이동 에이전트를 이용한 보안 관리 시스템 모델

김현배
부산교육대학교 컴퓨터교육과

요 약

이동 에전트를 응용한 프로그래밍은 최근 급격히 확산되고 있는 경향으로서, 이는 코드의 이동성을 기초로 한 프로그래밍 패다임이다. 네트워크 환경에서 컴퓨터의 사용이 증가함에 따라 컴퓨터 네트워크의 보안 관리의 중요성이 부각되고 있다. 본 연구는 이동 에이전트를 이용한 보안 관리 시스템의 기본 모델을 설계하였다.

Security Management System Model using Mobile Agent

Hyunbae Kim
Pusan National University of Education, Deptartment of Computer Education

ABSTRACT

The goal of this study is to survey a mobile agent technology and apply it to security management system. A tool for implementing mobile agent is introduced and a prototype model for sedcrity management system is proposed. This study suggest the possiblity of security management system using mobile agent.

1. 서 론

이동 에이전트를 응용한 프로그래밍은 최근 급격히 확산되고 있는 경향으로서, 이는 코드의 이동성을 기초로 한 프로그래밍 패러다임이다. 이러한 경향은 웹 상에서의 단방향이동을 기초로 한 자바와 같은 언어의 성공으로 인해 향후의 일반화 가능성을 예견할 수 있다. 이동 에이전트는 플랫폼 독립적인 분산 환경에서의 프로그래밍을 기존의 RPC, 메시지전달, 자바와 같은 언어의 코드 이동을 더욱 일반화한 프로그래밍 환경을 제공할 것이다.

본 연구에서는 이동 에이전트를 전산망기반의 보안 관리에 적용하기 위한 기본 모델을 연구하였다. 이는 기존의 고정적인 C/S 개념의 보안 관리 방법을 이동 에이전트에 적용한 모델을 제시한다. 또한 IBM이 지원하는 이동 에이전트의 개발도구인 IBM Aglets Workbench의 기능을 살펴보고 이를 이용한 구현 방법을 모색한다.

2. IBM Aglets Workbench

Aglets Workbench[3][4][5][6][7]는 IBM에서 개발중인 자바 기반의 이동 에이전트 개발환경이다[6]. Aglets Workbench는 Java 프로그래밍 언어를 기반으로 하여, 인터넷상을 이동하는 플랫폼 독립적인 에이전트의 작성을 가능하게 한다. Aglets Workbench의 시각적 도구는 이동 에이전트의 개발을 손쉽게 해주며, Aglets Workbench의 풍부한 소프트웨어 콤포넌트들은 에이전트 및 플레이스의 개발, 데이터베이스의 제어나 탐색, 여행, 표준화된 에이전트간 통신을 가능하게 한다.

Aglet은 Aglets Workbench로 작성된 이동 에이전트를 가리키는 용어이다. Aglet은 자바의 애플릿, 혹은 웹브라우저에서 웹서버로 업로드되는 서블릿과 같은 용어를 모방하여 만든 용어로 Agile applet이라는 의미를 가지고 있다.

이동 에이전트의 프로그래밍을 위한 별도의 프로그래밍언어를 개발하는 대신, IBM Aglets Workbench는 J-AAPI(Java Aglet API)를 개발하여 Aglet 프로그래밍을 지원한다. 이들 API는 자바언어에서 사용 가능한 클래스의 형태로 제공되며, 프로그래머는 자바언어를 사용하여 이동 에이전트 및 이동 에이전트의 플레이스를 개발할 수 있다. 이렇게 하여 자바언어에 익숙한 프로그래머들이 손쉽게 이동 에이전트를 구현할 수 있도록 한다[5].

이동 에이전트 실행 환경으로 Tahiti, Fiji를 제공하고 있다. 이들은 에이전트의 가상 기계 역할을 함과 동시에 에이전트의 관리를 위한 도구를 지원한다. Tahiti의 경우 시각적인 에이전트 관리가 가능하도록 하고 있으며, Fiji는 웹서버와 웹브라우저를 이용하여 Aglet을 생성, 실행, 관리할 수 있도록 하고 있다.

Aglet의 ATP(Agent Transfer Protocol)은 인터넷상에서 플랫폼에 독립적으로 에이전트를 이동하고, 에이전트간의 ATP 메시지를 전송할 수 있도록 통신 규약을 정의하고 있다.

2.1 Aglet의 구성

Aglets Workbench는 이동 자바 에이전트를 위한 Aglet Framework, Agent 전송 프로토콜(ATP: Agent Transfer Protocol), 시각적 에이전트 관리자인 Tahiti, 웹을 사용한 에이전트 관리자인 Fiji, JDBC 등의 패키지[6]를 포함한다.

