Inc., *Smalltalk/V Tutorial and Programming Handbook*, 1986.

Gehani, N. and H.V. Jagadish, "Ode as an Active Database: Constr-aint and Triggers," *Proc. the 17th Int'l Conf. on VLDB*, Barcelona, September 1991.

Gehani, N. and H.V. Jagadish, "Event Specification in an Active Obj-ect-Oriented Database," *ACM-SIGMOD 1992 Int'l Conf. Management of Data*, CA, USA, June 1992.

Kim, W., *Introduction to Object-Oriented Database*, MIT Press, 1990.

McCarthy, D. R. and Umeshwar Dayal, "The Architecture Of An Active DataBase Management System," *Proc. ACM-SIGMOD 1989 Int'l Conf. Management of Data*, Portland, Oregon, May-June 1989.

Stonebraker, M., A. Jhingran, J. Coh, and S. Potamianos, "On Rules, Procedures, Caching and Views in Data Base Systems," *ACM-SIGMOD 1990 Int'l Conf. Management of Data*, 1990.

Ullman, J.D., *Principles of Database and Knowledge - base Systems*, Vol.1, Computer Science Press, 1988.

UniSQL, Inc., *UniSQL/X User's Manual Release 2.0*, 1993.

Urban, S.D. and L.M.L. Decambre, "Constraint Analysis: A Tool for Explaining the Semantics of Complex Objects," *Proc. 2nd Int'l Workshop on Object-Oriented Database Systems*, Bad Munster am Stein-Ebernburg, FRG, 1988.

Urban, S.D., A.P. Karadimce, and R.E. Nannapaneni, "The Implementation and Evaluation of Integrity Maintenance Rules in an Object-Oriented Database," *Proc. Int'l Conf. Data Engineering*, 1992.