객체지향 데이타모델에서 다중인스턴스화 지원을 위한 버전개념 확장

(Extension of Version Concept for Polyinstantiation in Object-Oriented Data Model)

서울대학교 컴퓨터공학과 이상원

서울대학교 컴퓨터공학과 김형주

요약

이 논문은 다단계보안 객체지향 데이타모델 MORION을 제시한다. 첫째, 객체지향 데이타모델에서 올바른 정보 흐름을 보장하는 보안정책들을 제시했다. 둘째, 다중인스턴스화 현상을 지원하기위해 새로운 의미관계 **poly-from**을 정의하고 이를 객체지향 데이타모델에서 자연스럽게 지원하기 위해 버전 개념을 확장했다. 이와 관련하여 두가지 새로운 보안정책을 제시하고 버전들사이의 효율적인 기억장소 공유방안을 언급했다. 마지막으로, 메소드를 이용한 보안정책 구현을 다루었다. MORION의 장점은 객체지향 데이타 모델의 버전과 메소드 개념을 확장해서 다단계보안을 자연스럽게 지원하는 것이다.

Abstract

We propose a multilevel secure object-oriented data model, MORION. First, we provide several security policies guaranteeing legal information flows in object-oriented data model. Second, in order to support polyinstantiation efficiently, we identify a new semantic relationship, *poly-from*, extend the *version* semantics. We also provide two new security policies and discuss an efficient

storage sharing mechanism for polyinstantiated versions. Finally, we use methods as implementation mechanism of security policies. It is advantage of MORION that it fully utilizes version and method concept to support multilevel security.

1 서론

데이타베이스는 여러명의 사용자가 접근하기 때문에, 정보의 보안을 위해서 DBMS는 정보에 대해 정당한 자격이 있는 사용자만이 정보를 접근하도록 하는 접근제어 장치를 반드시 갖추고 있어야 한다. 접근제어 장치의 기능은 다음과 같은 추상적인 모델로 표시할수 있다.

$$\mathbf{F(S,O,T) = yes\ or\ no}$$

여기서 S, O, T는 각각 사용자, 데이타, 접근 형식(읽기, 쓰기, 변경 등)을 나타낸다. 함수 F로 표시되는 접근제어 장치의 기능은 사용자 S가 데이타 O에 대해 T형식의 접근을 요청할때, 이의 허용여부를 결정하는 것이다.

데이타베이스 시스템의 접근제어 모델은 크게 임의적 접근제어(discretionary access control)와 강제적 접근제어(mandatory access control)로 분류할 수 있다. 임의적 접근제어는 함수 F의 값의 결정을 사용자의 권한을 표시한 자료구조, 예를 들어 접근 행렬이나 접근제어 리스트 등을 참조하여 결정한다. 반면에 강제적 접근제어는 사용자와 데이타 각각에 부여된 보안등급을 기반으로 하여 사용자의 접근을 제어한다. 강제적 접근제어는 사용자와 데이타에 부여할 수 있는 보안등급이 여러개가 있다는 의미에서 다단계보안(multilevel security)이라고도 한다. 이 글에서는 강제적 접근제어라는 말대신 다단계보안이라는 용어를 사용하겠다.

현재의 대부분의 관계형 DBMS에서는 릴레이션, 뷰, 애트리뷰트 단위까지의 임의적 접근제어를 지원하고 있다. 관계형 DBMS에서 취하고 있는 이 동적 권한 관리 방법은 [5]에 자세히 설명되어 있다. 관계형 데이타모델을 확장하여 다단계보안을 지원하는 대표적인 연구로서는 SeaView 모델 [11]과 LDV 모델 [15]이 있다.

데이타베이스의 응용의 범위가 은행, 행정업무에서 CAD, A.I, Multimedia의 영역으로 확대되면서 기존 관계형 DBMS는 새로운 응용의 환경에 여러가지 면에서 부적당하다[1]. 이러한 단점을 극복하기 위해서 관계형 데이타모델을 확장하거나, 복합 객체, 여러가지 의미상의 관계, 임의의 자료형 정의를 가능하게 해주는 객체지향 데이타모델이 나왔는데, 이 중의 하나인 ORION에서는 임의적 접근제어를 지원하고 있으며([14]), 강제적 접근제어 모델로는 ORION을 확장한 SORION 모델([16])이 있다.

표 1는 위에서 설명한 데이타베이스의 접근제어 상황을 분류한 것이다.

	RDBMS	OODBMS
임의적 접근제어	대부분의 RDBMS들	ORION
강제적 접근제어	SeaView LDV	SORION 모델 MORION 모델

표 1: 데이타베이스 접근제어 분류

본 논문은 ORION 모델을 확장한 다단계보안 객체지향 데이타모델 MORION[1](Multilevel ORION)을 제시한다. 다단계보안의 개념과 ORION 객체지향 데이타모델을 2장에서 간단히 살펴본다. 3장에서 MORION의 모델을 구성요소별로 설명하는데, 객체지향 데이타베이스에서 가능한 정보의 흐름들을 분석하고, 잘못된 정보흐름을 방지하기 위한 보안정책을 열거한다. 4장에서는 객체지향 데이타모델에서 다중인스턴스화 문제의 해결을 위해 새로운 의미 관계 **poly-from**을 정의하고, 기존 객체지향 개념을 이용해서 poly-from 관계를 자연스럽게 지원하기 위해 버전 개념을 확장한다. 5장에서는 보안정책을 구현하는 도구로 객체지향 데이타모델의 시스템 정의 메소드의 역활을 확장한 알고리즘과 poly-from 관계를 고려한 버전들사이의 기억장소 공유 방안을 다룬다. 6장에서 관련연구와 비교를 하고, 7장에서 결론과 앞으로의 연구방향을 논의한다.

2 다단계보안 및 객체지향 데이타모델의 개요

2.1 다단계보안 모델과 다중인스턴스화

다단계보안 모델 다단계보안 모델에서는 모든 데이타와 사용자에게 보안등급(security level)을 부여한다. 사용자 s의 보안등급을 $L(s)$로 표시하고, 데이타 o의 보안등급을 $L(o)$로 표시한다. 모든 보안등급은 두가지의 구성요소, $<$레벨(sensitivity level), 범주(categories)$>$로 이루어져 있는데, 데이타에 부여된 레벨을 분류(classification) 레벨, 사용자에게 부여된 레벨을 허용(clearance) 레벨이라 한다. 보안등급의 레벨들간에는 계층적인 순서관계가 성립한다. 예를 들어, 순서관계

[1]MORION은 투구의 일종임.

