SMART 이동 에이전트 시스템의 안전한 자원 접근 정책

유양우° · 김진홍 · 이명재 · 박양수 · 이명준
울산대학교 컴퓨터 · 정보통신공학부

Secure Resource Access Policy in the SMART Mobile Agent System

Yang-Woo Yu° · Jin-Hong Kim · Myeong-Jae Yi · Yang-Soo Park · Myung-Joon Lee
School of Computer Engineering & Information Technology, Univ. of Ulsan.

요 약

이동 에이전트는 사용자를 대신하여 자신의 작업을 수행하기 위하여 분산된 시스템을 자율적으로 이동할 수 있는 하나의 객체이다. 이러한 에이전트 패러다임은 분산 시스템의 자원을 사용하고자 할 때 심각한 문제를 야기 시킬 수 있으므로 신중하게 고려되어야 한다. 본 논문에서는 현재 다양한 응용분야에서 많이 사용되고 있는 에이전트 시스템의 보안정책을 설명하고, 그 장단점을 비교 분석하였다. 또한 이질적인 에이전트 시스템간의 상호운용성(interoperability)을 지원하기 위하여 개발한 SMART 시스템의 자원 접근 정책을 자세하게 설명할 것이다. SMART 시스템은 JDK1.2 보안 메커니즘을 기반으로 하였으며, 독자적인 자원 접근 정책으로 자원등급에 따른 차별적인 서비스를 제공하는 모델을 제시한다.

1. 서 론

이동 에이전트를 기초로 한 프로그래밍은 이질적인 네트워크로 구성된 인터넷 환경에서 분산 처리를 하기 위한 새로운 패러다임이다. *이동 에이전트(mobile agent)*는 컴퓨터 네트워크에서 자신의 작업을 수행하기 위하여 노드와 노드사이를 자율적으로 이동할 수 있는 사용자 대행 프로그램을 말한다. *이동 에이전트 시스템*은 이동 에이전트를 인터넷상에 보내어 그들이 미리 설계된 경로를 따라 가거나 그들 자신이 모은 정보에 의해서 직접 동적으로 경로를 설정하여 돌아다닐 수 있도록 만든다. 이들 에이전트들은 그들의 목표를 달성했을 때 그 결과를 사용자에게 전달하기 위하여 그들의 홈 사이트로 돌아오게 된다. 그들의 작업은 에이전트 프로그래밍에 의해서 결정되고, 그 응용은 인터넷 전자 상거래에서 실시간 디바이스 컨트롤, 과학적인 계산을 요하는 분산처리에 이르기까지 다양하다[1].

이동 에이전트 시스템은 에이전트들을 수행시키기 위한 기능을 제공한다. 이러한 시스템들은 악의의 에이전트 또는 바이러스에게 시스템 침입에 따른 위험으로부터 드러나 있다. 다양한 대응책에도 불구하고 그러한 에이전트들은 민감한 데이터를 파괴시키거나, 시스템의 정상적인 기능을 혼란시킨다. 그 뿐만 아니라 그들은 시스템의 CPU 사용과 기억 공간 자원들을 과도하게 소비시키기도 한다. 그 결과 다른 적법한 사용자에게 사용 기회를 제공하지 못하게 되는 결과를 초래한다. 이러한 문제를 해결하기 위하여 이동 에이전트 시스템을 개발 시 보안에 주된 관심을 갖어야한다.

대부분의 현재 이동 에이전트 시스템들은 설계 시 기본적인 요구사항으로 보안을 신중하게 고려하지 않고 있다. 여러 가지 경우에, 보안 요소는 완전히 결여되어 있거나, 기본 구조에 덧붙여진 형태 또는 빈약한 형태로 통합되어 있다. 본 논문에서는 보안의 필요성을 설명하고, 현재 가장 많이 이용되고 있는 다양한 이동 에이전트 시스템들의 보안 요소를 비교한다. 각 시스템은 기본적인 JDK 보안 관리자를 이용한 시스템과 이를 확장해서 사용하는 시스템, 그리고 독자적인 정책을 사용하는 세 가지 형태의 에이전트 시스템을 분석한다. 그리고 우리가 개발한 *SMART 이동 에이전트 시스템*을 소개하고 SMART 시스템에서 적용한 보안 모델을 설명할 것이다 [7].

