효과적인 의무 분리를 적용한 역할 기반 접근 제어 모델의 설계

김원일, 이창훈
건국대학교 컴퓨터 공학부
{unangel, chlee}@konkuk.ac.kr

Design of RBAC Applying Efficient Separation of Duty

Wonil Kim, ChangHun Lee
Dept. of Computer Engineering, Konkuk University

요 약

역할 기반 접근 제어는 유연하면서도 각 사용자의 업무에 적합한 역할을 통해 유연한 접근 제어를 제공한다. 또한 상속을 통해 역할을 유연하게 확장하고 할당할 수 있으며, 사용자가 갖는 다양한 역할을 지원하기 위해 다대다 관계를 지원한다. 여기에 역할에 포함될 수 있는 연산의 한계를 지정하여, 지정된 연산 이외의 연산을 수행할 수 없도록 정교한 접근 제어를 지원한다. 상속과 다대다의 관계를 통한 유연한 접근 제어를 제공하는 반면, 역할의 복수 할당을 통해 허용되지 않아야 하는 역할이 사용자에게 할당되는 문제점을 갖는데, 이를 해결하기 위해 역할이 갖는 연산의 한계와 역할 간의 충돌을 방지하기 위한 기능으로 의무 분리가 추가되었다. 의무 분리는 사용자에게 역할이 할당되고 적용하는 과정에서 각각 정적, 동적 의무 분리를 적용해야 하므로, 전체적인 접근 제어의 성능이 떨어지는 문제점을 갖는다. 본 논문에서는 역할의 의무 분리에서 발생하는 성능 저하 문제를 역할 할당의 과정에서 해결하여, 실제 사용시에 의무 분리를 적용하는 부하를 줄일 수 있는 역할 기반 접근 제어 시스템 모델을 제안한다.

1. 서 론

접근 제어는 시스템의 다양한 정보와 자원에 접근하기 위해 제공되는 기술적인 방법으로, 다수의 사용자가 시스템 자원을 공유하기 위해 고안되었다. 이러한 접근 제어 모델의 시초는 접근 제어 행렬을 통해 사용자와 자원을 사상하도록 구성되었고[14], 이후 접근 자격 리스트[15]를 이용하는 형태도 등장하였다. 접근 제어는 기본적으로 주체가 객체에 접근하기 위한 허가 또는 객체에 접근할 수 있는 사용자의 연속적인 배열 형태가 주류를 이루었으며, 이를 바탕으로 각 사용자가 갖는 자원의 접근 권한을 허용하거나 거부하는 형태로 구성된다[1, 3]. 최초의 접근 제어는 소수 전문가들이 사용하는 시스템의 자원 사용과 접근을 위해 주로 사용되었다. 개인용 컴퓨터의 보급과 일반적인 사용자의 증가, 그리고 인터넷의 급속적인 발달은 불특정 다수를 위한 정보 시스템의 등장을 가속화 시켰다.

본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT 연구센터 육성지원사업의 연구결과로 수행되었음.

이러한 정보시스템 자원에 대한 사용자의 접근과 관리는 기존의 접근 제어와 접근 결정으로 소화할 수 없는 막대한 양으로, 접근 제어에 의한 시스템 부하가 시스템이 제공하는 서비스 자체의 성능저하를 유발시킬 수 있었다. 이러한 문제점을 해결하고 정보 시스템 자원의 효율적인 분배와 활용을 위한 접근 제어로 역할 기반 접근 제어가 고안되었다[2]. 역할 기반 접근 제어는 기본적으로 주체가 수행하는 입력, 수정 삭제 및 조회와 같은 다양한 연산의 집합으로 구성되고, 주체는 이러한 연산의 집합인 역할을 통해 자원에 대한 접근을 요구하는 형태로 접근 제어가 수행된다. 이렇게 연산의 집합과 사용자를 연결하는 추상적인 개념을 역할이라 한다[6, 8].

역할 기반 접근 제어는 기존의 접근 제어가 접근 제어 행렬이나 접근 자격 리스트를 통해 작성되어, 자원이나 사용자 정보 변화에 따라 관련된 모든 정보를 재구성해야 한다는 문제점을 해결하였으나, 역할이 갖는 상속과 다대다의 관계로 인해 주체나 관리자가 의도하지 않은 연산을 수행할 수 있다는 문제점이 발생하였다. 따라서 주체에게 할당된 어떤 역할의 연산과 다른 역할에 포함된 연산을 주체가 동시 사용

가능하게 되는 경우를 점검하는 과정이 필요하게 되었다. 이렇게 역할이 갖는 연산의 한계를 지정하고 분류할 때, 발생할 수 있는 연산간의 충돌을 미연에 방지하는 것을 의무 분리라 한다. 의무 분리는 자원 접근에 대해 정교함을 제공하는 반면 역할에 포함된 모든 연산에 대해 세밀한 분리 작업을 요구하며, 상속을 통해 의도하지 않은 작업 수행이 묵시적으로 수행되는 것을 방지하도록 관리자의 세심한 검증을 요구한다[11].

