通信情報保護學會論文誌
第 7 卷 第 3 號. 1997. 9

Z 언어를 기반으로 CORBA 보안의 정형화된 접근 제어 모델 개발

김 영 균*, 김 경 범*, 인 소 란*

Development of a Formal Access Control Model in CORBA Security using the Z Language

Young-Kyun Kim*, Kyeong-Beom Kim*, So-Ran Ine*

요 약

분산 객체 시스템에서 보안 기술과 객체지향 기술의 통합은 중요한 요소이기 때문에 OMG(Object Management Group)에서는 객체 시스템에서 보안성을 다루기 위한 표준 보안 서비스 규격으로 CORBA(Common Object Request Broker Architecture) 보안 참조 모델(security reference model)을 제시하였다. CORBA 보안 참조 모델에서 접근 제어 기능은 특정 구현 메카니즘에 종속되지 않는 독립적인 개념으로 정의되고 있으며, 또한 보안 특성의 의미가 완전하게 기술되지 않는다. 이는 구현자와 사용자가 상호 일치되지 않은 개념을 갖을 가능성을 내포한다. 따라서 본 논문에서는 CORBA 보안 참조 모델에 기술된 접근 제어 기능이 갖는 의미가 정확히 기술될 수 있도록 집합론에 기초하는 정형화 언어인 Z 언어의 스키마 구조를 이용하여 CORBA 접근 제어 모델을 정형화한다.

Abstract

OMG(Object Management Group) published a security service specification, called CORBA (Common Object Request Broker Architecture) security reference model because the integration of security and object-oriented techniques was critical for successful deployment of distributed object systems. The CORBA security reference model treats access control as an implementation independent semantic concept but has incomplete semantics of the access control function. Because of such imcompleteness it is difficult for the system administrator and the CORBA security implementor to have the same understanding for the meaning of access control in the CORBA security. We propose a formal model for access control of the CORBA security using the formal description language, which is called Z language based on typed set theory. The proposed model provides concrete semantics of the access control function to both the system administrator and the implementor.

* 한국전자통신연구원 컴퓨터연구단 소프트웨어 공학연구실

1. 서 론

OMG(Object Management Group)는 객체지향 기술을 근간으로 이기종의 시스템들과 다양한 시스템 하부 구조들에 상호 협동할 수 있는 응용들을 구축할 수 있도록 지원하기 위한 표준화 컨소시움으로서, 분산 객체 응용을 개발하기 위한 객체 모델(object model)과 참조 구조(reference architecture)로 구성된 객체 관리 구조(Object Management Architecture: OMA)를 제시하고 있다.[10, 11] 이 구조에는 객체 요구 중개자(Object Request Broker : ORB)와 객체의 위치, 활성화, 그리고 통신에 있어서 투명성을 제공하는 메카니즘 등과 같이 다양한 요소들이 포함되어 있다. 공통 객체 요구 중개자 구조(Common Object Request Broker Architecture, 이후 CORBA라 칭함.)는 ORB에 의해서 반드시 제공되어야 하는 인터페이스와 객체 서비스들을 기술한 것으로서 분산된 응용들이 공유하는 객체 서비스는 분산 객체 환경을 위한 기본적인 기능을 지원한다. 이름(naming) 서비스, 사건, 영속성, 트랜잭션, 그리고 동시성 제어 등이 대표적인 객체 서비스의 형태들이다.

OMA를 기반으로 하는 새로운 분산 객체 응용 시스템은 전통적인 정보 시스템과 비교해서 보안 위협의 공격 포인트가 다양하기 때문에 보안성 측면에서는 취약하다.[12] 따라서 개방형 분산 환경을 갖는 CORBA 시스템에서는 다양한 보안 위협으로 부터 시스템을 보호하기 위한 보안 서비스가 필요하며, 이러한 요구를 충족시키기 위해서 OMG에서는 CORBA를 기반으로 하는 응용들에 대해서 각기 적정한 수준의 보안성을 보장하는데 필요한 핵심적인 보안 기능들과 인터페이스를 정의하고 있다.[12, 13]

분산 객체 응용들의 보안성을 위해 CORBA에서는 객체 서비스의 한 형태로 보안 객체 서비스를 정의한다. 이는 보안 시스템이 어느 곳에서 어떤 방법으로 보안 정책을 시행하는지를 정의한 보안 참조 모델과 이를 구현한 구조, 그리고 응용 개발자와 시스템 관리자, 객체 시스템 구현자가 이용할 수 있는 보안 기능과 인터페이스를 기술하고 있다.

CORBA는 이질적인 시스템들 사이의 분산 환경을 제공하기 때문에 신뢰성 있는 분산 시스템 환경을 제공하기 위해서는 CORBA 보안 서비스가 다음과 같은 요구사항들을 만족시켜야 한다. 먼저, 어떤 플랫폼(platform) 상에서 이식 가능하고, 표준화된 규격을 준수해야 하며, 또한 소규모 시스템 뿐만 아니라 대규모의 개방 환경의 분산 시스템을 지원할 수 있어야 한다. 그리고 사용자 또는 객체 응용에게 가능한 투명성을 제공해야 하고, 시스템에 의해 설정된 보안 정책을 사용자 또는 객체 서비스가 피해갈 수 없음을 보장해야 한다. 그리고 다른 영역에 있는 객체들과 상호 운영성을 투명하게 지원해야 할 뿐만 아니라 보안 모델이 특정한 보안 알고리즘에 종속되지 않는 보안 기법의 독립성이 제공되어야 한다. 위와 같은 기초적인 요구사항 위에서 CORBA 보안 서비스는 OMA 구조를 만족시키며, 다른 객체 서비스들 사이에서 보안의 특성인 비밀성, 무결성, 책임성, 유용성 등을 만족시키는 특성을 갖는다. CORBA 보안 서비스가 지원하는 보안 기능으로는 인증 서비스, 특권 서비스, 키 분배, 감사 기능 그리고 접근 제어 서비스 등을 가지며, 또한 객체 응용들에 적용되는 보안 정책도 지원한다. 특히, CORBA의 보안 서비스들 중에서도 접근 제어 기능은 사용자와 제어 대상인 객체의 갯수를 줄이고, 또한 권한부여 상태를 나타내는 접근 행렬(access matrix)의 크기를 감소시켜서 보안 관리자의 정책 관리에 수반되는 복잡성을 줄이는 특징을 갖는다.