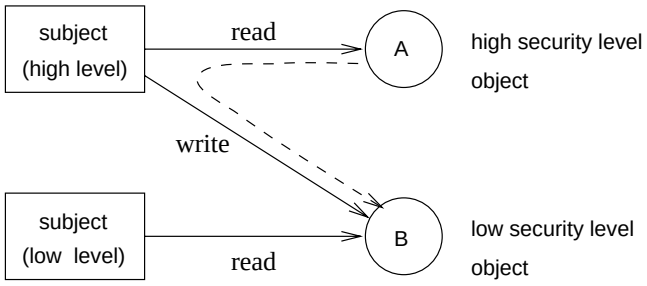

그림 1: 잘못된 정보흐름의 예

$U(unclassified) < C(confidential) < S(secret) < TS(top-secret)$가 성립한다. 범주는 다른 용어로 컴파트먼트(compartment)라고도 하는데, 데이타나 사용자가 관련한 프로젝트들의 집합으로 이루어져 있다. 따라서, 보안등급들은 격자구조(lattice)를 형성한다.

대표적인 다단계보안 모델로는 Bell-LaPadula 모델[3]이 있는데, 이 모델에서는 데이타의 보안을 위해, 사용자가 데이타를 접근할 때 반드시 만족되어야 하는 다음 두가지의 중요한 성질을 요구한다.

1. 단순 보안 성질: 사용자 s는 데이타 o에 대해, $L(o) \leq L(s)$일때만 읽기접근을 할 수있다.

$$F(s, o, R) = yes \text{ iff } L(s) \geq L(o)$$

2. $*$-성질: 사용자 s는 다른 데이타 o에 대해 $L(s) \leq L(o)$일 때만 쓰기접근을 할수 있다.

$$F(s, o, W) = yes \text{ iff } L(s) \leq L(o)$$

첫번째 성질은 사용자보다 높은 보안등급을 가진 데이타에 대한 접근을 방지하기 위한 것이다. 두번째 성질은 그림 1과 같이, 높은 보안등급을 가진 객체의 정보를 낮은 보안등급을 가진 객체에 옮기므로서 일어나는 잘못된 정보 흐름을 방지하기 위한 것이다.

다중인스턴스화 다중인스턴스화(polyinstantiation)란, 다단계 데이타를 취급할 때 데이타의 값은 같으면서 보안등급을 달리하는 여러개의 데이타가 동시에 존재하는 것을 일컫는다. 다단계 데이타를 취급할때 다중인스턴스화는 불가피한 현상이다([10]).

다중인스턴스화의 종류는 다중인스턴스화된 개체, 다중인스턴스화된 애트리뷰트의 두가지로 나눌수 있다. 다중인스턴스화된 개체는 하나의 개체를 유일하게 결정하는 애트리뷰트(예: 릴레이션의 주키(primary key))의 값을 이미 높은 보안등급의 사용자가 어떤 개체의 값으로 할당한

뒤에, 낮은 등급의 사용자가 그 사실을 모르고 낮은 등급의 개체의 값으로 할당할 때 발생한다. 다중인스턴스화된 애트리뷰트는 하나의 개체의 특성을 묘사하는 애트리뷰트의 값이 서로 다른 보안등급상에서 틀린 경우이다.

2.2 객체지향 데이타모델

객체지향 데이타모델에서는 현실 세계의 모든 개념적인 개체(entity)들을 객체로 모델링하고, 이 객체들간의 정보의 교환은 메시지를 통해서 이루어진다. 비슷한 성질을 갖는 객체들의 공통적인 성질을 모아서 클래스를 정의하고, 이렇게 정의된 클래스들은 클래스 계층구조를 이룬다. 여기서는 ORION 객체지향 데이타모델의 중요 개념들을 앞으로 논문의 전개에 필요한 범위에서 설명한다. 이들 개념은 참고문헌 [9]에 자세히 설명되어 있다.

객체 객체는 관계형 데이타베이스의 릴레이션의 한 튜플과 비슷한데, 구조를 나타내는 인스턴스 변수(instance variable)들과 동작을 나타내는 메소드(method)들로 이루어지는데, 이는 기존의 데이타가 단순히 수동적인 성질을 갖는 반면에 객체의 메소드가 객체의 인스턴스변수에 대한 접근 수단이기 때문에 객체는 능동적인 성질을 지니고 있다. 즉, Bell-LaPadula 모델의 사용자와 데이타의 개념이 객체에 모두 포함되어 있는 것이다.

메시지와 메소드 메시지(message)는 하나의 객체($o1$)에서 다른 객체($o2$)의 메소드를 실행하기 위해 보내어지는데[2], 모든 메시지는 다음과 같은 구성요소를 가진다.

(선택자, 수신자, 메시지 인자)

수신자(receiver)는 메시지를 받는 객체, 즉 $o2$를 가리키고, 선택자(selector)는 메시지가 호출하는 수신자의 메소드 이름이다. 메시지 인자는 선택자에 해당하는 메소드가 필요로 하는 인자이다.

클래스 모든 객체가 인스턴스 변수의 이름, 자료형, 값, 메소드의 코드를 포함한다면, 데이타베이스의 기억장소에 대한 유지경비(overhead)가 너무 크다. 그리고, 특정 객체를 생성하고자 한다면, 어느 객체에 새로운 객체 생성을 위한 메시지를 보낼 것인가 하는 문제가 있다. 따라서, 객체

[2] $o1 = o2$ 일 수도 있다.

지향 데이타모델에서는 공통된 성질, 즉 같은 인스턴스 변수와 메소드를 갖는 객체들을 위해 클래스 개념을 도입했다.

상속 객체지향 데이타모델에서 상속(inheritance)의 개념은 이미 정의된 클래스(들)에서 새로운 클래스를 유도할 수 있게 해준다. 이때 새로운 클래스는 기존 클래스의 모든 성질, 즉 인스턴스 변수들과 메소드들을 물려받거나 물려받은 인스턴스 변수나 메소드를 재정의할 수도 있다. 그리고, 새로 정의하는 클래스에 필요한 인스턴스 변수나 메소드를 추가할 수도 있다. 이때 기존의 클래스를 상위클래스(superclass)라 하고, 상속을 통해 새로 정의한 클래스를 하위클래스(subclass)라 한다.

버전 CAD, CAM과 같은 차세대 데이타베이스 응용환경에서는 하나의 설계 객체로부터 다른 버전(version)을 유도해서 작업을 할 수 있는 기능이 요구된다. 예를 들어, CAD 툴 사용자는 특정 기능을 수행하는 칩을 설계한 뒤에 이 칩 설계를 여러가지 방법으로 변경해서 테스트를 하기를 원하는 경우가 있다. 이때 원래 설계와 다른 변형 설계들은 사용자가 설계하고자 하는 특정 기능을 수행하는 칩의 일종이다. 이때 원래 설계와 다른 변형 설계 모두 사용자가 설계하고자 한 칩의 버전들이다. 기존 관계형 DBMS를 이용하는 경우 응용프로그램 자체내에서 버전관리를 해야한다. 이러한 응용프로그램의 부담을 줄이기 위해 몇몇 객체지향 DBMS에서는(예를 들어, ORION, IRIS 등) 버전을 데이타모델의 일부로 포함시켜 DBMS내에서 버전관리를 지원한다.