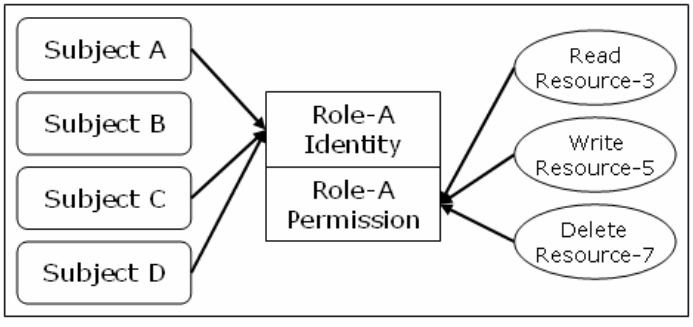
본 논문에서는 의무 분리에 의해 발생하는 성능 저하의 문제를 해결할 수 있는 역할 기반 접근 제어 시스템 모델을 제안한다. 제안 시스템은 인터넷을 기반으로 하는 정보 시스템에 적용 가능한 역할 기반 접근 제어로, 역할 할당과 이에 따른 의무 분리 문제를 해결할 수 있는 접근 제어 시스템의 설계 모델로 그 한계를 정한다.

2 장에서는 역할 기반 접근 제어의 장점과 의무 분리에 대해 알아보고, 3 장에서는 문제점을 해결하기 위한 역할 기반 접근 제어 시스템을 제안한다. 4 장에서는 결론과 향후 과제 및 연구 방향에 대해 제시한다.

2. 역할과 의무분리

1) 역할을 통한 유연한 접근 제어

기존의 접근 제어는 주체와 객체의 관계를 리스트와 같은 배열의 형태로 구성하였기 때문에, 포함된 정보의 변경이 발생하면 관련된 모든 정보를 변경해야 하는 문제점이 있다. 따라서 조직이나 회사와 같이 자원과 사용자의 정보 변화가 비일비재한 정보 시스템에 적용하면, 자원과 사용자의 정보 관리를 위해 과다한 자원을 소비해야 하는 문제점이 발생한다. NIST 는 이러한 문제를 인식하고 정보 시스템과 같이 자원과 사용자의 정보가 자유롭고 동적으로 변화하는 시스템에 적용할 수 있는 역할 기반 접근 제어를 고안하였다[2]. 역할은 주체와 연결되어 있으면서 또한 자원과도 연결되어 있는 추상적 개념으로 사용자나 자원의 변경에 대한 관리 문제를 해결하는 방법을 제공한다[6].



<그림 1> 주체와 객체 및 역할의 관계

먼저 역할은 <그림 1>과 같이 주체와 객체를 사상한다. 시스템에서 주체의 역할이 변경되어야 하는 경우, 주체를 해당하는 역할에 사상하는 것으로 간단히 주체의 정보 변경이 가능하다. 즉, 역할은 주체의 변경에 대해 변화 없이 존재할 수 있으며, 자원에 대한 주체의 접근 허가 변경이 필요한 경우에는 해당 자원이 연결되어 있는 역할의 접근 권한을 변경하는 것으로 해당 역할을 수행하는 전체 주체에게 동일한 권한의 적용이 가능하다. 또한 역할이 접근할 수 있는 자원과 필요한 권한을 설정하는 것으로 간단히 역할을 생성할 수 있으며, 생성한 역할을 주체에게 할당하는 것으로 접근 제어를 위한 주체의 역할을 추가할 수 있다. 여기에 자원의 효율적인 통합과 공유를 위해 상속의 개념을 지원한다[2]. 역할 기반 접근 제어는 이러한 역할과 자원의 다대다 관계 및 역할의 상속을 통해 역할의 구성과 할당을 유연하게 하였으며, 할당된 역할에 포함된 연산 외의 연산은 수행할 수 없도록 강건하게 구성되어 있다[2, 5].