본 논문의 구성은 다음과 같다. 2장에서는 다양한 이동 에이전트 시스템들의 자원 접근 정책을 살펴보고, 그 특성을 비교한다. 3장에서는 SMART 시스템을 소개하고 적용된 자원 접근 정책에 대해 설명한다. 끝으로 4장에서 결론 및 향후 연구방향에 대해 살펴본다.

2. 이동 에이전트 시스템들의 자원 접근 정책

2.1 IBM: Aglets

†Aglets의 에이전트는 라이프사이클 동안에 자신의 작업을 수행하기 위하여 파일이나 쓰레드와 같은 로컬 자원을 사용한다 [3]. 그리고 사용자와 상호작용하기 위하여 대화 창을 만들 수도 있으며, 어떤 병행적 작업(concurrent task)을 수행하기 위하여 새로운 쓰레드를 생성할 수도 있다. 이러한 자원들은 각 에이전트에게 할당된 *ResourceManager 객체*에 의해서 관리된다. 에이전트가 다른 사이트로 이동하거나, 비활성, 또는 소멸되면 그 자원들은 즉각 멈추고 소멸된다. 파일과 소켓같은 자원들은 ResourceManager에 의해서 관리되지 않는다. 이는 프로그래머가 수동적으로 직접 자원을 회수해야 한다.

† 본 연구는 정보통신진흥원의 '99년도 정보통신 우수시범학교 지원사업의 지원으로 수행되었음.

에이전트가 자바 속성(property), 쓰레드 또는 파일과 같은 내부 자원을 액세스하고자 할 때, 주어진 사용승인(permission)하에 제어되어야 한다. 그 사용승인은 GUI 또는 정책(policy) 데이터베이스를 직접 편집하여 명시할 수 있다. Aglets에서 사용하는 정책 데이터베이스의 형식은 JDK1.2 명세에 맞추어 설계되어 있으며, 사용자는 정책 데이터베이스 내에 다음과 같은 사용승인들을 명시할 수 있다.

```
java.io.FilePermission          : File read/write/execute
java.net.SocketPermission       : Socket resolve/connect/listen
                                  /accept
java.awt.AWTPermission          : showWindowWithoutWarningBanner,
                                  accessClipboard
java.util.PropertyPermission    : Java property
java.lang.RuntimePermission     : queuePrintJob,
                                  load library
java.security.SecurityPermission : getPolicy,
                                  setSystemScope
java.security.AllPermission     : all other permissions
```

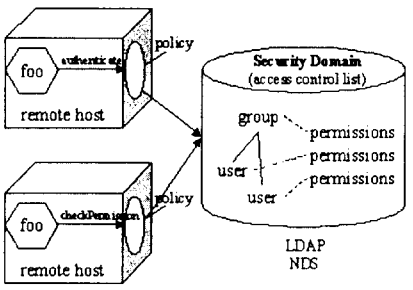
[그림 1] Aglets의 자원에 대한 사용승인

Aglets 시스템은 코드 서명을 지원하지 않고, 영역 기초(domain-based) 정책을 현재까지는 제공하지 않고 있다.

2.2 ObjectSpace: Voyager

Voyager는 자바 보안 메커니즘인 모래상자(sand-box) 모델을 기초로하고 있다 [4]. 서명자(signer) 객체가 해당 클래스를 "trust"라는 태그를 붙임으로서 신뢰성있는 클래스로 간주하고, 원래의 클래스 로더에 의해 적재된 클래스들은 명백한 "trust"가 된다.