ORION 모델에서는 클래스를 정의할때 클래스의 인스턴스 객체들의 버전화가능 여부를 명시한다. 버전화가능 클래스의 객체를 생성하면, 생성객체 v와 처음 버전 v_1이 생겨난다. 사용자는 이미 생성된 임의의 버전 v_i로부터 새로운 버전 v_j를 derive-version이라는 명령을 통해 생성할 수 있다. 이때 생성객체 v와 버전들은 의미적 관계 **version-of**에 있고, v_i와 v_j사이에는 의미적 관계 **derived-from**이 성립한다. 하나의 버전화가능 객체는 생성객체와 버전들의 집합으로 이루어진다. 버전화가능 객체의 각 버전들사이의 유도관계를 그림으로 표시하면 버전 계층구조(version hierarchy)가 된다.

그림 2는 버전화가능 객체 v의 생성객체(generic object)와 버전들의 버전계층구조를 묘사하고 있다. 생성객체는 v의 각 버전들의 버전계층구조에 대한 정보와 기타의 정보를 유지한다. 실선은 하나의 버전 v_j가 다른 버전 v_i로부터 유도되었다는 **derived-from** 관계를 나타내고, 점선

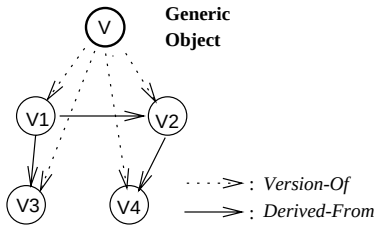

그림 2: 버전 계층구조의 예

은 버전 객체가 버전화가능 객체의 버전임을 표시하는 **version-of** 관계를 나타낸다.

3 다단계보안 객체지향 데이타모델 MORION

이 논문에서 제시하는 MORION은 보안등급들의 집합 S, 보안등급을 갖는 객체들의 집합 O, 그리고 객체에 보안등급을 부여하고 객체에 대한 접근을 제어하는 보안정책의 집합 P의 세가지 구성요소로 이루어져 있다. 특히, 보안정책은, 객체지향 데이타모델의 기본적인 특징이자 정보흐름의 수단인 메시지 전달, 상속, 버전을 고려해서, 객체지향 데이타베이스에서 발생하는 잘못된 정보흐름을 방지한다. MORION은 가장 큰 특징인 다중인스턴스화를 고려한 버전개념의 확장과 이와 관련한 보안정책은 다음 장에서 설명한다. 보안정책의 구현은 시스템 정의 메소드를 이용하는데, 5장에서 자세히 설명한다.

3.1 보안등급과 객체

MORION의 보안등급들의 집합 S는 다른 다단계보안 모델과 마찬가지로 격자구조를 이룬다. 여기서는 설명의 편의를 위해 보안등급들의 집합이 순서관계에 있는 것으로 가정한다.

모델의 두번째 구성요소는 보안등급을 갖는 객체들의 집합 O인데, 이 경우의 객체는 인스턴스 객체, 클래스 객체, 생성객체 그리고 버전 모두를 가리키는 포괄적인 의미의 객체이다. 모든 객체들의 집합을 O라 하고 보안등급의 집합을 S라 할 때, 다음의 조건을 만족하는 함수 L이 있다.(앞으로는 객체 o의 보안등급은 $L(o)$로 표시한다.)

$$L : O \longrightarrow S$$

MORION은 객체의 인스턴스 변수들의 보안등급은 동일하다는 의미에서 단일등급의 객체를 지원한다.

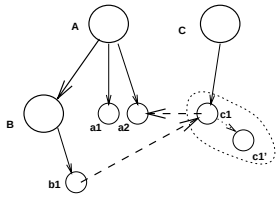

그림 3: 객체지향 데이타베이스에서의 다양한 정보흐름의 예

3.2 정보흐름과 보안정책

여기서는 보안정책들의 집합 P에 대해 설명하는데, 보안정책은 객체지향 데이타베이스내의 잘못된 정보흐름을 막는 역활을 한다. 우선 객체지향 데이타모델에서 가능한 정보흐름의 형태를 메시지 전달, 상속, 버전과 관련하여 살핀 후에, 이를 근거로 보안정책을 기술하겠다.

객체지향 데이타모델에서의 정보흐름

객체지향 데이타모델에서는 객체의 메소드를 통한 캡슐화(encapsulation) 특성때문에 메시지 전달, 상속([6, 12]), 그리고 새로운 버전 유도([2])를 통해서 정보흐름이 발생한다. 그림 3은 객체지향 데이타베이스에서의 다양한 정보흐름들의 예들을 보여주고 있다. 클래스 B는 클래스 A의 하위클래스이고, C는 버전화가능 클래스이다. a1, a2, b1, c1, c2는 인스턴스 객체들이다.

그림 3에서 객체 $c1$에서 객체 $a2$로의 굵은 점선은 메시지 호출을 의미하는데, 화살표방향으로 수신자인 객체 $a2$로의 메시지 인자가 전달되는 순방향 정보흐름이 발생하고, 반대방향으로 결과값이 메시지 호출 객체 $c1$에 전달되는 역방향 정보흐름이 있다. 객체 $b1$에서 객체 $c1$으로의 굵은 점선도 마찬가지다. 이와 같이 메시지 인자나 결과값을 통한 정보흐름을 명시적 정보흐름(explicit information flow)이라 한다. 클래스 객체에 객체 생성 메시지를 보내 새로운 객체를 생성하는 경우, 메시지 인자값이 생성되는 객체의 인스턴스 변수들의 값으로 전달되므로 순방향 정보흐름으로 간주한다

반면에 클래스 A에서 상속을 통해 클래스 B를 생성하는 경우, 클래스 A의 인스턴스 변수와 메소드의 정보가 하위클래스 B로 전달된다. 또, 그림 3에서 버전화가능 클래스 C의 한 객체 $c1$에서 다른 버전 $c1'$를 유도하는 경우 $c1$의 인스턴스 변수의 값이 새로운 버전 $c1'$로 전달된다[2]. 이 두가지의 정보흐름은 모두 암시적 정보흐름(implicit information flow)이라고 볼수 있다.

객체 $o1$에서 객체 $o2$로의 정보흐름이 있고, 다시 $o2$에서 제삼의 객체 $o3$로의 정보흐름이 있

는 경우, 이행적 정보흐름(transitive information flow)라 한다. 객체 $b1$에서 객체 $c1$를 거쳐 객체 $a1$로 메시지 호출이 일어날 경우 인자를 통해 $b1$에서 $a1$으로 이행적 정보흐름이 일어난 것이다.