이렇듯 역할과 역할이 갖는 특성을 통해 유연한 접근 제어를 제공할 수 있는 반면, 상속은 동일한 작업에 대한 지침과 기능만을 제공할 뿐, 역할간의 간섭에 대한 제약이 없고, 관리자가 정한 규칙에 의해 상속의 가능 여부가 결정된다. 그러나 다대다 관계로 인해 사용자가 갖는 역할에 포함해서는 안 되는 역할이 할당되어, 특정 사용자가 허용된 권한 이외의 특권을 할당 받는 문제가 발생할 수 있으므로, 역할 간에 발생할 수 있는 역할의 침범과 권한의 허용 범위 문제를 해결하기 위해 의무 분리가 추가되었다. 즉, 역할이 갖는 권한이 허용된 범위를 벗어나지 않도록 하는, 역할의 잠금 장치로 의무 분리를 적용하여 접근 제어의 유연함에 정교한 접근 제어를 제공한다.

역할 기반 접근 제어에서는 사용자에게 할당되는 역할의 수에 제한이 없지만, 특정 순간에 활성화되는 역할은 반드시 하나여야 한다는 규정을 통해 역할의 동시 활성을 방지하여 의무 분리 문제를 해결하는 방법을 제공하고 있다[2, 16].

2) 역할 간의 의무 분리

의무 분리는 말뜻 그대로 사용자에게 할당된 특정 역할이 할당된 다른 역할에 의해 필요 이상의 권한이 주어지는 것을 방지하도록 역할의 한계를 규정 짓고 할당이 불가능하도록 구성하는 것을 의미한다.

예를 들어, 회계 출납을 담당하는 사용자가 상속을 통해 입금 연산과 업무가 상충되는 출금 연산이 가능하도록 역할이 구성되어 있다면, 입금에 대한 처리와 출금에 대한 처리를 동시에 수행할 수 있기 때문에 자금의 무결성 문제가 발생할 수 있다. 즉, 역할은 각 사용자가 갖는 고유한 업무를 대변할 뿐만 아니라 사용자가 수행할 수 있는 연산의 한계를 동시에 지정하는 것과 같다. 이로 인해 역할의 상속은 본

목적과는 다르게 상호 배제되어야 할 역할이 묵시적으로 할당되는 결과를 발생시킬 수 있다[3].

또한 역할을 통해 자원과 사용자는 다대다의 관계를 가질 수 있으므로, 의무 분리가 적용되지 않은 역할을 할당하거나 사용하는 것은 의도하지 않은 작업을 수행할 수 있는 통로를 제공한다[11].

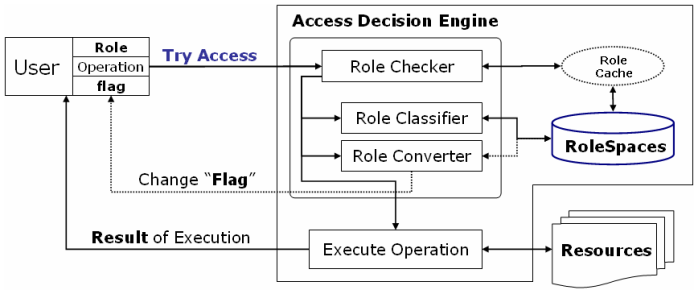
예를 들어, 역할 A 가 자원 K 에 대해 읽기 연산만을 수행할 수 있고, 역할 B 는 자원 K 에 대해 읽고, 쓰는 연산이 가능하다고 가정하자. 사용자 C 는 기본적으로 역할 A 를 통해 자원 K 에 읽기 연산만이 가능하도록 관리자가 업무를 규정하였지만, 사용자 C 가 갖는 업무의 특성상 역할 B 도 동시에 수행해야 하는 경우, 사용자 C 는 본래 규정지어진 업무의 범위를 벗어나 관리자가 의도하지 않은 자원 K 에 대한 쓰기 업무도 가능해진다는 것이다.

따라서 역할 기반 접근 제어는 자원과 사용자 간의 다대다 관계를 지원하고 상속을 허용하기 위해서 필수 불가결하게 의무 분리를 제공해야 한다. 의무 분리는 단순히 사용자에게 할당된 역할이 연산의 허용 한계를 벗어나는가를 점검하는 것을 벗어나 상속 등을 통해 묵시적으로 허용되는 모든 경우를 점검할 수 있어야 한다. 이를 위해 등장한 것이 실행 시간의 사용자 세션을 점검하는 동적 의무 분리와 현재 역할이 의무 분리에 적합한가 여부를 결정하는 정적 의무 분리 기능이다[4, 13].