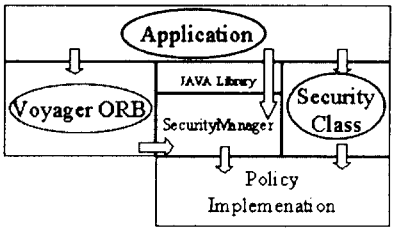
Voyager에서, 인증된 개체는 보증자(principal)로 표현한다. 보증자는 문맥(context)에 따라 쓰레드의 수행과 관련이 있다. 이러한 쓰레드가 원격 객체의 메쏘드를 실행시키기 때문에, 이와 관련된 문맥은 그런 메쏘드 호출과 함께 여러 가상 머신으로 이동한다. 이 시스템에서 보증자는 java.security.Principal을 구현한 것이다.



[그림 2] Voyager의 사용승인 정책

Voyager는 보안 라이브러리에서 제공하는 표준 인터페이스를 이용하여 인증과 사용승인 검증이 가능하다. 이 라이브러리는 여러 개의 컴포넌트를 포함하고 있다. [그림 2]는 Voyager의 정책을 보여주고 있고, [그림 3]은 Voyager ORB와 특정 애플리케이션과 함께 Voyager 보안 라이브러리의 전체 컴포넌트들을 설명하고 있다. 주어진 애플리케이션은 JDK1.2 SecurityManager를 통해 Policy 사용승인 검사를 암시적으로 적용되고, Security Class API를 이용하여 사용승인을 명백하게 검증할 수 있다.

모든 Voyager 보안 호출과 기능은 Voyager 보안 클래스 API를 이용하여 시작된다. 시작 시, Security 클래스가 특정 정책을 로딩하여 시스템을 초기화하고 보안 소유자를 정한다. 모든 보안 영역은 특정 개체를 인식하고 그들에게 부여된 권한을 관리하기 위하여 Policy 파일을 유지한다.



[그림 3] Voyager 보안 라이브러리의 컴포넌트

Voyager 보안 라이브러리의 Policy 추상클래스를 상속받고 SecurityContext 인터페이스를 구현함으로써, 이러한 보안 영역은 애플리케이션에서 적용될 수 있다. Security 클래스는 또한 쓰레드, 프록시와 관련된 본인을 관리하기 위한 기능 또한 제공한다. Voyager 라이브러리의 Security 클래스는 SecurityContext 의 구현과 쓰레드 사이의 밀접한 관련성을 유지하고 이러한 SecurityContext 구현이 JVM들 사이를 돌아다니게 된다. 앞에서 설명했던 대로 SecurityContext가 구현되었을 때, Principal 인스턴스가 이동할 수 있다

2.3 IKV++ GmbH: Grasshopper

Grasshopper의 액세스 제어 정책은 특정 보안 영역 내에서 보안 요구사항을 제시한다 [5]. 이는 일반적으로 검사가 일어날 때마다 액세스 제어 결정 기능에 따라 행동하는 몇 개의 규칙으로 구성되어 있다. 내부 보안은 에이전트가 권한 없는 액세스로부터 에이전시(agency)의 자원을 보호하기 위한 것이다. 더 나아가, 이는 에이전트를 다른 에이전트로부터 보호하는데 유용하다.

액세스 제어에 대하여, Grasshopper는 JDK1.2의 보안 메커니즘을 적용하였으며, 계정 기반(identity-based)과 그룹 기반(group-based) 액세스 제어 정책을 제공한다. 액세스 제어 정책은 몇 개의 개체들로 구성된 액세스 제어 리스트를 이용한다. Grasshopper에서 주체(subject)는 하나의 계정 또는 그룹이 될 수 있다. 몇 개의 사용승인으로 이루어진 각 주체에는 Grasshopper 에이전시의 모든 중요한 부분에 대한 액세스를 제공하는 것이다.

사용승인은 타입, 목표(target) 그리고 하나 이상의 행동(action)으로 구성되어 있다. 모든 주체와 그에 따른 사용승인들을 갖는 정책 데이터베이스는 정책 파일로 만들어진다.