시스템변수 A.L

시스템변수 A.L(Active Level)은 객체지향 데이타베이스의 동작중에 실행되는 메소드의 보안등급을 나타낸다. 이 A.L의 값은 제일 처음 메시지의 호출을 일어나게 한 사용자의 보안등급과 같다. 이 시스템변수 A.L은 객체지향 데이타베이스내에서 정보의 흐름을 제어하는데 중요한 역할을 한다. 시스템변수 A.L을 도입하게 된 이유는, 특정 사용자의 메시지호출로 인한 객체지향 데이타베이스의 연산이 사용자와 다른 보안등급을 갖는 객체들에게 영향을 주지않고, 사용자의 보안등급보다 낮거나 같은 보안등급을 가진 객체의 정보만을 사용자에게 보여주기 위해서이다.

보안정책

객체지향 데이타베이스내의 임의의 두 객체 o_i와 o_j에 있어, o_i로부터 o_j로의 정보흐름이 있을 경우, $L(o_i) \leq L(o_j)$의 조건을 만족해야 한다. 여기서 설명할 여러 보안정책들은 앞에서 살펴본 객체지향 데이타베이스내의 모든 종류의 정보흐름에 있어서 이 조건을 만족시키기 위한 것이다.

클래스 객체가 다른 객체(대개는 사용자 객체)로부터 객체 생성 메시지를 받아서 객체 생성 메소드를 수행하므로써 객체가 생성된다. 객체는 보안등급을 가지기때문에 객체의 생성 메시지 인자는 새로이 생성되는 객체의 보안등급을 위한 인자를 포함한다. 이때 새로이 생성되는 객체의 보안등급은 반드시 A.L보다 높거나 같아야 한다. 이를 형식화해서 다음 보안정책으로 나타낼 수 있다.

보안정책 1 (객체생성) 객체생성 메시지의 인자중 생성될 객체의 보안등급을 위한 인자의 값이 s라 할 때, $A.L \leq s$를 만족해야 한다.

다음 보안정책 2는 인스턴스 객체와 클래스 객체의 보안등급사이의 관계를 나타내는데, 이는 사용자가 인스턴스 객체의 구조와 메소드에 관한 정보를 클래스 객체로부터 얻기 때문에 필요하다.

보안정책 2 (인스턴스 성질) 클래스 객체를 C라 하고, 이 클래스 객체에 속하는 임의의 인스턴스 객체를 o라 할때, $L(C) \leq L(o)$의 관계가 성립해야 한다.

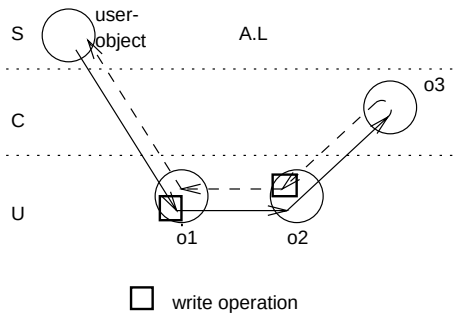

그림 4: 보안정책 4와 관련한 잘못된 정보흐름

보안성질 2는 다른 말로 메타-데이타[3]성질이라고 할수 있다. 왜냐하면, 데이타베이스에서 어떤 데이타가 사용자에게 이용가능하다면, 그 데이타의 메타-데이타도 역시 이용가능한 것이 합리적이다.

다음 보안정책 3은 읽기 연산시 A.L이 객체의 보안등급보다 낮은 경우, 역방향 정보흐름과 이행적 정보흐름에 의한 잘못된 정보흐름을 막기 위한 것이다.

보안정책 3 (단순보안성질) 객체 o의 인스턴스 변수에 대한 읽기 접근은 $L(o) \leq A.L$인 경우에만 허용한다.

다음의 보안정책 4는 버전화불가능 객체에 대한 쓰기 연산을 제어하는 보안정책이다.(버전화가능 객체에 대해서는 보안정책 7에서 다룬다.)

보안정책 4 (*-성질) 어떤 클래스가 버전화가능하지 않을때, 그 클래스에 속하는 한 객체 o의 인스턴스 변수의 값을 갱신하고자 하면, $L(o) = A.L$인 경우에만 허용한다.

$L(o) < A.L$인 경우에 갱신을 허용하면, 그림 4에서처럼 잘못된 정보흐름이 발생할수 있다. 그림에서 실선과 점선은 각각 메시지 전달에 의한 순방향 정보흐름과 역방향 정보흐름을 나타내고, 굵은 사각형은 쓰기 연산을 나타낸다. 그림에서 굵은 사각형이 표시된 두 곳은 사용자 객체로부터 메시지를 받은 객체 $o1$이 쓰기 연산과, 객체 $o3$에서 객체 $o2$를 거쳐 객체 $o1$에, 즉 이행적 정보흐름에 의해 결과값이 전달되었을때 쓰기 연산은 잘못된 정보흐름이다.

[3]메타-데이타(meta-data)란 "데이타에 관한 데이타"란 의미인데, 예를 들어 관계형 데이타베이스 시스템에서 릴레이션들의 이름, 애트리뷰트의 이름, 도메인, 제한조건을 한곳에 모아둔 시스템 카타로그가 있다.

한편, Bell-LaPadula 모델에서의 $*$-성질은 사용자의 보안등급이 데이타의 보안등급보다 낮은 경우, 즉 $L(o) > A.L$일때에도 쓰기를 허용하고 있는데 여기서 제시하는 모델에서는 이를 금지하고 있다. Bell LaPadula 모델은 보안(secrecy)의 측면만 다루고 무결성(integrity)은 고려하지 않고 있는데, 보안정책 4에서는 무결성을 위해서 write-equal을 택했다.

다음 보안정책은 상속시 발생하는 암시적 정보흐름을 제어하기 위한 것이다.

보안정책 5 (상속) 하나의 클래스 객체 C_i에서 상속을 통해 새로운 클래스 객체 C_j를 상속할 때, $L(C_i) \leq L(C_j)$의 관계가 성립해야 한다.

이 보안성질 5는 잘못된 정보흐름을 방지하는 측면이외에, 객체지향 데이타모델의 상속 개념을 다단계보안 객체지향 데이타모델에서도 유지하기 위해 필요하다. 상속받은 메소드를 하위클래스에 중복적으로 저장하지 않고, 상속받은 메소드 실행을 위해 상위클래스를 탐색해야 하는 경우 상위클래스의 보안등급이 높다면 상위클래스의 메소드를 이용할수 없으므로 상속 메카니즘이 파괴된다.

버전화가능 클래스 경우, 버전 객체 v_i에서 다른 버전 객체 v_j를 유도할때 v_j는 v_i로부터 인스턴스 변수들의 값을 상속받는데, 이와 같은 암시적 정보흐름을 제어하기 위해, 다음의 보안정책이 필요하다.

보안정책 6 (버전) 어떤 클래스가 버전화가능할때, 그 클래스에 속하는 객체의 한 버전 v_i에서 새로운 버전 v_j를 유도할때, $L(v_i) \leq L(v_j)$의 관계가 성립해야 한다.