본 논문에서는 이렇게 발생하는 다양한 역할 간의 의무 분리 문제를 해결하기 위한 역할의 할당과 유연한 역할의 변경을 지원하기 위한 역할 정보의 분배 방법을 통해 역할 기반 접근 제어의 성능 향상과 역할과 의무 분리의 체계적인 관리가 가능한 시스템을 제안한다.

3. 제안 모델

본 논문이 제안하는 역할 기반 접근 제어 시스템은 역할이 사용자에게 독립적으로 할당되어 다대다의 관계 및 상속을 통한 역할의 묵시적 할당을 통해 의무 분리가 부서지는 문제점을 해결하기 위해 <그림 2>의 구조를 갖는 역할 기반 접근 제어 시스템을 제안한다.

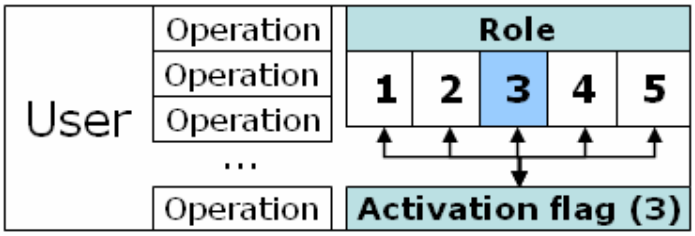


<그림 2> 제안된 역할 기반 접근 제어 시스템

시스템은 역할과 의무분리에 관련된 정보를 저장하기 위한 *RoleSpaces* 와 자원 접근에 대한 허용과 거부를 결정하는 *Role Checker*, 사용자의 역할을 자동으로 변경하는 *Role Converter* 그리고, 역할의 정적이며 동적인 의무 분리 분류를 담당하는 *Role Classifier* 로 구성된다. 여기에 역할의 효율적인 활용과 주체의 역할 변경 시의 성능 향상을 위해 주체 영역에 역할 저장소를 생성하는 것을 골자로 한다.

1) 사용자 역할 저장소

사용자 혹은 사용자 프로그램에서는 사용자의 로그인 후, 해당 사용자의 사용 빈도수가 높은 5 개의 역할이 사용자 혹은 프로그램에 직접 할당된다. 빈도가 높은 역할을 미리 할당하게 되면, 이후 역할의 변경을 위해 역할 정보를 얻기 위한 서버 접속과 같은 불필요한 성능 저하를 막을 수 있다[12]. 빈도수가 높은 역할이 사용자에게 이미 할당되어 있다면, 접근하고자 하는 자원에 대해 현재 보유한 역할의 변경을 통해 접근할 수 있다. 사용자 역할 저장소의 구조는 <그림 3>과 같다.



<그림 3>사용자 역할 저장소의 구조

Activation Flag 는 할당된 5 개의 역할 가운데 사용자가 사용하고 있는 역할을 구분해주는 동시에 현재 역할을 대변하고 있다. 즉, 사용자 역할을 내부적으로나 외부적 요인으로 인해 변경해야 하는 경우, 현재 역할을 가리키고 있는 *Activation Flag* 를 변경하는 것으로 손쉽게 사용자의 역할을 변경할 수 있으며, 역할 변경 자체가 간단하기 때문에 변경이 필요한 시점에 추가적인 처리 없이 변경 작업을 수행할 수 있다[12].

2) *Role Checker*

본 논문이 제안하는 접근 제어 시스템의 핵심 기능은 *Role Checker* 가 담당하며, 현재 *Context* 정보를 바탕으로 요구한 연산의 수행 가능 여부를 결정한다. 또한 *Role Checker* 는 접근 제어가 수행되는 시작점이자 각종 요구에 대해 적절한 모듈로 *Context* 를 전달하는 시스템의 중심 기능을 담당하며 다음과 같은 기능을 수행한다.

- 사용자의 로그인 정보를 통한 사용자 인증
- 자원 접근의 허용과 거부를 결정

- 허용된 연산을 실제 수행 모듈로 전달
- 거부 결정된 접근을 역할 변경 모듈로 전달
- 사용자 역할의 할당 검증 모듈로 전달

세부적으로 살펴보면 사용자가 정보 시스템 자원을 사용하기 위한 인증과 정당한 사용자인 경우에 5 개의 역할과 가장 최근에 사용한 역할을 *Activation Flag* 에 할당하며, 사용자의 자원 접근 요구에 대해 접근의 허용과 거부를 결정하는 기능을 담당한다. 여기에 접근의 허용과 거부에 대해 적절한 시스템의 기능으로 *Context* 를 전달하는 접근 결정 기능을 주로 수행한다. 여기에 사용자에게 할당하려는 역할이 보유하고 있는 역할과 의무 분리가 수행될 수 있도록 검증 모듈로 전달하는 핵심 기능을 담당한다.