CORBA 보안 서비스에서 접근 제어 기능은 특정 시스템의 메카니즘에 종속적인 구현과는 분리된 독립적인 의미를 갖는 개념으로 정의

된다. 이는 CORBA 보안의 접근 제어 기능을 이용하는 보안 관리자와 이를 구현하는 개발자가 보안성 기능을 이해하는데 있어서 상호 일치되지 않은 혼돈된 개념을 갖을 가능성이 있다. 따라서 CORBA 보안의 접근 제어 기능을 표현하는 언어가 정형화된 의미(formal semantics)를 갖을 필요성이 있으며, 이는 보안 관리자와 개발자가 동일한 개념의 접근 제어 기능을 공유하여 상호 정확한 의미를 파악할 수 있게 한다. 특히, 정형화된 의미를 표현하는 접근 제어 모델의 필요성은 시스템 개발자가 CORBA 보안의 접근 제어 기능을 구현하는 시점에서 중요한 요소이다. 이와 같은 정형적 방법론을 적용하므로써 얻을 수 있는 잇점은 개발하는 모델의 정확성 입증이 가능할 뿐만 아니라 개발자와 사용자가 쉽게 상호 일치된 의미를 공유할 수 있다는 것이다.

또한, CORBA 보안 참조 모델은 이를 기초로 하여 구현한 보안 객체 서비스가 어느 수준의 보안 기능을 지원하는가를 평가할 수 있도록 보안 기능의 부합 수준(security function conformance level)을 제시하고 있다. 그러나 국제적인 보안 평가 기관인 TCSEC(Trust Computer System Evaluation Criteria), ITSEC (Information Technology Security Evaluation Criteria) 등이 마련한 평가 기준[16, 17]에서 높은 수준의 보안성 평가 등급을 만족시키기 위해서는 평가할 보안 기능이 정형적으로 기술되어야 함을 요구한다.

시스템의 정형화 개발 방법에서는 논리식, 함수, 집합론 등과 같은 수학적 표기법을 기초로 하는 정형적 개발 방법론이 있으며, 그 종류로는 Z 정형화 언어, B Abstract Machine 표기법, PVS, VDM 등이 있다.[3, 7]

1980년대 초에 Oxford 대학에 의해서 정형화된 규격 표기법으로 Z 언어가 개발되었으며, 현재는 Z 정형화 표기법이 소프트웨어와 하드웨어 개발 현장에서 널리 이용되고 있다. Z 언어[3]는 집합론(set theory)과 일차 술어(first order predicate)에 기초하여 컴퓨터 연산 시스템을 기술하기 위한 범용 정형화 규격 표현법이다. Z 언어의 기본 구조는 간결하지만 정교한 내용을 표현할 수 있는 스키마(schema)이며, 스키마는 각기 구분되는 개체(entity)들과 이 개체들이 정의된 공리(axiom)에 의해 맺어지는 관계의 모임으로 구성된다. 스키마는 시스템의 정적인 측면과 동적인 측면을 모두 나타낼 수 있는 특징을 갖으며, 상태(state) 스키마와 연산(operation) 스키마로 분류된다. 상태 스키마는 시스템의 상태를 정의하고, 이 상태에 대한 여러 형태의 제약조건을 표현할 수 있다. 반면에 시스템의 동적인 특성은 연산 스키마로 구조화되며, 이는 여러 형태의 연산들을 표현할 수 있다. 그리고 Z 언어는 서로 다른 스키마들을 다양한 방법으로 결합시키는 표기법을 제공하기 때문에 규모가 큰 응용을 단계적으로 쉽게 구성할 수 있다. 이러한 Z 언어는 특정 시스템의 모델을 기술하는데 있어서 그릇된 개념들을 쉽게 식별할 수 있는 잇점을 제공한다.

본 논문에서는 CORBA 보안 참조 모델에 제시된 접근 제어 기능을 정형화시키기 위해서 집합론에 기초한 이산 수학식과 Z 언어의 기본 구조인 상태 스키마와 연산 스키마를 각기 이용하여 정형화된 접근 제어 모델을 개발한다. 정형화된 CORBA 보안의 접근 제어 모델은 전통적인 접근 제어 모델의 구성 요소들을 바탕으로 확장된 구조를 갖으며, 또한 접근 제어를 위한 권한부여 관리 인터페이스를 포함한다.

본 논문의 구성은 다음과 같다. 먼저, 2장에서 CORBA 보안 문서에 기술된 접근 제어 기능을 간략히 설명하고, 기존의 접근 제어 모델과 구분되는 특징들을 비교 분석한다. 3장에는 본 논문에서 개발한 정형화된 접근 제어 모델의 구조를 제시하고, 그리고 4장에서는 접근 제어 관리를 위한 정형화된 인터페이스를 정

의한다. 마지막으로 5장에서 결론과 추후 연구 방향을 언급한다.

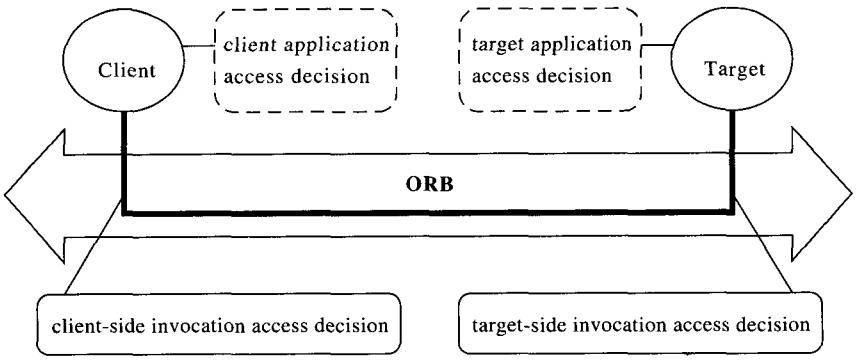
2. 관련 연구

본 장에서는 보안 모델의 개발 분야에서 Z 언어를 적용한 사례 연구들을 간략히 살펴 보고, 그리고 본 논문에서 목표로 하는 정형화 접근 제어 모델의 대상인 CORBA 보안의 접근 제어 개념을 설명한다.

먼저, 시스템 보안 분야에서 정형화된 보안 모델의 개발에 Z 언어를 적용한 연구 분야들을 살펴 보면, 주로 화일 시스템의 접근 제어 모델, 데이타베이스 시스템의 권한부여(authorization) 모델, 그리고 다단계 보안(multilevel security) 모델 등에서 수행되고 있다. 특히, 이러한 대부분의 연구들은 주로 다단계 보안 정책의 정형적 정의와 보안 모델의 개발에 초점을 두고 있다.

화일 시스템의 보호 분야에서 시스템의 보안 특히, 접근 제어에 관련된 화일 연산들을 포함하는 UNIX 화일 시스템을 Z 언어로 제시한 연구가 [9]에서 수행되었으며, 반면에 데이타베이스 또는 운영 체제에 적용되는 다단계 보안 정책을 수용하는 다단계 보안 모델을 Z 언어로 제시한 연구가 [2, 4, 6]에서 수행되었다. 또한, 자율적인(discretionary) 접근 제어 정책과 다단계 보안 정책을 모두 포함하는 보안 모델에 대한 Z 정의가 [1]에 제시되어 있으나, 이 연구도 주로 다단계 보안 모델의 구성 요소에 대한 부분을 다루고 있다. 한편, 데이타베이스의 개념 설계에서 이용되는 데이타베이스 모형화 모델인 개체 관련성(entity relationship) 모델과 데이타 흐름 다이아그램(data flow diagram)의 관계를 Z 언어로 정형화시킨 연구가 [8]에서 수행되었다.