2.1.1 Aglet Framework

Aglets Workbench의 핵심은 Aglet Framework이다. Framework은 자바 프로그래밍언어를 기반으로 하고 있으며 Aglet을 작성하기 위한 기본 클래스를 제공한다. Aglet class를 사용하는 것은 사용자 정의 에이전트가 이동 에이전트 구현을 위해 부족한 속성과 기능을 상속받기 위한 평범한 방법이다. Aglet class는 다음과 같은 것들을 포함한다.

- 에이전트를 위한 유일한 명명법.
- 다수의 목적지를 갖는 복잡한 여행 패턴을 구별하기 위한 여행 계획표와 자동적인 failure handling.
- 다수의 에이전트가 비동기 적으로 협력하고 자료를 공유하기 위해 사용하는 white board 기법.
- 에이전트 메시지-전달 기법.
- 네트워크를 돌아다니는 에이전트의 자바 바이트코드와 상태 정보를 허용하는 네트워크 에이전트클래스 로더.
- 플랫폼 독립적인 에이전트 실행 환경.

2.1.2 에이전트 전송 프로토콜

에이전트 전송 프로토콜(ATP)은 네트워크 상에서 에이전트를 전송하기 위해 사용된다. 이 프로토콜은 분산 에이전트 기반 정보 시스템을 위한 응용레벨 표준 프로토콜이다.

ATP는 에이전트 이동성을 제어하기 위해 정형화된 방식을 제공한다. ATP는 전체가 자바로 작성되었으며 고도의 이동 가능한 클래스들은 ATP 데몬을 위한 표준 API, ATP 사이트 연결, ATP의 요청과 응답 생성을 제공한다.

2.1.3 Tahiti

Tahiti는 시각적인 에이전트 관리자이며 Aglet 모니터와 관리를 위해 특별한 그래픽 사용자 인터페이스를 사용한다. 드래그/드롭 인터페이스를 통하여 두 개의 Aglet 사이의 통신을 시키거나 특정 사이트로 Aglet을 파견할 수도 있다.

2.1.4 Fiji

WWW은 이동 에이전트를 위한 매우 중요한 기반이다. Aglets Workbench에는 Fiji라고 불리는 Agent Web Launcher를 포함하고 있다. Fiji는 Aglet Framework를 기반으로 한 자바 애플릿이다. Fiji 애플릿은 매개변수로서 에이전트 URL을 가지며 자바 애플릿처럼 HTML을 사용하여 쉽게 Web page에 포함시킬 수 있다. 그리고, 다른 애플릿들과 마찬가지로 소프트웨어를 클라이언트의 브라우저로 직접 다운로드 할 수 있다. 굳이 에이전트 사용을 위해 소프트웨어를 클라이언트에 다운로드하고 설치할 필요가 없다.

2.2 Aglet 보안

보안은 이동 에이전트 사용자에게 가장 중요하다. 알려지지 않은 에이전트는 잠재적으로 모든 종류의 문제점을 발생시킬 수 있다. Aglet Framework은 계층화된 보안 모델을 제공한다. 첫 번째 보안 계층은 자바 언어 자체이다. 에이전트에 내포된 코드 조각들은 일련의 검사를 받게 된다. 정확한 코드 형식이라는 것을 보장하는 검사를 시작으로 자바 바이트코드 검증기(bytecode verifier)에 의한 일관성 확인 검사를 마지막으로 받게된다.

그 다음 계층은 보안 관리자이다. 보안 관리자는 Aglet Framework의 사용자가 자신의 보안 기법을 구현하도록 한다. Tahiti는 호스트 컴퓨터 시스템과 그 소유자를 위해 신뢰할 수 있을 정도의 보안을 제공하는 구성 가능한 보안 관리자를 구현한다. 허가되지 않은 에이전트의 접근 시도는 보안 위반으로 보고되고 그러한 에이전트의 접근은 허가되지 않는다.

마지막 계층은 자바 보안 API 이다. 이 API는 에이전트 개발자가 에이전트 내에 보안 기능을 포함하기 쉽게 해주는 계층이다.

2.3 Aglet API

Aglet API[4]는 Aglet Framework의 일부로서 J-AAPI로 명명된다. J-AAPI는 응용 프로그래머가 Aglet을 이용하여 독자적인 Aglet 시스템을 구축할 수 있도록 지원한다.

2.3.1 Aglet Context

모든 Aglet은 Aglet Context에서 실행된다. Aglet Context는 일종의 가상기계로서 플랫폼에 관계없이 Aglet의 실행을 가능하게 해 준다.