3) *RoleSpaces*

본 논문에서는 사용자에게 할당된 역할과 의무 분리에 관련된 정보가 저장된 데이터 영역을 *RoleSpaces* 라 정의하는데, *RoleSpaces* 는 단순히 역할과 의무분리에 대한 정보를 저장하기 위한 단순 저장소의 개념이 아니라, 포함된 정보를 사용자 역할 저장소에 그대로 적용할 수 있는 일원화된 모델이다. 접근 제어를 위해 생성되는 테이블은 사용자에게 할당되는 역할을 저장하는 *Role_Space* 테이블과 역할 간의 의무 분리 정보를 포함하는 *Separation_of_Duty* 테이블이다. 역할은 각종 정보 시스템뿐만 아니라 다양한 형태의 응용 프로그램에서 동작할 수 있어야 하므로 데이터베이스 테이블 형태로 구성한다.

즉, 정보 시스템에 기본적으로 포함된 사용자 정보뿐만 아니라 사용자가 활용할 수 있는 역할에 대한 정보를 더하고, 여기에 역할 간의 의무 분리 정보를 포함하여 사용자가 자원에 접근하는 모든 경우에 적용이 가능하도록 구성한 데이터의 저장 및 관리 모델이다. 물론 이 모델은 파일 시스템이나 세션 또는 XML 을 이용한 모든 형식에 적용이 가능하다. *Role_Space* 는 <그림 4>와 같이 사용자를 구분을 위한 사용자 구분자, 최근에 사용한 역할 저장을 위한 *Activation Flag* 정보 그리고, 사용자가 활용할 수 있는 역할의 리스트로 구성된다. 로그인 시에 사용자에게 바로 할당되는 5 개의 역할을 비롯한 *Activation Flag* 정보는 *Role_Space* 테이블 구조와 동일한 구조임을 알 수 있다. 따라서 빈도가 높은 역할을 미리 저장하고 있다면, 추후 발생할 수 있는 *RoleSpaces* 의 접근 횟수를 줄일 수 있다.

USER-ID	Last Used Flag	Available Role - 1	Available Role - 2	...	Available Role - N
USER	Flag	Role-A	Role-B		Role-O

<그림 4> *Role_Space* 테이블 정의

본 시스템에서는 사용자에 부여되는 역할이 단순히 역할 저장을 위한 위치의 선정으로 결정되고 할당되지 않는다. 먼저 현재 할당하려는 역할이 이미 포함되어 있는 역할과 의무분리에 위배되는지 여부를 확인하고, 의무분리가 준수되는 경우에만 해당 역할이 사용자에게 할당된다. 즉, 역할의 할당 과정 중에 현재 사용자의 모든 의무 분리가 점검되므로, 할당되는 모든 역할은 묵시적이나 명시적으로 의무 분리를 위반하지 않음을 보장한다. 이러한 의무 분리의 점검은 <그림 5>에 나타난 의무 분리 테이블을 기반으로 수행된다. 의무 분리 작업을 위한 테이블은 관리자에 의해 생성되고, 관리되며 역할의 할당 과정에서만 사용되므로, 의무 분리를 위한 점검 시간을 줄일 수 있다.

SUBJECT	OBJECT - 1	OBJECT - 2	...	OBJECT - N
Role-Z	Role-P	Role-A		Role-O

<그림 5> *Separation_of_Duty* 테이블 정의

즉, 사용자에게 할당되는 모든 역할은 *RoleSpaces* 에 존재하는 정보에 의거하여 역할 할당 여부를 결정하여 구성된다. 이것은 의도적이지 않은 역할의 할당을 통해 사용자가 할당 받아서는 안 되는 역할을 정교하게 구분할 수 있도록 하며, 사용자의 역할에 맞는 연산과 자원 접근을 허용하도록 구성할 수 있는 기초를 제공한다. 나아가 역할이 갖는 정교한 자원 접근 권한을 제공할 수 있다. 또한 의무 분리를 실행 시간 중에 점검하지 않도록 하므로, 의무 분리에 의해 발생하는 성능 저하를 막을 수 있다.

4) *Role Classifier*

주체는 자원 접근을 위해 하나의 역할만을 사용하지 않고 할당되어 있는 다수의 역할에서 필요한 역할을 선택하여 이용할 수 있는데, 이것은 역할 기반 접근 제어가 기본적으로 지원하는 자원과 사용자 간의 다대다 관계 및 상속에 의거한다[4].