분산 객체 시스템의 보호를 위한 CORBA 보안 참조 모델에서는 접근 제어 정책이 〈그림 1〉에서와 같이 두 가지 형태의 수준에서 정의될 수 있도록 허용한다.[12, 13] 즉, CORBA 보안의 접근 제어 기능은 객체 호출(object invocation) 시점과 응용 객체 내에 각기 독립적으로 병행 수행될 수 있다. 객체 호출시 수행되는 접근 결정은 객체에 정의된 정책과 구현된 접근 결정이 무엇인지에 관계없이 표준 접근 결정 인터페이스를 통해서 이루어진다. 비록 모든 응용들이 자신의 접근 제어 정책에 따른 접근 제어를 시행할 수 있을지라도, 현재 응용들에서 사용하는 표준화된 접근 제어 정책이 존재하지 않기 때문에 본 논문에서는 객체 호출시 수행되는 접근 제어 정책만을 고려한다.



〈그림 1〉 CORBA 보안의 접근 제어를 위한 프레임워크

CORBA 보안에서 접근 제어를 위한 보안 정책은 시스템 자원과 수행 요청 개시자(principal)의 보안 어트리뷰트에 기초한다. 이는 접근 제어를 위한 정보의 크기를 줄일 뿐만 아니라 객체보다는 개개의 연산들에 대해서 보다 정교한 형태의 접근 제어를 시행할 수 있는 잇점이 있다. 개시자는 사용자 또는 특정 사용자에 결합된 행위에 대해서 책임을 갖는 프로세스이며, 정의된 정책에 따라서 개시자는 접근 제어에 사용되는 특정한 권한(privilege) 어트리뷰트들을 갖는다. 권한 어트리뷰트의 타입으로 사용자 식별자, 역할(role), 그룹(group), 보안 등급(clearance), 그리고 능력 리스트가 있으며, 어느 시점이든 개시자는 객체에 접근하는데 필요한 자신의 권리를 설정하기 위해서 자신에게 허용된 권한 어트리뷰트들 중에서 특정한 권한을 선택할 수 있다. 반면에, 시스템 자원의 보안 어트리뷰트는 응용 객체인 타겟(target)에 결합된 보안 제어(security control) 어트리뷰트로 정의된다. 이러한 보안 제어 어트리뷰트를 기초로 CORBA 보안에서는 객체 호출시 적용되는 특별한 형태의 접근 제어를 정의한다.

먼저, 객체 호출시 적용되는 접근 제어는 객체 인스턴스 단위보다는 도메인 단위로 보안 정책 관리 문제를 처리할 수 있도록 보안 정책 도메인 개념을 정의한다. 보안 정책 도메인은 응용 객체의 집합이며, 하나의 도메인에 소속된 모든 객체들이 보안성을 인식하고 있는 객체인지 혹은 보안을 인식하지 못하는 객체인지에 상관없이 공통된 접근 제어 정책이 적용된다.

실제로 접근 제어 결정을 위해 필요한 요소들로는 도메인 접근 정책(domain access policy)과 요구 권리(required rights) 정책, 그리고 이러한 정보를 이용하여 접근 결정을 수행하는 접근 결정(access decision) 함수가 있다. 도메인 접근 정책은 사용자의 집합에 하나의 도메인에 포함되는 모든 객체들에 대해서 연산을 수행할 수 있는 한정된 일련의 권리 집합을 부여한 것이다. 이는 〈그림 2(a)〉에서와 같이 특정한 도메인에 존재하는 응용 객체들에 대해서 개시자가 갖는 권리를 정의한다. 반면에 요구 권리 정책은 하나의 보안 객체 인터페이스에 있는 각각의 연산을 호출할 때 요구되는 권리들의 집합을 정의한다(〈그림 2(b)〉). 또한 요구 권리 정책은 한 사용자가 메소드를 수행하기 위해서 그 메소드가 요구하는 권리 항목에 기술된 모든 권리가 필요한지 또는 그 권리 항목에 기술된 권리들 중에서 최소한 하나의 권리만 일치해도 수행할 수 있는지를 정의하는 권리 조합자(right combinator)라는 특별한 형태의 메카니즘을 지원한다. 이러한 요구 권리는 객체 인스턴스 보다는 클래스 기반으로 할당되기 때문에 특정 클래스의 모든 인스턴스는 항상 동일한 요구 권리를 갖는다.

Privilege	Delegation	Granted right
alice	initiator	**corba:gs-**
alice	delegate	**corba:g--**
manager	initiator	**corba:gsm**

Required rights	Combinator	Op.	Interface
corba:g--	all	m1	c1
corba:gs-	any	m2	c2
corba:gsm	all	m3	c2

(a) 도메인 보안 정책 리스트 b) 요구 권리 정책 리스트

〈그림 2〉 도메인 접근 정책과 요구 권리 정책의 상관 관계

CORBA 보안 참조 모델은 어떠한 방법으로 접근 결정을 수행해야 하는지에 대한 명시적인 규칙을 기술하지 않지만, CORBA 보안 규격에 기술된 예로 부터 접근 결정 규칙을 유도할 수 있다. 객체 호출시 접근 결정 함수는 호출된 객체의 메소드 수행 요청에 결합된 사용자의 신임장이 이 객체를 포함하는 도메인에 대한 권리를 부여한 권한 어트리뷰트를 포함하고, 또한 부여된 권리와 동일한 권리가 호출한 메소드의 요구 권리 정책에 포함된 경우에 접근을 허용한다.

지금까지 설명된 CORBA 보안 참조 모델의 접근 제어 기능과 기존의 접근 제어에서 널리 이용되는 접근 행렬 모델[14]에서 제시한 접근 제어 리스트의 구조를 비교해 볼 때 몇가지 구분되는 특징들이 존재한다. 첫째, CORBA의 도메인 접근 정책은 접근 제어의 대상인 개시자를 그들의 권한 어트리뷰트를 이용하여 사용자 항목으로 집단화한다. 이는 접근 제어 정책의 관리를 단순화시킬 수 있다. 둘째, 기존의 접근 제어 리스트와는 달리 CORBA의 도메인 접근 정책은 하나의 권한 어트리뷰트가 위임(delegate) 상태에서 사용될 때와 개시(initiator) 상태에서 사용될 때 각기 서로 다른 권리를 부여할 수 있다. 셋째, 다양한 형태의 권한부여를 가능하게 하도록 CORBA의 접근 제어에서는 모든 권리를 권리 한정자(rights family)라는 특별한 접근 제어 타입으로 한정한다. 이는 기존의 접근 제어 리스트에는 존재하지 않는 개념으로서 보안 관리자가 융통적으로 권한부여 상태를 관리하는데 도움이 된다. 반면에, CORBA 보안의 접근 제어 모델은 SESAME 시스템[15]과 데이타베이스의 보안에서 널리 이용되는 확장된 접근 제어 모델[5]에서 접근 제어 정보의 융통성있는 관리를 위해 지원하는 함축적 권한부여(implied authorization)와 부정적 권한부여(negative authorization) 기능에 대해서는 고려하지 않는다.