시스템 액세스를 사용하고자 할 때, 액세스 제어 장치(controller)는 「액세스 결정을 만들도록 도움을 요청한다. 사실상, 시스템 액세스가 일어날 때마다 액세스 제어 장치가 호출된다. 그래서 그 액세스가 에이전트에서 만든 것인지 신뢰할 수 있는 시스템 코드에서 만든 것인지를 구분하는 것이 가능하다. 만약 에이전트로부터 액세스가 오면, 액세스 제어기는 에이전트 자체에서 에이전트의 소유자를 추출해낸다. 이러한 정보를 갖고, 유효한 몇 개의 사용승인을 추출하기 위하여 Policy 객체와 접촉한다.

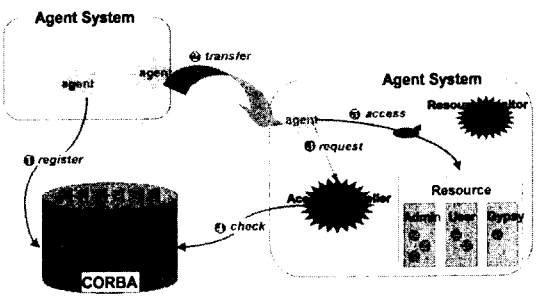
Grasshopper는 JDK1.2에서 소개했던 보호 영역(protection domain)의 개념을 사용한다. 각 에이전트는 보호 영역과 관련된 클래스 적재기에 의해서 적재된다, 그래서 다른 보안 영역과 적절하게 구분하고 있다. CLASSPATH 변수에 있는 클래스들은 시스템 클

래스로 간주된다. 그래서, "trust"로 여겨진다. 코드 베이스로부터 적재된 에이전트 클래스들은 액세스 제어에 대한 주체가 된다.

3. SMART 시스템의 자원 접근 정책

OMG에서 이동 에이전트 시스템의 상호운용성 기능에 대하여 표준 구조로 MAF 명세를 제안하였다[1]. MAF 표준은 1998년 새로운 OMG 기술로 채택하고 있고, 그 표준안은 에이전트 관리, 코드의 이동성, 명명 규칙과 같은 중요한 특성들로 이루어져 있다. 우리가 개발한 SMART 시스템은 MAF 명세를 만족하는 이동 에이전트 시스템으로서 Java 프로그래밍 언어를 이용하여 개발하였다. 이로써, 본 시스템은 상호운용성을 제공할 수 있으며, 시스템의 상이성을 해결할 수 있는 특징을 가진다.

SMART 시스템의 자원에 대한 접근은 JDK1.2의 보안 관리자(SecurityManager) 메커니즘을 기반으로 하였으며, 독자적으로 자원관리자(ResourceManager) 클래스를 구현하여 자원 접근 정책을 적용하고 있다. 자원관리자 추상클래스를 상속받아 서비스에 따른 세 가지 클래스 Admin_Service, User_Service, Gypsy_Service를 구현한다. 각 클래스는 자원등급에 맞추어 에이전트가 수행할 수 있는 모든 저 수준의 API를 정의하고 있다.



[그림 4] SMART 시스템의 자원 접근 정책

SMART 시스템에서 이동 에이전트가 가질 수 있는 자원 등급은 *Admin, User, Gypsy* 세 가지로 분류된다. 에이전트는 계정에 따른 자원 등급과 자신이 사용할 수 있는 서비스 객체의 메쏘드 정보를 가지고 에이전트 시스템에 도착한다. 에이전트 시스템에서는 이러한 정보를 원격 호스트에서 유지하고 있는 자원 관리 객체를 통해 자원 등급을 비교하여 등급이 같거나 작으면 그에 따른 자원을 할당하여 준다. 이 자원 관리 객체는 CORBA를 이용하여 구현하였으며, 저장하고 있는 정보는 계정과 그와 관련된 자원등급 그리고 사용하고자 하는 객체와 메쏘드 정보를 유지하고 있다. 이를 관리하는 *관리자(Manager)*는 계정에 따른 자원 등급을 상향시키거나 조절할 수 있는 권한을 가진다. SMART 시스템은 이동 에이전트가 사용할 수 있는 객체만을 사용하고 있는가를 감시하는 *자원모니터(ResourceMonitor)*가 있다. 이러한 기능의 구현은 에이전트가 동적 클래스 로더를 이용할 때 그 정보를 알아낼 수 있다.