역할 기반 접근 제어는 역할 자체가 갖는 추상적인 개념으로 인해 각각의 역할이 다른 역할에 대해 항상 독립적으로 존재할 수 없으므로 역할 할당과 자원 사용의 효율성을 높이기 위해 상속의 개념을 지원한다. 대부분의 상속은 자원의 효율적인 접근과 자원 활용의 측면에서 트리 계층과 역 트리 계층으로 표현되며, 상속을 이용하면 역할의 관리와 할당을 통해 자원에 대한 사용자들의 접근 방법이 일원화되어 자원이 각각 독립적으로 존재하는 것이 아니라, 자원의 통합과 공유가 가능해지므로 관리상에 편리함을 제공하는 장점이 있다[1, 2, 7]. 상속은 주로 특정한 역할이 기본적으로 포함해야 하는 역할에 대한 템플릿을 생성하고, 해당 역할이 필요한 사용자에게 할당하는

경우에 적합하다. 그러나 상속 적용을 통해 자원 접근의 의무 분리 문제가 발생할 수 있고, 이중에서 가장 많이 발생하는 문제는 현재 사용자에게 할당되어 있는 역할이 적용해야 하는 역할과 의무 분리 침해가 발생한 경우, 점검을 위한 기능을 추가하지 않는다면 여과 없이 역할이 할당되는 것이다. 즉, 의무 분리를 해야 하는 역할들이 묵시적으로 허용되어 시스템에 문제를 야기할 수 있다는 것이다.

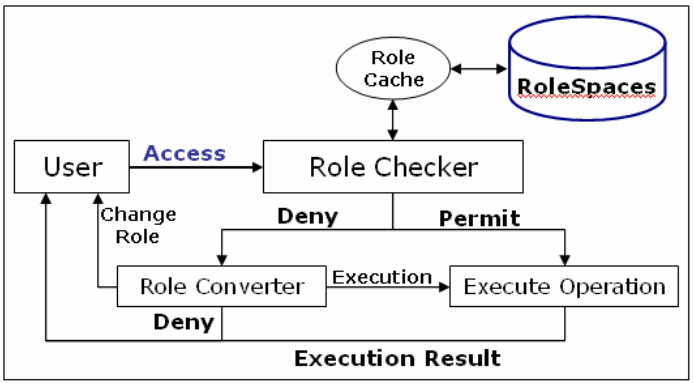
Role Classifier 는 이러한 문제를 해결하기 위해 *Role_Space* 에 할당하려는 역할이 현재 사용자가 보유하고 있는 역할과 의무 분리에 위배 되는가 여부를 점검한다. *Role Classifier* 는 시스템 관리자가 사용자에게 역할을 할당하는 경우에만 기능을 수행하므로, 역할을 할당하는 과정 이외의 모든 접근 제어 시스템의 기능 수행 중에 발생하는 의무 분리 점검에 의한 부하를 줄일 수 있다. 이것은 관리자 이외의 사용자가 임의로 접근할 수 없으며, 역할의 일반적인 사용에 의무 분리를 점검하는 일이 발생하지 않는다는 것을 의미한다. *Separation_of_Duty* 테이블에 의해 의무 분리 점검되고, 그 결과에 의해 의무 분리에 문제가 없는 역할만이 *Role_Space* 에 입력되기 때문에 의무 분리의 문제는 사용자에게 역할을 할당하는 작업의 수행을 통해 해결된다.

이렇게 *Role_Space* 와 *Separation_of_Duty* 테이블에 기초하여 동작하는 *Role Classifier* 는 역할 간의 의무 분리를 정교하게 수행하므로, 시스템에 존재하는 모든 사용자가 각각 보유하고 있는 역할의 리스트가 의무 분리에 의한 문제점을 야기하지 않는다는 것이다. 또한 의무 분리에 대한 묵시적이거나 명시적인 작업이 모두 점검되므로 의도하지 않는 작업의 발생을 미연에 방지할 수 있는 장점이 있다.