3. CORBA 접근 제어 모델의 정형화

응용 객체 호출시에 ORB에 의해 자동적으로 수행되는 접근 제어를 위한 CORBA의 접근 제어 모델은 특정 연산이 수행될 수 있는지를 결정하는 접근 결정 함수를 기초로 한다. 접근 결정 함수는 사용자의 접근 허용 여부를 판단하기 위해 사용자의 권한 어트리뷰트, 응용의 접근 제어 리스트를 나타내는 응용 객체의 보안 제어 어트리뷰트, 그리고 수행하려는 연산 등과 같은 행위에 관련된 보안 문맥(security execution context) 어트리뷰트가 필요하다. 따라서 이러한 어트리뷰트들은 한데 묶어져서 하나의 권한부여 상태(authorization state)로 정의될 수 있다. 결과적으로 CORBA의 접근 제어 모델은 다음과 같이 두가지 요소로 정의된다.

AccessDecisionFunction

AuthorizationState

privilege : P PrivilegeAttribute
security_control_attribute : DomainAccessPolicy
security_context_attribute : RequireRights

AccessControlModel

AuthorizationState
AccessDecisionFunction

먼저, 접근 제어 모델의 구성요소들을 각각 정형화하기 이전에 정형화 단계에서 이용되는 기본적인 타입들을 아래와 같이 정리한다.

[USERS, ACCESS_ID, GROUP_ID, ROLE, CLASS, OPERTATION]

- ✓ USERS : 사용자의 집합
- ✓ ACCESS_ID : 사용자 식별자의 집합
- ✓ GROUP_ID : 그룹 식별자의 집합
- ✓ ROLE : 역할 이름(식별자)의 집합
- ✓ CLASS : 클래스 이름의 집합
- ✓ OPERTATION : 연산(메소드)의 집합

- ✓ BOOLEAN : =true | false
- ✓ DELEGATE_STATE : =initiator | delegator
- ✓ RIGHTS_FAMILY : =corba | other
- ✓ RIGHTS_VALUE : =get | set | manage
- ✓ RIGHTS_COMBINATOR : =SecAllRights | SecAnyRights

3.1 권한부여 상태

CORBA 접근 제어에서 특정 사용자의 접근을 제한하는데 필수적인 정보는 시스템 보안 관리자에 의해 결정된 권한부여 상태의 집합이다. 이 권한부여 상태는 Lampson에 의해 제시된 초기의 접근 제어 모델[14]에 기술된 접근 행렬의 요소인 주체, 객체 그리고 연산 뿐만 아니라 권리(rights) 개념을 도입한 변형된 구조를 갖는다. 즉, 권리를 기반으로 하는 권한부여 상태는 사용자가 갖는 권한 어트리뷰트와 이 권한으로 응용 객체에 포함되는 인터페이스의 각 메소드들을 수행하는데 요구되는 권리로 구성된다.

3.1.1 사용자 권한 어트리뷰트

CORBA 접근 제어 모델은 개개의 사용자 단위로 권한을 부여할 뿐만 아니라 사용자들을 집단화시켜 관리하기 위해서 2장에서 설명된 여러 형태의 권한 어트리뷰트들을 정의하고 있다. 그러나 객체 호출시 ORB에 의해 자동적으로 수행되는 접근 제어에서는 권한 어트리뷰트의 타입으로 사용자 식별자, 기본 그룹, 그룹 그리고 역할만을 사용한다.

먼저, 사용자 식별자는 개개의 사용자에게 구분되는 접근 권한을 부여하기 위한 권한 어트리뷰트이며, 그 의미는 다음과 같이 정형적으로 정의할 수 있다.

AccessId_Privilege

$$\text{Has ID} : \text{USERS} \rightarrow \text{ACCESS\_ID}$$

$$\forall u : \text{USERS};\ a_i, a_j : \text{ACCESS\_ID} \bullet$$
$$(\langle u, a_i\rangle \in \text{HasID} \land \langle u, a_j\rangle \in \text{HasID} \Rightarrow a_i = a_j)$$
$$\land\ \forall u_i, u_j : \text{USERS};\ a : \text{ACCESS\_ID} \bullet$$
$$(\langle a, u_i\rangle \in \text{HasID}^{-1} \land \langle a, u_j\rangle \in \text{HasID}^{-1} \Rightarrow u_i = u_j)$$
$$\land\ \text{dom HasID} \subseteq \text{USERS}$$

위에 정의된 AccessId_Privilege 스키마의 상태를 표현하는 시그니쳐(signature)는 사용자와 사용자 식별자 권한 집합이 존재하고, 또한 이들 사이에는 함수 관계가 존재함을 의미한다. 이와 같이 사용자와 사용자 식별자 권한 사이에 정의되는 함수 관계는 반드시 이 스키마의 프레디킷(predicate)에 기술된 제약조건을 만족해야 한다. 즉, AccessId_Privilege 스키마의 프레디킷에서 첫번째와 두번째 조건은 한 사용자가 동시에 둘 이상의 사용자 식별자 권한을 갖지 못하고, 또한 식별자 권한을 갖는 사용자는 사용자 집합에 반드시 존재해야함을 제약한다. 그리고 세번째 제약조건에 나타나는 'dom' 연산자는 특정 함수의 정의역에 해당하는 집합을 표현하며, 이는 사용자 식별자 권한을 갖는 사용자는 반드시 전체 사용자 집합의 부분집합이어야 함을 제한한다.