Aglet은 Aglet Context 내에서 만들어진다. Aglet이 기계들을 돌아다닌다는 것은 실제로 Aglet Context들이 돌아다닌다는 말이다.

2.3.2 Aglet의 생명주기

Aglet의 생명주기 Aglet의 생성과 복제에 관련한 생성, 삭제에 관련한 소멸, 파견과 회수에 관련한 이동, 비활성화와 활성화에 관련한 저장으로 분류되어진다.

(1) Aglet의 생성

Aglet으로 어떤 일을 하기 위한 첫 번째 단계가 Aglet을 생성하는 것이다. Aglet 생성에는 두 가지 방법이 있다. 하나는 Aglet 클래스로부터 실제로 만드는 방법과 만들어져 있는 Aglet으로부터 복제하는 방법이다.

<그림 1>Aglet 생성

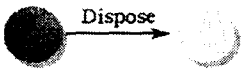
새로운 Aglet을 생성하기 위해서는 먼저 Aglet Context에 접근해야 한다. 이런 접근은 getAgletContext 메소드의 호출에 의해 이루어진다.

예를 들어, 현재 어떤 context에서 동작하는 Aglet이 같은 기계 상에 위치하고 있는 SomeAglet 클래스의 인스턴스를 생성하려고 한다면 먼저 현재의 Aglet Context를 알아내고, 알아낸 후 createAglet를 사용해야 한다. 다음의 프로그램 코드가 이와 같은 일을 해준다.

새롭게 Aglet을 생성하는 대신 현재 위치한 context상에서 존재하는 Aglet을 복제하는 방법이 있다. Aglet의 clone method를 사용하는 것이다. 이 메소드는 복제된 Aglet을 직접적으로 반환하지 않고 대신 Aglet의 프록시를 준다. 프록시는 Aglet을 가리키는 이름과 같다.

(2) Aglet 소멸

Aglet은 Aglet Context에 있는 동안 다양한 자원들을 차지한다. 그러므로, 좋은 Aglet은 주어진 임무의 수행을 끝마친 후에 적절히 재배치되어야 한다. 항상 대부분의 Aglet이 손님의 자격으로 원격 서버에서 수행되므로 호스트 자원을 최소한도로 소비하도록 해야 한다. Aglet이 소멸되면 Aglet Context는 해당 Aglet에 속한 모든 쓰레드를 재 사용하게 될 것이다.



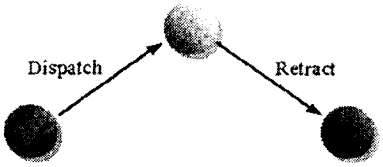
<그림 2> Aglet 소멸

(3) Aglet의 이동

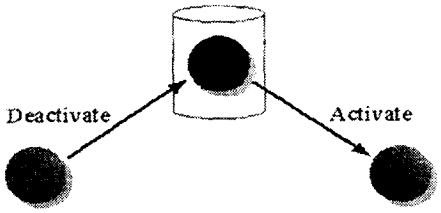
Aglet이 움직이는데는 두 가지 형태가 있다. 하나는 원격지로 파견하는 것이다. 좀 더 명확하게 말해서 Aglet을 파견한다는 것은 Aglet을 실행될 다른 위치로 보낼 것을 의뢰하는 것이다. Aglet이 움직이는 다른 형태는 원격 호스트로부터 회수하는 것이다. 즉, Aglet의 회수로 인해 원격지로부터 되돌아오는 것을 의뢰하는 것이다.

(4) Aglet의 저장

Aglet은 하드디스크와 같은 저장장치에 일시적으로 저장될 수 있다. 비활성화된 Aglet이란 실행이 중지되어 영구 저장 장치에 저장되어 있는 Aglet을 말한다. Aglet이 다시 Aglet Context로 되돌아오게 되면 다시 활성화되었다고 한다.



<그림 3> Aglet의 이동



<그림 4> Aglet의 저장

2.3.3 Aglet 메시지 교환

Aglet은 Aglet간의 메시지 교환 기법을 제공한다. Aglet의 동기적 메시지와 비동기적 메시지를 동시에 제공한다. 또, 1대1 통신과 방송의 기법을 제공한다.

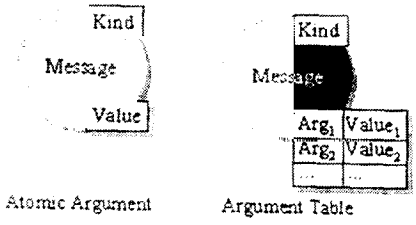
(1) Aglet 메시지의 구조

Aglet의 메시지는 Kind와 값으로 구성되어 있다. Aglet 메시지는 한 개의 값을 가지는 Atomic Argument와 테이블을 값으로 가지는 매개변수 테이블이 있다.