5) *Role Converter*

사용자는 로그인 시에 가장 최근에 사용한 역할이 기본적으로 할당되며, 이 정보는 세션이나 프로그램과 같은 사용자 영역에 저장된다. 저장된 역할 정보는 한 순간에 하나만 활성화 되어야 하므로, 필요한 경우에 역할을 변경해주어야 한다[12]. 이러한 작업은 사용자가 특정 역할을 수행하고자 할 때, 명시적으로 사용자가 변경하도록 구성되어 있는데, 이러한 경우 특정 작업을 하려는 사용자는 항상 작업에 대한 정보를 숙지하고 있어야 하며, 필요한 시점에 시기 적절한 역할의 변경을 수행할 수 있어야 한다. 그러나 다양하며 또한 많은 수의 자원의 접근을 위한 정보 시스템에서는 이러한 정보를 숙지하기 어렵다는 단점이 있다. 즉, 사용자가 수행해야 하는 작업보다 작업을 수행하기 위해 필요 이상의 부가 정보를 취득하고 있어야 한다는 것이다. *Role Converter* 는 사용자가 수행해야 하는 작업에 따른 역할 변경을 자동으로 수행하는 기능을 담당한다.

예를 들어 *Role Checker* 가 자원 접근에 대해 거부를 결정하면, 결과가 바로 사용자에게 반환되는 것이 아니라 *Role Converter* 에게 해당 작업의 수행을 결정하도록 현재 *Context* 정보를 전달한다. *Role Converter* 는 전달받은 *Context* 정보를 이용하여 현재 사용자가 다른 역할로 현재 *Context* 를 수행할 수 있는 역할을 보유하고 있는지 여부를 확인한다. 즉, 현재 역할로는 불가능하나 할당되어 있는 다른 역할로 요구한 연산을 수행할 수 있는 경우, 역할의 변경을 통해 *Context* 를 수행하고, 역할 정보 저장소에 변경 내용을 알리는 기능을 수행한다.



<그림 6> *Role Checker*의 접근 결정

로그인 과정에서 빈도수가 높은 5 개의 역할을 사용자에게 할당한 것은 <그림 6>에서 발생하는 접근 거부 시에 *RoleSpaces* 의 접근 빈도를 줄이기 위한 방법이다. 즉, 접근이 거부된 경우, 해당 사용자가 수행하고자 하는 접근에 대한 역할은 대부분 많이 수행하는 역할 또는 거의 사용하지 않는 역할로 볼 수 있다. 이를 통해 많이 수행하는 역할을 미리 사용자에게 전달하면 *RoleSpaces* 에 대한 접근 횟수를 줄 일 수 있다[12].

예를 들어 *Role Checker* 에서 자원 접근이 거부된 경우, *RoleSpaces* 에서 요구한 사용자가 정당한 사용자가 아니며, 자원에 접근하기 위해서 필요한 역할이 반환되는데, 만약 사용자에게 해당하는 역할이 할당되어 있지 않다면 접근 불가능 메시지를 전달한다. 만약 사용자의 빈도수 높은 5 개의 역할이 아닌, 즉 빈도수가 낮은 역할인 경우에는 역할 변경을 위한 정보가 반환된다. 이렇게 반환된 접근 가능한 역할을 사용자의 원 접근 요구에 포함하여 *Role Converter* 로 전달한다. 거부된 접근 요구는 *Role Cache* 에 보관되어 동일한 사용자의 다른 요구가 있을 때까지 유지된다. 거부된 접근에 대한 처리는 먼저 사용자에게 다른 역할을 이용하여 접근 하도록 필요한 역할을 전달하는데, 이 정보에 의해 사용자의 역할이 변경되면 사용자는 정당한 역할을 통해 자원에 접근할 수 있게 된다. 따라서 사용자는 보유한 역할을 최대한 활용할 수

있으며, 사용자의 명시적인 역할 변경 없이도 시기 적절하게 역할 변경이 가능하다.

4. 결론 및 향후 연구

본 논문에서는 역할 기반 접근 제어를 적용한 정보 시스템의 인증 및 허가의 설계와 이를 구현하기 위한 방법을 제안하였다.

제안된 시스템은 역할의 효과적인 역할 변경을 위해 로그인 시에 빈도수가 높은 역할을 미리 할당하여 적절한 시기에 역할이 변경될 수 있도록 사용자의 역할 저장소를 구성 하였다. 여기에 접근 제어를 위한 접근 결정 엔진인 Role Checker 가 전체 시스템을 관리하고, *RoleSpaces* 를 통해 사용자의 역할과 역할 간의 의무 분리에 대한 기반 정보를 작성하였다. 접근 결정의 거부 결정을 통해 사용자의 역할을 변경하는 *Role Converter* 가 할당된 역할을 시기 적절하게 변경하도록 하였으며, 의무 분리가 적용된 역할의 할당을 위해서 *Role Classifier* 를 이용하였다. 즉, 사용자가 갖는 역할이 서버에만 존재하는 것이 아니라 사용자에게도 할당되어 유연한 역할 변경이 가능하도록 하였으며, 역할이 할당되는 과정에서 의무 분리 문제가 해결되므로, 할당된 역할은 의무 분리에서 발생할 수 있는 의도하지 않는 연산의 수행이 실제로 발생하기 전에 방지할 수 있다. 그리고, *Execute Operation* 은 엔진에서 수행하는 것이 아니라 실제 수행하고자 하는 작업 그 자체를 가리키는 것으로 엔진과는 별도로 원래 시스템이 제공하는 정보 시스템의 기능을 나타낸다.