4 다중인스턴스화를 위한 버전의 의미 확장

지금까지는 기존의 ORION 객체지향 데이타모델의 구성요소들과 관련한 다단계 보안정책들을 알아보았다. 여기서는 다단계보안 모델에서 불가피한 다중인스턴스화 문제를 ORION 모델에서 자연스럽게 해결하기 위한 확장된 버전 개념을 제시한다. MORION은 객체지향 데이타모델에서 다중인스턴스화를 해결하기 위해 버전개념을 확장을 구체적으로 다룬 최초의 논문이다.

앞에서 다중인스턴스화의 종류에는 다중인스턴스화된 개체와 다중인스턴스화된 애트리뷰트 두가지가 있음을 설명했다. 객체지향 데이타모델에서는 객체 식별자가 유일하게 객체를 결정하기 때문에 다중인스턴스화된 개체의 문제는 발생하지 않는다[12]. 그러나, 다중인스턴스화된

애트리뷰트 문제는 발생한다. 이의 해결을 위해 새로운 의미적 관계 **poly-from**을 정의하고, ORION 모델의 버전 개념의 확장과 이와 관련한 새로운 두가지의 보안정책을 제시한다.

4.1 의미적 관계 poly-from

버전화불가능 클래스의 경우에 있어 하나의 개체를 모델링하는 낮은 보안등급의 객체 o1의 특정 인스턴스 변수 a의 값을 높은 보안등급의 사용자가 자신이 아는 값으로 수정하려는 경우 보안정책 4는 이를 금지한다. 따라서, 사용자는 이를 위해 새로운 객체 o2를 생성해서 변경된 인스턴스 변수의 값을 갖게 한다[4] 이때 객체 o2의 보안등급은 사용자 객체의 보안등급과 같다. 객체 o1과 o2는 인스턴스 변수 a의 값만 틀리고, 의미적으로 **poly-from**의 관계에 있다. 그런데, 두 객체사이의 poly-from 관계를 시스템에서는 관리를 해주지않기 때문에 사용자가 이를 관리해야 한다. 만일에 객체 o1을 생성한 사용자가 o1을 삭제한 경우에, 의미적으로 o1과 o2가 같은 개체를 모델링하는 객체이기 때문에 o2도 지워야 하는데 이는 전적으로 사용자에게 달려있다[5].

4.2 poly-from 지원을 위한 버전 개념의 확장

주의해서 볼점은 **poly-from** 관계에 있는 객체들과 버전화가능 객체의 버전들은 실세계의 하나의 개체를 모델링한다는 점에서 유사한 점이 있다. 이 생각을 기반으로 해서 MORION 모델에서는 poly-from 관계를 객체지향 패러다임에서 자연스럽게 지원하기 위해 ORION 모델의 버전의 개념을 확장했다. 즉, 응용프로그램에서 요구되는 기존의 version-of, derived-from 관계와 다중인스턴스화를 위해 새로이 필요한 poly-from 관계를 모두 버전이라는 의미 모델링 개념을 통해서 지원하는 것이다.

우선 이해를 돕기위해 poly-from과 derived-from 관계를 동시에 포함하고 있는 버전화가능 객체의 예를 보자. 그림 5은 derived-from 관계와 poly-from 관계를 포함하고 있는 버전 계층구조의 예를 보여주고 있는데, 버전 v2, v5, v7은 각각 버전 v1, v2, v4와 derived-from 관계에 있고, 버전 v3, v4, v6, v8 각각은 버전 v1, v1, v5, v4와 poly-from 관계에 있는 버전들이다. v2, v5등은 사용자가 derive-version 명령을 이용해서 새로운 버전을 유도한 결과이고, v3, v4등은 높은 보안등급

[4]다단계보안 관계형 데이타모델인 SeaView에서는 다중인스턴스화된 애트리뷰트를 새로운 튜플로 모델링하는 것과 유사하다. 차이점은 SeaView 모델에서는 시스템에서 자동적으로 새로운 튜플을 생성한다는 점이다.

[5]SeaView의 경우에 있어서는 시스템이 자동적으로 다중인스턴스화된 애트리뷰트를 모델링하는 튜플을 삭제한다.

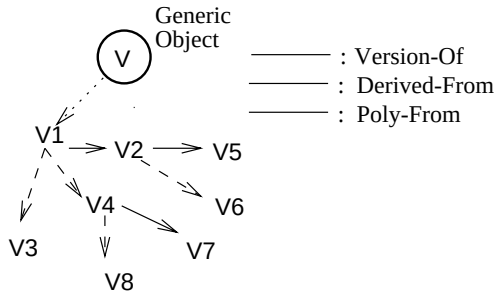

그림 5: derived-from 관계와 poly-from 관계를 포함하는 버전 계층구조

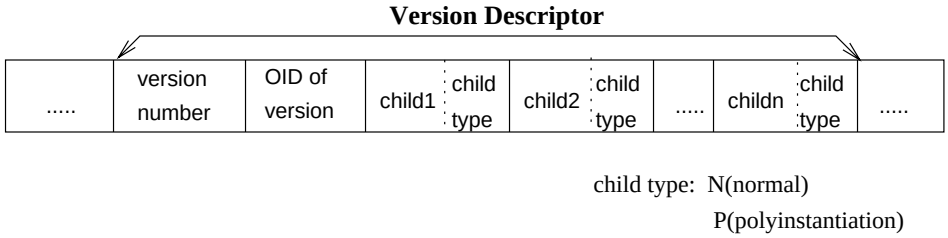

그림 6: **poly-from** 관계를 위해 확장된 생성객체의 자료구조

을 가진 사용자가 인스턴스 변수의 값을 고칠때 다중인스턴스화된 애트리뷰트를 모델링하기 위해 시스템에서 자동적으로 생성한 버전이다.

원래 ORION에서는 버전화된 객체의 버전들사이의 derived-from 관계를 나타내는 버전계층구조를 생성객체의 자료구조, 버전 유도 계층구조(version derivation hierarchy)를 이용해서 관리한다. 버전 유도 계층구조에서는 각각의 버전에 대해 하나의 버전서술자를 갖고 있다. 각각의 버전서술자는 해당 버전의 버전 번호와 버전 객체의 객체 식별자 그리고 이 버전으로부터 유도된 다른 버전들의 버전번호들의 집합으로 구성된다. MORION에서는 다중인스턴스화를 위한 **poly-from** 관계를 **derived-from** 관계와 구분하기 위해 버전 서술자(version descriptor)의 children을 버전 번호와 유도관계를 구별할수 있는 플래그의 집합으로 확장한다. 그림 6은 ORION의 생성객체의 구조를 확장한 것이다. 버전계층구조의 한 노드는 하나의 버전을 나타내는데, 이는 설명자를 이용한다. 플래그 child type은 두가지의 값을 가질수 있는데, N은 버전간의 관계가 **derived-from**임을 나타내고, P는 **poly-from**의 관계를 나타낸다. poly-from의 관계를 고려하지 않은 경우에는 하나의 버전으로부터 유도된 다른 버전들이 하나의 의미적 관계 derived-from만 있기때문에 child의 집합으로만 표시하면 된다.