Group_Privilege

$$HasGroup : USERS \leftrightarrow Group\_ID$$

$$(\forall u : USERS ; \exists g_i, g_j : GROUP\_ID \bullet (g_i \neq g_j \Rightarrow \langle u, g_i\rangle \in HasGroup \wedge \langle u, g_j\rangle \in HasGroup) \wedge (\forall u_i, u_j : USERS ; \forall g : GROUP\_ID \bullet (\langle u_i, g\rangle \in HasGroup \wedge \langle u_j, g\rangle \in HasGroup) \Rightarrow u_i \neq u_j) \wedge dom\ HasGroup \subseteq USERS$$

Role_Privilege

$$HasRole : USERS \leftrightarrow Role$$

$$(\forall u : USERS ; \exists r_i, r_j : ROLE \bullet (r_i \neq r_j \Rightarrow \langle u, r_i\rangle \in HasRole \wedge \langle u, r_j\rangle \in HasRole) \wedge (\forall u_i, u_j : USERS ; \forall r : ROLE \bullet (\langle u_i, r\rangle \in HasRole \wedge \langle u_j, r\rangle \in HasRole) \Rightarrow u_i \neq u_j) \wedge dom\ HasRole \subseteq USERS$$

CORBA 접근 제어 모델에서 그룹 권한 어트리뷰트는 비록 기본 그룹과 기타의 그룹 어트리뷰트로 구분되어 있지만, 각각의 갖는 의미는 개개의 사용자를 그룹화하기 위한 것이며, 이는 역할 권한 어트리뷰트에서도 동일하다. 특정 사용자는 기본적으로 사용자 식별자 권한 어트리뷰트를 갖으며, 또한 하나 이상의 그룹(또는 역할) 권한 어트리뷰트를 가질 수 있다(PrivilegeAttribute 스키마 참조). 따라서, 한명의 사용자가 특정한 응용에 접근 할 때, 접근 제어는 사용자 식별자 권한과 그룹 권한에 대해서 모두 시행된다. 일반적으로 접근 제어 모델에서 그룹 개념을 차용하는 경우에는 그룹들 간의 계층구조(hierarchy)를 기술한다. 그러나 CORBA 보안 참조 모델에서는 이를 구현시 고려사항으로 넘기고, 접근 제어 모델에는 정의하지 않고 있다. 따라서 정형화 접근 제어 모델에서도 Group_Privilege 스키마와 Role_Privilege 스키마에 대한 정형화에서 계층구조를 고려치 않고, 단지 한 사용자가 하나 이상의 그룹 또는 역할에 소속될 수 있도록 정의한다.

결과적으로, CORBA 접근 제어에서 사용자가 응용 객체에 대해서 특정 연산을 실행할 때 갖는 권한은 식별자 권한 뿐만 아니라 사용자가 정의된 그룹과 역할에 대한 권한 어트

리뷰트를 동시에 갖을 수 있다. 따라서 이러한 의미를 포함시켜서 총괄적으로 권한 어트리뷰트를 정형화하면 다음의 PrivilegeAttribute 상태 스키마와 같이 정의된다.

PrivilegeAttribute

$pa : \mathbb{P}\ PrivilegeAttribute$

AccessId_Privilege

Group_Privilege

Role_Privilege

$pa \subset AccessId\_Privilege \cup Group\_Privilege \cup Role\_Privilege$

3.1.2 제어 어트리뷰트

CORBA 접근 제어 모델에서 접근 제어 리스트를 나타내는 보안 제어 어트리뷰트는 응용 객체인 타겟에 결합되며, 개개의 보안 도메인들마다 독립적으로 존재할 수 있다. 특히, 객체 호출시 적용되는 접근 제어에서는 권리에 기반한 보안 제어 어트리뷰트로 도메인 접근 정책을 사용한다. 도메인 접근 정책은 사용자의 권한 어트리뷰트에 특정한 권리를 결합시킨 형태이며, 이 정보를 근간으로 허가된 사용자를 식별한다.

도메인 접근 정책의 구성은 권한 어트리뷰트, 위임 플래그 그리고 권리로 이루어진다. 먼저, 권리는 권리 한정자와 권리 값의 쌍으로 기술되며, 예를 들어 권리 한정자가 'corba'이고 권리 값이 'get'인 경우에 권리는 'corba:g-'가 된다. 따라서 CORBA의 접근 제어에서 이용되는 권리를 다음과 같이 정형적으로 정의할 수 있다.

Rights

$RightsDetail : RIGHTS\_FAMILY \leftrightarrow RIGHTS\_VALUE$

$\exists v : RIGHTS\_VALUE \bullet (\forall f_i, f_j : RIGHTS\_FAMILY \bullet$

$\langle f_i, v\rangle \in RightsDetail \wedge \langle f_j, v\rangle \in RightsDetail) \Rightarrow f_i \neq f_j$

위에 정의된 Rights 스키마의 상태를 표현하는 RightsDetail 릴레이션을 집합 표현으로 바꾸어 재정의하면 $(\forall r : RightsDetail \bullet r \in (RIGHTS_FAMILY \times \mathbb{P}\ RIGHTS_VALUE)$이 되며, 이 스키마의 프레디킷은 특정 권리 값이 하나 이상의 권리 한정자에 중복하여 사용될 수 있음을 정의한다.

3.1.1절에서 정의된 사용자의 권한 어트리뷰트를 표현한 상태 스키마와 위에 정의한 권리 스키마를 이용하여 완전한 형태의 도메인 보안 정책을 정형화하면 다음과 같다.

DomainAccessPolicy

HasState : PrivilegeAttribute ↔ F DELEGATE_STATE

HasRight : PrivilegeAttribute ↔ Rights

∀p, q : PrivilegeAttribute • (∃s : HasState ; r : HasRight

• dom s = p ^ dom r = q ⇒ p = q

Δ DomainAccessPolicy

domain_policy : DomainAccessPolicy

domain_policy' : DomainAccessPolicy

---

domain_policy' ^ domain_policy

Ξ DomainAccessPolicy

Δ DomainAccessPolicy

θ DomainAccessPolicy' = θ DomainAccessPolicy

위에 정의된 스키마들 중에서 DomainAccessPolicy 스키마는 도메인 보안 정책의 내부 구조를 정의한 것이며, 이 스키마의 시그니쳐와 프레디킷은 하나의 권한 어트리뷰트가 존재하고, 그리고 이 권한 어트리뷰트가 하나의 위임 플래그를 갖으면 반드시 이 권한은 하나의 권리를 갖어야 함을 의미한다. 그리고 ΔDomainAccessPolicy 스키마는 기존의 도메인 접근 정책의 상태가 변경되는 스키마를 정의한 것이며, 새로운 상태의 도메인 보안 정책인 domain_policy'와 기존의 domain_policy 스키마 사이에는 상호 공통된 요소가 존재하지 않음을 의미한다. 즉, ∀p, p' : PrivilegeAttribute; d, d' : DELEGATION_STATE; r, r' : Rights • (<p, d> ∈ HasState ^ <p, r> ∈ HasRight ^ <p', d'> ∈ HasState' ^ <p', r'> ∈ HasRight' ⇒ HasState ∩ HasState' = ∅ ^ HasRight ∩ HasRight' = ∅)를 의미한다. 반면에 세번째 스키마인 ΞDomainAccessPolicy 스키마는 도메인 접근 정책을 권한 어트리뷰트와 권리를 추가 또는 삭제하여 상태를 변경할 때에도 상태가 불변인 상태의 스키마를 정의한 것이며, 이는 변경되지 않는 스키마의 변수들을 의미한다. 이 스키마의 프레디킷에서 'θ'연산자가 스키마에 정의된 변수의 갯수를 나타낸다. 두번째와 세번째 스키마는 4장에서 언급되는 접근 제어 정보의 관리 인터페이스 정형화 과정에서 이용된다.