다음은 우리가 개발한 SMART 시스템과 앞에서 살펴보았던 에이전트 시스템들과의 특징을 비교하여 보았다. IBM사의 Aglets 시스템은 자바 보안 기법을 적용하였지만, 코드 서명(Code Signing)을 제공하지 않았으며, Voyager 시스템은 자바 보안 기법을 더 확장한 개념을 도입하였다. Grasshopper 시스템은 자바 보안의 모든 기능을 사용할 수 있도록 지원하였으며, SMART 시스템은 자바 보안 기법을 기초로 한 독자적인 자원 보안 정책을 적용하고 있다.

<표 1> 에이전트 시스템의 자원 정책

에이전트 시스템	Java Security	자원 승인	정책 정보
Aglets	제한적으로 지원	permission	local
Voyager	확 장	permission	central
Grasshopper	모든 기능 지원	permission	local
SMART	독자적인 모델	자원 등급	central

에이전트가 자원을 할당받고자 할 때, Aglets, Voyager 그리고 Grasshopper는 각각의 자원에 대하여 승인이 필요로 한다. 하지만 SMART 시스템은 자원등급에 따른 차별적인 서비스를 할당하여 준다. 정책 정보는 Aglets과 Grasshopper는 로칼 시스템에서 유지하고, Voyager와 SMART 시스템은 그들 정보를 원격 호스트에 중앙 집중적인 형태로 관리하는 특징을 가진다.

4. 결 론

본 논문에서는 현재 다양한 응용분야에서 많이 사용되고 있는 에이전트 시스템의 보안정책을 설명하고, 그 장단점을 비교 분석하였다. 또한 이질적인 에이전트 시스템간의 상호운용성을 지원하기 위하여 개발한 SMART 시스템의 자원 접근 정책을 설명하였다. SMART 시스템은 JDK1.2 보안 메커니즘을 기반으로 하였으며, 독자적인 자원 접근 정책으로 자원등급에 따른 차별적인 서비스를 제공하는 모델을 제시하였다.

추후 연구과제로서 현재의 시스템은 계정 관리와 서비스 정보를 관리하는 자원 관리 객체를 중앙 집중적인 형태로 구성하고 있다. 이러한 시스템의 단점은 자원 관리 객체를 구성하고 있는 호스트에서 실패가 발생하면 모든 에이전트 시스템은 정상적인 동작을 수행할 수 없다. 이를 개선시키기 위하여 자원 관리 객체를 분산시키고 이들 간의 일관성을 유지하기 위하여 JACE 그룹통신 시스템을 이용할 것이다 [8].

[참고문헌]

[1] Mobile Agent System Interoperability Facilities Specification, OMG Inc, 1998. 3.

[2] Neeran M. Karnik, "Security in Mobile Agent Systems", PhD thesis, University of Minnesota, October 1998.

[3] Gunter Karjoth, Danny B. Lange, Mitsuru Oshima, "A Security Model For Aglets", IEEE Internet Computing, 1997.7.

[4] ObjectSpace Inc. "VOYAGER Security 3.2 Developer Guide", http://www.objectspace.com/products /voyager, 1999.

[5] S. Choy, T.Magedanz, "Grasshopper Technical Overview", IKV++ GmbH, February 1999.

[6] T. Taka, T. Mizuno, T. Watanabe, "A Model of Mobile Agent Services Enhanced for Resource Restrictions and Security", IEEE 0-8186-8603-0/98, 1998.

[7] 유양우, 김진호, 안건태, 문남두, 박양수, 이명준, "OMG MAF 명세를 지원하는 이동 에이전트 시스템의 개발", 한국정보과학회 99가을 학술발표논문집(III) 1999.

[8] 문남두, 안건태, 유양우, 이명준, "JACE: 인터넷 환경을 지원하는 신뢰성 있는 그룹통신 시스템", 한국정보처리학회 논문지 특집호, 1999. 12