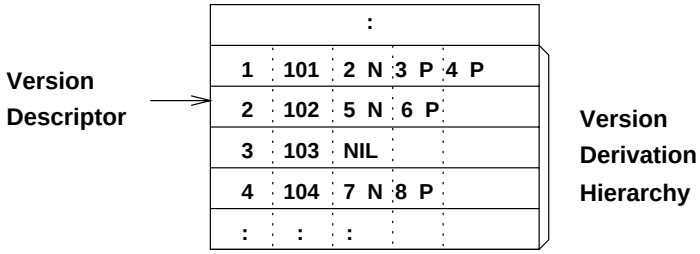

그림 7: 생성객체의 자료구조의 예

그림 7은 그림 5의 버전화가능 객체의 생성객체의 버전 유도 계층구조를 보여주고 있다.

4.3 다중인스턴스화 관련 보안정책

여기서는 다중인스턴스화된 애트리뷰트(MORION의 용어로는 **poly-from** 관계)의 모델링과 관련한 두가지의 보안정책을 제시한다. 보안정책 7는 버전하가능 객체의 인스턴스 변수의 값을 높은 보안등급의 사용자가 다른 값으로 변경하고자 할때 적용되는 정책이다. 이는 사용자가 명시적으로 derive-version 명령을 사용해서 derived-from의 관계를 갖는 버전을 유도할때 필요한 보안성질 6과는 구별해야 한다.

보안정책 7 (다중인스턴스화 규칙) 어떤 클래스가 버전화가능할때, 이 클래스에 속하는 어떤 객체 v_i의 보안등급보다 높은 $A.L$을 가지고 인스턴스 변수의 값을 갱신하고자 하면, 보안등급이 $A.L$인 새로운 버전 v_j를 생성해서 해당 인스턴스의 변수의 값을 고친다. 이때 v_i와 v_j는 **poly-from**의 관계이다.

기존 Bell-LaPadula 모델에서 높은 등급의 사용자가 낮은 등급의 데이타의 값을 바꾸지 못하게 하는 $*$-성질은 정보를 고의적으로 유출하려고 하는 의도가 없는 사용자의 낮은 보안등급의 데이타에 대한 갱신을 제한하는 단점이 있는데, 보안성질 7는 클래스를 버전화가능하게 정의하므로서 이런 단점을 극복하게 해준다.

어떤 클래스가 버전화가능인 경우, 보안성질 7에 의해 같은 개념적인 개체를 모델링하면서 서로 다른 보안등급상에서의 정보를 표현하기 위해, 즉 poly-from의 관계에 있는 버전화된 객체 v_i들이 있을수 있다. 이 경우 낮은 보안등급의 A.L 상태에서 하나의 버전을 삭제하면, 이 버전과 poly-from의 관계에 있는 모든 버전들도 함께 삭제한다. 객체 v_i들은 하나의 개념적인 개체를 모

델링하고 있기 때문에 하나의 버전이 삭제되면, 나머지 버전들도 삭제되는 것이 합리적이다. 이를 다음과 같은 보안성질로 정리할 수 있다.

보안정책 8 (연쇄삭제) 어떤 클래스가 버전화가능할때, 그 클래스에 속하는 어떤 객체 o_i의 보안등급가 같은 $A.L$을 가지고 해당 객체를 삭제하고자 한다면, 해당 객체뿐만 아니라, **poly-from**의 관계에 있는 버전 v_i 모두 삭제한다.

예를 들어, 그림 5에서 v1을 삭제할 경우 v1과 직접 poly-from 관계에 있는 v3과 v4는 물론, v4와 poly-from 관계에 있는 v8도 함께 삭제한다. 다단계보안 관계형 데이타모델의 경우에도 다중인스턴스화를 위해 생성된 튜플들의 경우 낮은 등급의 튜플이 생성되면 그 튜플과 다중인스턴스화의 관계에 있는 튜플들이 동시에 삭제된다([7, 8]).

5 보안모델의 구현

여기서는 우선 앞에서 나열한 보안정책들을 구현하는 도구로 객체지향 데이타모델의 시스템정의 메소드(system-defined method)를 이용하는 방안을 다룬다. 앞에서 지적했듯이 객체의 인스턴스변수에 대한 접근수단은 객체의 능동적인 성질을 나타내는 메소드이기 때문에 MORION에서는 메소드의 역활을 확장해서 보안정책의 구현도구로 이용했다. 이는 앞의 버전개념의 확장과 더불어, 기존 객체지향 개념을 이용해서 다단계보안을 지원한다는 면에서 자연스러운 접근방법이다. MORION에서 고려한 시스템정의 메소드로는 클래스객체의 객체생성 메소드 create, 객체의 삭제를 위한 delete, 모든 객체에 관련된 read, write이다. 여기서는 예로서 버전화가능 객체의 write 메소드를 예로 보인다. 단, 여기서 제시한 메소드의 알고리즘은 다단계보안을 지원하기 위해 수행해야할 개념적인 알고리즘이다. 그리고, 다중인스턴스화와 관련한 버전들사이의 기억장소 공유방안에 대해 논의한다.

5.1 버전화가능 객체의 write 알고리즘

보안정책 7를 만족하기 위해서는, 버전화가능한 객체의 쓰기를 위한 시스템정의 메소드는 해당 객체와 poly-from의 관계를 가지는 새로운 버전을 유도하고, poly-from의 관계를 생성객체의 자료구조에 표시해야 한다. 그림 8는 이런 일들을 수행하는 버전화가능 객체의 write 알고리즘을 보여주고 있다. 먼저 알고리즘에서 A.L은 앞에서 설명한 시스템 변수이고, Self는 메소드를 수행

```
write(A,a){
   if (A.L > L(o)) {
      oid = derive-version(Self, P);
      write(oid, (A,a));}
   else if (A.L = L(o))
      Self.A = a;
   else
      reject;
}
```

그림 8: 버전화가능 객체의 write 알고리즘

하는 객체 자신을 가리킨다. 'reject'는 아무런 일도 수행하지 않고 메소드의 수행을 끝내는 것을 나타낸다. 그리고, 메시지 호출은 다음과 같이 함수호출 형식과 비슷하다.

선택자(수신자,(메시지 인자))

$A.L > L(o)$인 경우, 메시지 derive-version(Self, (P))는 수신자를 Self, 즉 자신으로 하고, 두 버전간의 관계 poly-from을 나타내는 P를 인자로 갖는다. 이때 새로이 생성되는 버전의 보안등급은 A.L이 된다. 메시지 호출의 결과는 생성된 객체의 객체 식별자이다. 새로운 버전의 생성이 끝난 후에 이 버전의 인스턴스 변수의 값을 바꾸기 위한 메시지를 호출한다.