3.1.3 보안 문맥 어트리뷰트

사용자가 응용 객체에 기술된 인터페이스의 특정한 메소드를 실행하는 정보가 보안 문맥 정보이며, 이는 접근 결정 함수에서 접근의 허용 여부를 판단하는데 필수적인 정보이다. 전통적인 접근 제어 리스트와 비교해 보면, 보안 문맥 어트리뷰트는 주체 즉, 사용자가 접근하는 객체와 객체에 대해 수행하는 연산에 대응한다. CORBA 접근 제어 모델에서 이러한 문맥 정보는 요구 권리(RequiredRights) 인터페이스로 기술되며, 이는 접근 정책 정보의 데이타베이스로 간주할 수 있다.

요구 권리의 구성은 사용자가 접근하는 응용 객체의 인터페이스, 인터페이스에 정의된 메소드, 권리 그리고 권리 조합자로 이루어진

다. 여기서의 권리는 타켓 응용인 인터페이스의 특정 메소드를 수행할 수 있도록 접근이 허용된 권한을 의미하며, 이 권리는 반드시 도메인 보안 정책에 포함되어야 한다. 특히, 권리 조합자는 CORBA 접근 제어 모델에서 제안한 특징적인 요소이며, 이는 접근 결정 함수에서 접근 허용 여부를 결정할 때 이용되는 정보이다. 따라서 위와 같은 의미를 갖는 요구 권리 정책을 정형적으로 정의하면 다음과 같다.

RequireRights

HasCombinator : Rights ↔ F RIGHTS_COMBINATOR
HasOperations : Rights ↔ OPERATIONS
HasClass : Rights ↔ CLASS
IsDefined : OPERATIONS → CLASS

HasClass ⊆ HasOperations ; IsDefined

요구 권리 정책의 상태는 위의 RequiredRights 상태 스키마의 시그니쳐에 정의된 것처럼 3가지의 릴레이션들과 하나의 함수로 표현된다. 즉, RequiredRights 스키마의 시그니쳐는 요구 권리 정책이 권리, 권리 조합자, 메소드 그리고 접근하는 응용 객체인 인터페이스로 구성됨을 의미한다. 또한, RequiredRights 스키마의 프레디킷에서 ';' 연산자는 Z 언어에서 제공하는 기본 연산자로서, 두 릴레이션의 복합 관계를 표현한다. 이 연산자가 표현하는 의미를 집합 표현으로 재정의하면 ∀r : Rights; c : CLASS • (<r, c> ∈ HasClass ⇒ ∃op : OPERATIONS • <r, op> ∈ HasOperations ^ <op, c> ∈ IsDefined)로 기술할 수 있다.

그리고 요구 권리 정책의 한 인스턴스에 하나의 권리가 존재하면 반드시 인터페이스와 메소드 그리고 권리 조합자가 동시에 존재해야 하는 불변의 특성을 정의하기 위해서 다음과 같은 InvRequiredRights 스키마를 정형적으로 정의한다. 또한 도메인 보안 정책의 스키마에서와 같이 요구 권리 정책의 정보를 변경시키는 연산들에 의해 그 상태가 변경되는 스키마와 변경되지 않는 스키마를 각기 정형화하면 다음과 같다.

InvRequiredRights

RequiredRights

---

dom HasCombinator = dom HasOperations = dom HasClass

Δ RequiredRights

InvRequiredRights
InvRequiredRights'

InvRequiredRights' ^ InvRequiredRights

Ξ RequiredRights

Δ RequiredRights

θ RequiredRights' = θ RequiredRights

3.2 접근 결정 함수

CORBA 접근 제어 모델에서 객체 호출시 접근 결정은 'AccessDecision' 인터페이스에 의해 수행되며, 이는 실행 시간에 접근 제어를 검사하는데 이용된다. CORBA의 접근 결정 함수는 이 결정의 근간이 되는 보안 정책 정보에 의존하며, 보안 정책 정보는 앞에서 설명된 도메인 보안 정책과 요구 권리 정책과 같은 권한부여 상태의 집합을 의미한다.

먼저, 접근 결정 함수에는 특정한 인터페이스의 메소드를 수행하도록 요청한 사용자의 권한 어트리뷰트가 객체 호출 시점에 전달된다고 가정하며, 이런 가정에서 CORBA의 접근 결정 함수는 두 가지 형태의 접근 결정을 수행한다. 즉, 요구 권리 정책의 권리 조합자가 'SecAllRights'인 경우와 'SecAnyRights'인 경우에 각각 서로 다른 접근 결정 메카니즘을 바탕으로 수행된다.

특정한 인터페이스에 정의된 메소드를 수행하는 권리의 권리 조합자가 'SecAllRights'인 경우는 메소드 수행을 요청한 사용자가 갖는 권리 집합이 이 메소드를 수행하기 위해 요구되는 권리의 집합과 동일해야만 접근이 허용되도록 결정 함수가 동작한다. 반면에 권리 조합자가 'SecAnyRights'인 경우에 접근 결정 함수는 사용자의 권리가 메소드 수행에 요구되는 권리 집합의 부분 집합이 되면 요청한 접근을 허용한다. 이와 같은 의미를 포함하는 접근 결정 함수를 정형화시켜 정의하면 다음과 같다.

AccessDecisionFunction

$$
\begin{aligned}
&\Xi\ \text{DomainAccessPolicy}\\
&\Xi\ \text{RequiredRights}\\
&pa? : \text{PrivilegeAttribute}\\
&operation? : \text{OPERATIONS}\\
&name? : \text{CLASS}\\
&flag! : \text{BOOLEAN}\\
&\hline
&pa? \in \text{dom HasRight}\\
&(\exists gr : \text{Rights} \bullet gr = \text{ran HasRight } pa? \wedge (\exists rr : \text{Rights} \bullet\\
&\quad op? = \text{ran HasOperations } rr \wedge name? = \text{ran IsDefined } op?)\\
&\quad \wedge (\exists rc : \text{RIGHTS\_COMBINATOR} \bullet rc \in \text{ran HasCombinator } rr\\
&\quad \wedge rc = \text{SecAllRights}) \wedge gr \sqsubseteq rr = \#rr \Rightarrow flag! = \text{true})\\
&\quad \vee\\
&(\exists gr : \text{Rights} \bullet gr = \text{ran HasRight } pa? \wedge (\exists rr : \text{Rights} \bullet\\
&\quad op? = \text{ran HasOperations } rr \wedge name? = \text{ran IsDefined } op?)\\
&\quad \wedge (\exists rc : \text{RIGHTS\_COMBINATOR} \bullet rc \in \text{ran HasCombinator } rr\\
&\quad \wedge rc = \text{SecAnyRights}) \wedge (gr \cap r \subseteq rr) \wedge gr \sqsubseteq rr \geq 1 \Rightarrow flag! = \text{true})\\
&\quad \vee\\
&\quad flag! = \text{false}\\
&\text{DomainAccessPolicy}' = \text{DomainAccessPolicy}\\
&\text{RequiredRights}' = \text{RequiredRights}
\end{aligned}
$$

위의 연산 스키마에서 '?'는 입력 변수 그리고 '!'는 출력 변수를 나타내며, 이 결정 함수의 수행은 기존의 도메인 보안 정책과 요구 권리 정책에는 어떠한 영향도 미치지 않음을 의미하기 위해 3.1.2절과 3.1.3절에 기술된 'ΞDomainAccessPolicy'와 'ΞRequiredRights' 상태 스키마가 참조 변수로 이용된다.