따라서 전체 시스템은 기존의 역할 기반 접근 제어보다 관리 차원의 의무 분리에서 많은 이득을 취할 수 있으며, 이를 통해 할당된 역할은 의무 분리 문제가 포함되지 않은 역할로 사용자 측면에서도 안전하게 자원과 정보에 대해 접근할 수 있어 시스템의 전체적인 성능 향상이 가능하다.

최근 NIST 는 RBAC 을 기본으로 하는 EDAC 를 발표했는데, EDAC 는 Enterprise 환경에서 사용 가능한 역할 기반 접근 제어이다[9]. 이렇듯 역할 기반 접근 제어는 활용될 수 있는 폭이 넓고 유연하면서도 정교한 제어를 제공하기 때문에 더 크고 복잡한 정보 시스템에서 사용하기 위한 노력이 지속되고 있다. 이것은 비단 Enterprise 환경에서뿐만 아니라, 개인 사용자를 위한 유비쿼터스 환경에서도 연구가 진행되고 있다[10]. 앞으로의 컴퓨팅 환경은 개인 사용자를 위한 유비쿼터스 환경이 주류를 이루겠지만, 크게는 정보 시스템의 범주에 포함된다. 즉, 현재의 역할 기반 접근 제어는 모든 정보 시스템에 적용이 가능하며 앞으로의 모든 컴퓨팅 환경에서도 적용이 가능할 것이다.

앞으로의 연구는 역할 기반 접근 제어 시스템의 역할 변경 효율성을 높이고, 광범위한 컴퓨팅 환경에서 사용할 수 있는 접근 제어로 RBAC 엔진을 개선하여 역할의 유연하면서도 효과적인 역할 변경 방안에 대해 연구해야 할 것이다.

참고문헌

[1] U.S. Department of Defense. Department of Defense Trusted Computer System Evaluation Criteria, DOD 5200.28-STD, National Computer Security Center, 1985.

[2] 김학범, 김동규: RBAC 표준 참조 모델 연구동향. 情報保護學會誌. 2000.

[3] Ravi Sandhu, David Ferraiolo, Richard Kuhn: The NIST Model for Role-Based Access Control; Towards A Unified Standard. 2000.

[4] Wilfredo Alvarez: Presentation on RBAC standard[8].

[5] 유두규, 문봉근, 전문석: PMI 기반의 RBAC 을 이용한 NEIS 의 DB 보안 구현. 情報保護學會論文誌. 2004.

[6] Ravi S. sandhu, Edward J.Coyne, Hal L. Feinstein and Charles E. Youman, "Role-Based Access Control Models," IEEE computer, 1996.

[7] 이철원, 이병각, 김기현, 박정호, 이홍섭, 최용락: 역할 속성을 이용한 역할기반 접근통제 매커니즘. 情報保護學會論文誌. 1998.

[8] http://csrc.nist.gov/rbac/

[9] Richard Fernandez: Enterprise Dynamic Access Control Version 2 Overview. NIST. Jan 1, 2006.

[10] 문창주: 유비쿼터스 컴퓨팅 환경에서 프라이버시 보호. ICAT. 2006.

[11] 심완보, 박석: 애드호크러시 조직의 특성을 고려한 역할기반 모델, 정보보호학회논문지. 2002. 8.

[12] 김원일, 하홍준, 이창훈: RBAC 을 위한 역할 정보 저장소의 설계, 한국 정보처리 학회. 2006. 5.

[13] W.A.Jansen. Ingeritance Properties of Role Hierachies, 21th National Information Systems Security Conference. Oct, 1998.

[14] Lampson, Butler W.: "Protection". Proceedings of the 5th Princeton Conference on Information Sciences and Systems. 1971.

[15] Levy, Henry M.: Capability-Based Computer Systems. Digital Equipment Corporation 1984.

[16] INCITS BSR 359. December 31,. 2001