5.2 Poly-from 관계를 고려한 버전들사이의 기억장소 공유방안

poly-from의 관계에 있는 두개의 버전은 derived-from의 관계에 있는 두 버전보다 서로 공유하는 정보의 양이 대체로 많다. 예를 들어, derived-from의 관계에 있는 두개의 디자인 객체 버전의 새로이 유도한 버전에 대해서는 앞 버전과는 다른 여러가지의 시도를 한다. 반면에, 다중인스턴스화를 위해 생성하는 버전의 경우 앞의 버전과는 대개 인스턴스 변수 한 두개의 값이 틀린 경우가 많다.

하나의 버전의 크기가 작다면 각각의 버전에 대해 다른 기억장소를 할당하면 되지만, 큰 버전에 대해 poly-from의 관계에 있는 다른 버전을 유도할 때, 새로이 기억장소를 할당하는 것은 기억장소의 낭비가 된다. 따라서, 효율적으로 poly-from의 관계에 있는 여러 버전들사이의 기억장소 공유에 관한 방안이 필요하다.

EXODUS 저장시스템에서는 큰 버전의 경우 B+ 트리와 유사한 형식을 이용해 새로운 버전을

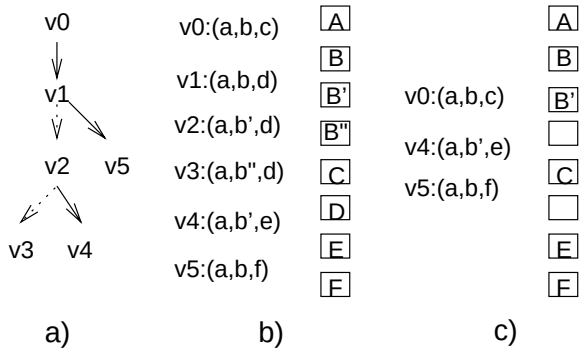

그림 9: 버전들사이의 디스크 공유와 버전 삭제

유도할 경우 앞 버전과 다른 부분에 대해서만 기억장소를 할당하고 나머지 부분은 두 버전이 공유한다[4]. poly-from 관계나 derived-from 관계의 버전 생성시에는 EXODUS의 알고리즘을 그대로 이용하면 되지만, 버전 삭제의 경우에는 보안정책 8에 의한 연쇄삭제를 고려해야한다. 이를 위해 두가지의 방법이 있는데, 첫번째 방법은 상위 레벨[6]에서 삭제되는 버전으로부터 직접, 간접으로 poly-from 의 관계에 있는 버전들을 따라가면서 기존 EXODUS의 버전삭제 알고리즘을 이용해 지우는 방법이고, 두번째 방법은 EXODUS의 버전삭제 알고리즘을 확장해서 하위레벨[7]에서 삭제되는 버전과 직접, 간접으로 poly-from의 관계에 있는 모든 버전들을 기억장소에서 지우는 방법이다. (EXODUS의 버전 생성및 삭제 알고리즘은 [4]에 자세히 나와있으므로 여기서는 설명하지 않는다.)

이 두가지의 방법을 설명하기 위해 그림 9를 예로 든다. a)의 버전계층구조에서 만일 v1을 삭제하는 경우 연쇄삭제에 의해 v1과 poly-from 관계에 있는 v2, v3은 함께 삭제해야 하고, v4와 v5는 버전 계층구조에 그대로 남아있어야 한다. 그림에서 b)와 c)에서 사각형안의 대문자는 디스크의 페이지 내용을 나타내고 버전번호와 괄호안의 소문자는 각각의 버전이 차지하고 있는 디스크 페이지를 가리킨다. b)는 v1 삭제이전의 디스크의 상태를 나타내고, c)는 v1 삭제이후의 상태를 나타낸다.

첫번째 방법은 버전관리자(version manager)에서 삭제되는 버전에 대해 EXODUS의 버전삭제 알고리즘을 호출해서 삭제한 뒤 해당 버전과 poly-from 관계에 있는 버전 각각에 대해 EXODUS

[6] 객체관리계층(object management layer)

[7] 저장시스템 계층(storage system layer)

의 버전삭제 알고리즘을 호출한다. 이전단계에 지워진 버전들과 poly-from 관계에 있는 버전이 더이상 없을때까지 반복한다. 물론 생성객체의 버전유도 계층구조에서 각각의 버전들이 삭제되었음을 표시한다. v1을 삭제하기 위해 버전삭제 알고리즘을 호출하면, 디스크 페이지 A, B, D 모두 다른 버전과 공유하고 있기때문에 실제적인 디스크 삭제는 일어나지 않는다. 버전 v2를 지우기 위해 버전삭제 알고리즘을 호출해도 A, B', D를 다른 버전과 공유하고 있으므로 디스크 삭제는 일어나지 않는다. 버전 v3를 삭제의 경우에는 디스크 페이지 B"와 D의 내용이 지워져서 결국 그림 9의 c)의 상태가 된다.

두번째 방법은 버전관리자에서 삭제할 처음 버전에 대해 삭제 알고리즘을 호출하면 해당 버전과 poly-from의 관계에 있는 버전들이 차지하고 있는 디스크의 기억공간도 모두 지우도록 EXODUS의 버전삭제 알고리즘을 확장하는 것이다. 간단히 확장된 알고리즘을 그림 9를 예로들어 설명하면 다음과 같다. v1, v1의 바로 이전 버전인 v0, v1에서 파생된 모든 버전들 v2, v3, v4, v5이 차지하는 모든 디스크 페이지를 고려해서 이전 버전인 vo, v1과 직접, 간접적으로 derived-from 관계의 버전 v4, v5이 차지하는 디스크 페이지 A, B, B', C, E, F를 표시하고, 표시가 안된 디스크 페이지 B", D를 삭제한다.

첫번째 방법은 단순하지만 비효율적이고, 두번째 방법은 버전삭제 알고리즘이 복잡해지지만 삭제할 버전들의 디스크의 내용을 한꺼번에 지우기때문에 효율적이다.

6 관련연구와의 비교

다단계보안 객체지향 데이타모델에 관한 두가지의 기존 연구로는 첫째, 다단계 객체(즉, 하나의 객체내의 인스턴스 변수들의 보안등급이 다른 경우)를 지원하는 모델(대표적인 예로 SORION 모델: [16, 17])이 있고, 둘째, 단일등급의 객체(즉, 인스턴스 변수들의 보안등급이 객체와 동일한 경우)를 지원하는 모델([6])이 있다.