접근 결정 함수인 AccessDecisionFunction 연산 스키마에서 두번째 프레디킷은 권리 조합자가 'SecAllRights'인 경우에 접근 결정이 참이되기 위한 제약조건을 정의한 것으로서, 이 조건은 'gr ⊐ rr = #rr'에 의해 정형적으로 표현된다. 이 제약조건에서 '⊐' 연산자는 Z 언어에서 기본적으로 제공하는 연산자로서, 비교할 두 집합들의 교집합이 갖는 원소의 갯수를 반환한다. 즉, 이 표기법은 사용자 권한이 갖는 권리의 집합과 인터페이스의 메소드 수행에 요구되는 권리 집합의 교집합이 반드시 요구 권리의 집합과 같아야 한다는 사실을 나타낸다. 이를 동일한 의미를 갖는 집합 표현으로 정의하면 $\forall r_i, r_j : Rights \bullet (r_i \cap r_j \subseteq r_i \wedge \# (r_i \cap r_j) = \#r_i)$와 같다. 반면에, 권리 조합자가 'SecAnyRights'인 경우는 'gr ∩ rr ⊆ rr ^ gr ⊐ rr ≥1'에 의해 그 의미가 정의되며, 이는 두 권리 집합들의 교집합이 요구 권리 집합의 부분 집합일 뿐만 아니라 부분 집합의 원소가 하나 이상인 경우만 참이 되는 조건을 의미한다. 즉, 적어도 사용자 권리 집합의 하나 이상의 원소가 요구 권리 집합에 존재해야 함을 나타낸다. 마지막으로 접근 결정 함수의 세번째 프레디킷은 위의 두가지 조건에 해당되지 않는 경우는 모두 접근이 허용되지 않도록 하는 제약사항이다.

4. 접근 제어 관리 인터페이스

접근 제어 모델의 완전성을 위해서는 권한 부여 상태에 영향을 미치는 보안 정책의 관리 인터페이스 정의가 요구되며, 여기서는 CORBA 보안 참조 모델에 제시된 도메인 접근 정책과 요구 권리 정책의 관리 메소드들을 정의한다. 접근 제어 관리를 위한 인터페이스는 시스템 관리자에 의해서 수행되며, 이는 접근 제어에 관련된 정책들을 관리하는 인터페이스이다. 이러한 관리 인터페이스를 사용하므로써 3장에서 설명된 권한부여 상태가 변경될 수 있다. [6, 7]에 기술된 CORBA 접근 제어 모델에서는 접근 제어에 관련된 도메인 접근 정책과 요구 권리 정책을 관리하기 위한 기본적인 인터페이스를 정의할 뿐 사용자의 권한을 관리하는 인터페이스는 존재하지 않는다.

4.1 도메인 접근 정책 관리 인터페이스

도메인 접근 정책을 관리하는 인터페이스의 메소드의 종류는 3가지가 있으며, 이들은 〈그림 2(a)〉에 제시된 테이블에 하나의 튜플을 추가 또는 삭제하거나 기존의 튜플을 갱신하는 연산에 해당한다. 먼저, 아래에 정의된 Grant_Rights 연산 스키마는 특정한 사용자에게 새로운 권리를 부여하는 메소드를 정의한 것으로서, 이 메소드가 수행된 후 도메인 접근 정책의 상태는 스키마의 프레디킷에 기술된 것처럼 주어진 권리가 추가되는 상태로 변경된다.

Grant_Rights

Δ DomainAccessPolicy

priv? : PrivilegeAttribute, state? : DELEGATION_STATE

rights? : Rights

HasState' = HasState ∪ {(priv?, state?)}

^ HasRight' = HasRight ∪ {(priv?, rights?)}

^ rights? = ran HasState' priv?

반면에, 특정 사용자에게 이미 주어진 권리를 철회(revoke)하는 메소드를 정의하기 위해 아래와 같은 'Revoke_Rights' 연산 스키마를 정의한다. 이 스키마의 프레디킷은 메소드의 수행 결과로 변경된 도메인 접근 정책의 상태가 초기의 상태에서 입력 인자로 주어진 권리를 제외시킨 상태로 대체됨을 정형적으로 표현한다. 세번째 연산 스키마인 'Replace_Rights'은 앞에 설명된 두 메소드들을 통합시킨 관리 메소드이며, 이는 입력 인자로 주어진 권한이 갖는 권리를 특정한 권리로 모두 대체시키는 연산이다. 따라서 이 스키마의 프레디킷에는 입력으로 주어진 권한이 기존에 갖고 있는 권리를 입력된 권리로 대체하는 '⊕' 연산자를 사용하여 그 의미를 정형화한다.

Revoke_Rights

Δ DomainAccessPolicy

priv? : PrivilegeAttribute, state? : DELEGATION_STATE

rights? : Rights

---

priv? $\in$ dom HasRight $\wedge$ state? $\in$ ran HasState priv?

$\wedge$ HasRight' = HasRight $\rhd$ (HasRight $\setminus$ {(priv?, right?)})

Replace_Rights

Δ DomainAccessPolicy

priv? : PrivilegeAttribute, state? : DELEGATION_STATE

rights? : Rights

---

priv? $\in$ dom HasRight $\wedge$ state? $\in$ ran HasState priv?

$\wedge$ HasRight' = HasRight $\oplus$ {(priv?, right?)}

4.2 요구 권리 정책 관리 인터페이스

CORBA 접근 제어 모델에서 권한부여 상태를 구성하는 하나의 요소인 요구 권리 정책의 관리는 'set__required__rights' 메소드에 의해 수행된다. 이 메소드는 응용 객체에 소속된 인터페이스에 정의된 특정 메소드를 수행하는데 요구되는 새로운 권리를 추가하기 때문에 4.1 절에서 설명된 'grant__rights' 메소드와 비슷한 의미를 갖으며, 단지 요구 권리 정책의 상태를 변경시킨다는 것만 차이가 있다. 이 메소드에 대한 연산 스키마는 다음과 같이 정의된다.

Set_ Required_Rights

Δ RequiredRights

op? : OPERATIONS, class? : CLASS

right? : Rights, combinator? : RIGHTS_COMBINATOR

HasCombinator' = HasCombinator $\cup$ {(right?, combinator?)}

HasOperations' = HasOperations $\cup$ {(right?, op?)}

$\wedge$ dom HasCombinator' = right?

HasClass' = HasClass $\cup$ {(right?, class?)}

$\wedge$ dom HasClass' = right?

IsDefined' = IsDefinded $\cup$ {(op?, class?)}

5. 결 론

CORBA 보안 참조 모델에 기술된 접근 제어 기능은 특정한 구현 메카니즘에 종속되지 않는 독립적인 개념으로 정의되기 때문에 CORBA 보안 서비스의 접근 제어 기능을 구현하려는 개발자와 구현된 서비스를 이용하는 보안 관리자가 서로 혼돈된 개념을 갖을 가능성이 존재한다. 따라서 CORBA 보안의 접근 제어 기능을 표현하는 언어가 정형화된 의미를 갖을 필요성이 있으며, 이러한 정형화된 의미를 내포하는 접근 제어 모델은 구현하는 시점에서 중요한 요소이다.

위와 같은 목적을 달성하기 위해서 본 논문에서는 CORBA 보안 참조 모델에 기술되어 있는 객체 호출시 시행되는 접근 제어 기능을 집합론에 기초한 Z 언어를 이용하여 의미가 정확히 정의되는 정형화된 접근 제어 모델을 개발하였다. 정형화된 접근 제어 모델은 권한부여 상태와 접근 결정 함수, 그리고 권한부여 상태를 변경시키는 접근 제어 관리 인터페이스로 구성되며, 이 각각의 구성요소들을 모두 Z 언어의 기본적인 구조인 상태 스키마와 연산 스키마 구조로 각기 정의하였으며, 접근 제어 모델의 각 요소들이 갖는 함축적인 의미들을 각기 스키마의 프레디킷에 자세하게 정의하였다.

본 논문에서 제시한 정형화된 CORBA 보안의 접근 제어 모델은 실제 OMG에서 표준으로 발표한 CORBA 보안 서비스의 접근 제어 기능을 설계하고 구현하는 과정에서 이용될 수 있으며, 또한 안전한 데이타베이스 관리 시스템을 개발하는데 있어서 데이타베이스의 내용을 보호하는 접근 제어 기능의 설계 과정에 응용될 수 있다.

본 논문은 실제 CORBA 보안의 접근 제어 기능을 구현하기 앞서, 구현할 모델이 갖는 함축적인 의미가 자세하게 표현된 정형화 규격을 제시하므로써 설계와 구현 과정에서 발생가능한 모호성을 제거하는 것이 일차적인 목적이다. 따라서 추후에는 본 논문의 확장 연구로서 제시된 정형화 규격에 대한 검증(verification) 단계가 추가로 수행되어야 할 것이다. 그리고 본 논문에서는 객체 호출시에 적용되는 접근 제어 기능에서 자율적인 접근 제어만을 고려하고 있으나, 앞으로는 다단계 보안 모델의 구성요소가 포함된 통합 정형화 모델의 개발이 필요하다.

참 고 문 헌

[1] A. Boswell, "Specification and validation of a security policy model," IEEE Trans. on Software Engineering, Vol. 21, No. 2, 1995, pp63-68.

[2] C. E. Landwehr, "Formal methods for computer security," ACM Comp. Surveys, Vol. 13, No. 3, 1981, pp247-278.

[3] D. C. Ince, An Introduction to Discrete Mathematics, Formal System Specification, and Z, Clarendon Press, 1992.

[4] D. M. Berry, "Towards a formal basis for the Formal Development Method and the InaJo specification language," IEEE Trans. on Sofrware Engineering, Vol. 13, No. 2, 1987, pp184-201.

[5] E. Bertino, F. Origgi, P. Samarati, "A New Authorization Model for Object-oriented Databases," Database Security VIII : Status and Prospects, Elsevier Science Publisher, 1994, pp199-222.

[6] J. Jacob, "A uniform presentation of confidentiality properties," IEEE Trans. on Software Engineering, Vol. 17, No. 11, 1991, pp1186-1194.

[7] K. Lano, Formal Object-Oriented Development, Spinger-Verlag, 1995.

[8] L. Semmens, P. Allen, "Using Yourdon and Z : An approach to formal specification," Proceedings of the Z User Worksop, Springer-Verlag, 1990, pp228-253.

[9] M. Henning, A. Rohde, "On the security of Z for the specification of verifiable secure systems," In Computer Security in the Age of Information, Elsevier Science Publisher B. V., 1989, pp197-221.

[10] Object Management Group, Object Manageme-

nt Architecture Guide, Revision 3.0, June, 1995.

[11] Object Management Group, The Common Object Request Broker : Architecture and Specification, Revision 2.0, July, 1995.

[12] Object Management Group, CORBA Security, Document 95-12-1, December, 1995.

[13] OMG Security Working Group, OMG White Paper on Security, Document 94-4-16, April, 1994.

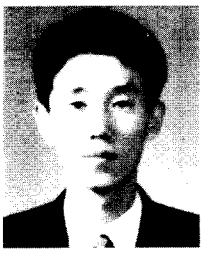
[14] Silvano Castano, Database Security, Addison-Wesley, 1995.

[15] T. Parker, D. Pinkas, SESAME V4 Overview, http://www.esat.kuleuven.ac.be/cosic/sesame.html, December, 1995.

[16] Department of Defense, Dod Trusted Computer System Evaluation Criteria (TCSEC), DoD5200.28-STD, http:// www.ovnet.com/~dckinder/documents/orangebook.htm.

[17] UK IT Security Evaluation & Certification Scheme, "Developers' Guide Part II : Reference for Developers," Uk Scheme Publication No. 4, http://www.itsec.gov.uk/, 1996.

□ 著者紹介



김 영 균

1991년 전남대학교 전산통계학과(이학사)
1993년 전남대학교 대학원 전산통계학과(이학석사)
1995년 전남대학교 대학원 전산통계학과(이학박사)
1995년 - 현재 한국전자통신연구원 선임연구원
※ 주관심분야 : 데이타베이스 시스템 보안, 분산 시스템 보안, 객체지향 데이타베이스 시스템

김 경 범

1981년 인하대학교 전자공학과(공학사)
1983년 인하대학교 대학원 전자공학과(공학석사)
1983년 - 현재 한국전자통신연구원 책임연구원, 전자계산기 기술사

※ 주관심분야 : 정보통신 시큐리티, 분산 시스템, 컴퓨터 통신

인 소 란

1978년 홍익대학교 전산학과(공학사)
1982년 홍익대학교 대학원 전산학과(공학석사)
1991년 홍익대학교 대학원 전산학과(공학박사)
1978년 - 현재 한국전자통신연구원 책임연구원, 소프트웨어공학연구실장

※ 주관심분야 : 소프트웨어 공학, 정보 보안, 프로토콜 공학, 분산 시스템