SORION 모델에서는 객체지향 데이타모델 ORION의 기본 구성요소, 즉, 객체, 클래스, 클래스 계층구조 뿐만 아니라 확장된 구성요소인 버전, 복합객체(composite object)까지 고려한 광범위한 보안정책을 제시했다. 하지만, 다단계 객체와 관련한 여러가지의 문제점들(예를 들어, 다중 인스턴스화)에 대한 해결책이 없다. 그리고, 객체지향 데이타모델의 모든 구성요소들을 단순히 Bell LaPadula 모델의 데이타객체로 취급하므로써 객체지향 데이타모델과 다단계보안을 자연스

럽게 연결시키지 못했다.

[6]에서 제시한 모델은 SORION 모델보다 객체지향 데이타모델을 잘 이해한 경우인데, 객체지향 데이타모델에서의 객체가 인스턴스 변수와 메소드를 포함하고 있기때문에 Bell LaPadula 모델의 주체(subject)와 데이타객체(object)의 역할을 동시에 수행한다고 보고 있다. 그리고, 객체지향 데이타모델에서의 발생하는 다양한 일들, 즉 객체 생성, 상속, 버전 유도등을 메시지에 의한 정보흐름으로 일관되게 취급하고 있다. 이 모델에서 보안정책의 구현도구로서 메시지필터링(message filtering)이라는 개념적인 도구를 사용하고 있는데, 이 모델이 근거한 [13]에서 지적한 것처럼 DBMS와 같은 동적인 환경에서는 유지경비가 심하다.

MORION에서는 [6]에서 도입한 메시지 필터링 개념을 객체지향 데이타모델의 구성요소인 메소드를 통해서 구현함으로서 위에서 지적한 단점을 극복했다. SORION 모델과 비교해서, MORION은 메소드를 이용한 보안정책 구현이외에 다중인스턴스화와 관련한 의미 관계 poly-from을 정의하고, 이 관계를 객체지향 데이타모델의 버전개념을 확장해서 자연스럽게 지원하는 장점을 들수 있다.

7 결론

이상에서 알수 있듯이, MORION은 버전과 메소드를 확장함으로써 지금까지의 다단계보안 객체지향 데이타모델중에서 가장 자연스럽게 다단계보안을 객체지향 데이타모델로 흡수한 모델이다.

이 논문에서는 우선 기존 객체지향 개념들과 관련한 보안정책들을 제시하고, 다단계보안을 위해서 불가피한 다중인스턴스화 문제해결을 위해 새로운 의미관계 poly-from을 정의하고 이를 지원하기 위해 객체지향 데이타모델의 버전과 다중인스턴스화의 유사성을 근거로 해서 poly-from 관계를 버전개념의 확장을 통해서 모델링할수 있도록 했다. 보안정책 구현을 위해서 시스템정의 메소드들의 역활을 확장한 알고리즘을 제시하고, poly-from의 관계에 있는 버전들사이의 기억장소 공유방안을 다루었다.

MORION 모델의 문제점으로 지적할수 있는 것은 버전화불가능 객체에 대한 다중인스턴스화를 지원하지 못한다는 점인데, 이는 모든 클래스를 버전화가능 클래스로 선언하거나, poly-from 관계를 위한 버전을 생성할 필요가 있는 경우 버전화가능 클래스로 시스템에서 자동적으로 스키

마 변경을 수행하면 된다.(ORION은 버전화불가능 클래스에서 버전화가능 클래스로 스키마 변경을 지원한다[9].)

이 논문에서 다루지 않은 객체지향 개념중의 하나가 복합객체인데, 복합객체에 다단계보안을 적용했을때의 합리적인 보안정책에 대한 연구가 필요하다[17]. 복합객체의 버전문제는 보안을 고려하지 않은 경우에도 굉장히 복잡한 문제인데[9], 다단계보안을 고려한 경우는 많은 문제가 예상된다. 다단계보안을 보장하기 위해서는 같은 클래스에 속하지만, 다른 보안등급을 갖는 객체들은 서로 다른 장소에 저장해야 한다. 그러나, 효율적인 객체의 검색을 위해서는 같은 클래스에 속하는 객체는 되도록 같은 장소에 저장(clustering)시키는 것이 좋다. 이 상충하는 요구사항을 만족시키는 객체의 저장방법에 대한 연구가 앞으로 필요하다.

참고 문헌

[1] J. Banerjee et al. "Data Model Issues for Object-Oriented Applications". *ACM Trans. on Office Infomation Systems*, 5(1), Jan. 1987.

[2] D. Batory and W. Kim. "Modelling Concepts for VLSI CAD Objects". *ACM Trans. Database Syst.*, 10(3), Sept. 1985.

[3] D. Bell and L. LaPadula. "Secure Computer Systems: Unified Exposition and Multics Interpretation". *Technical Report MTIS AD-A023588*, July 1975.

[4] M. J. Carey, D. J. DeWitt, et al. "Objects and File Management in the EXODUS Extensible Database System". *Proc. of the Conf. on VLDB*, Aug. 1986.

[5] R. Fagin. "On an Authorization Mechanism". *ACM Trans. Database Syst.*, 3(3), Sept. 1978.

[6] S. Jajodia and B. Kogan. "Integrating an Object-Oriented Data Model with Multilevel Security". *Proceedings of IEEE Symposium on Research in Security and Privacy*, May 1990.

[7] S. Jajodia and R. Sandhu. "Polyinstantiation Integrity in Multilevel Relations". *Proceedings of IEEE Symposium on Research in Security and Privacy*, May 1990.

[8] S. Jajodia and R. Sandhu. "Update Semantics for Multilevel Relations". *IEEE*, 1990.

[9] W. Kim. *Introduction to Object-Oriented Databases.* MIT press, 1990.

[10] T. F. Lunt. "Polyinstantiation: an Inevitable Part of a Multilevel World". *Proceedings of the Fourth Workshp on the Foundations of Computer Security*, June 1991.

[11] T. F. Lunt, D. E. Denning, et al. "The SeaView Security Model". *IEEE Trans. on Software Engineering*, 16(6), June 1990.

[12] J. K. Millen and T. F. Lunt. "Security for Object-Oriented Database Systems". *Proceedings of IEEE Symposium on Research in Security and Privacy*, May 1992.

[13] N. H. Minsky and D. Rozenshtein. "A Law-Based Approach to Object-Oriented Programming". *ACM OOPSLA Conference Proceedings*, Oct. 1987.

[14] F. Rabitti et al. "A Model of Authorization for Next-Generation Database Systems". *ACM Trans. Database Syst.*, 16(1), Mar. 1991.

[15] P. D. Stachour and B. Thuraisingham. "Design of LDV: A Multilevel Secure Relational Database Management System". *IEEE Trans. on Knowledge and Database Eng.*, 2(2), June 1990.

[16] M. Thuraisingham. "Mandatory Security in Object-Oriented Database Systems". *ACM OOPSLA Conference Proceedings*, Oct. 1989.

[17] M. Thuraisingham. "Multilevel secure object-oriented data model – issues on noncomposite objects, composite objects, and versioning". *Journal of Object-Oriented Programmings*, Nov. 1991.