(2) 1 대 1 메시지 교환

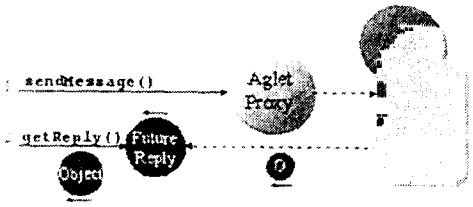
Aglet의 메시지는 Aglet 프록시를 사용하여 메시지가 전달될 상대 Aglet을 식별한다. Aglet 프록시는 Aglet이 생성될 때 값을 얻을 수 있다.

메시지를 수신하는 측은 messageHandler() 메소



<그림 5> Aglet 메시지의 구조

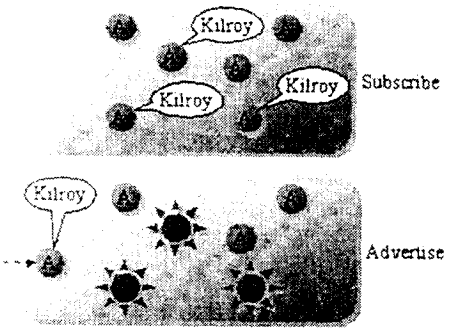
드를 재정의 하여 수신된 메시지를 처리한다. 동기 및 비동기 메시지는 각각 sendMessage()와 sendAsyncMessage() 메소드에 의해 지원된다.



<그림 6> 1 대 1 메시지 교환

(3) 방송

Aglet은 다수의 Aglet에게 동시에 메시지를 보낼 수 있다. 방송 메시지를 수신하려는 Aglet들은 원하는 메시지의 Kind에 등록해야 한다.



<그림 7> 방송

방송하는 Aglet은 multicastMessage() 메소드를 이용하여 메시지를 방송하며, 수신하는 측은 subscribeMessage(String kind)를 사용하여 원하는 메시지를 등록한다. 위의 경우 "Kilroy"라는 Kind를 가진 메시지가 방송되는 모양이다.

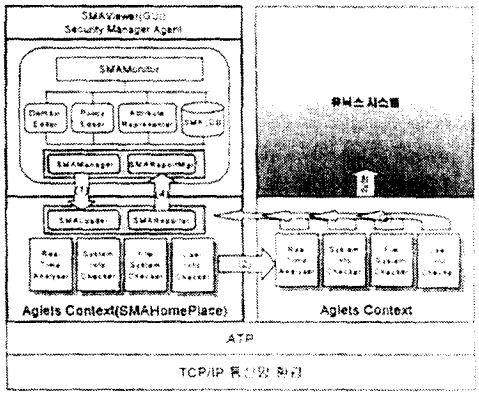
2.4 표준화

IBM에서는 Aglet를 표준화시키기 위한 활발한 노력을 하고 있다. 구체적으로 ATP와 API framework를 OMG의 이동 에이전트 위원회에 제안해놓은 상태이다. 또한 Aglet을 여러 곳에서 사용하는 범용 소프트웨어로 만들기 위해 무료로 배포하고 있다.

3. 시스템 설계

3.1 시스템 모델

<그림 8>은 전산망보안 관리를 위한 이동 에이전트 설계를 위한 모델을 보여주고 있다. 기본적으로 TCP/IP 인터넷 환경을 이용하는 IBM Aglet 가상기계 환경을 이용하여 동작하는 이동 에이전트 응용시스템을 구현한다. SMA(Security Management Agent)의 보안 관리를 위해 파견되는 이동 에이전트 역시 SMA로 부르기로 한다.



<그림 8> 보안 관리 이동 에이전트 설계 모델

시스템은 SMAViewer(GUI 콤포넌트), SMAMonitor(보안 모니터 관리기/보안 정보 분석기), SMAHomePlace(보안 관리 홈 플레이스)의 3부

분으로 구성된다.

SMA 시스템은 보안 관리의 마스터 시스템에만 설치된다. 보안 관리 목표 시스템들에는 단순히 Aglet 실행시간만 존재하며, 별도의 고정 에이전트가 존재하지 않는다.

3.2 SMA 동작

SMA 시스템의 동작 과정은 다음과 같다. SMAViewer를 통하여 SMA의 보안정책들을 설정하며, SMAMonitor의 모니터링 표현방식(텍스트, 테이블, 차트)을 결정할 수 있다.

설정된 내용은 AgletsManager가 AgletsHomePlace에 존재하는 AgletsLoader에게 에이전트의 생성을 의뢰하는 과정에서 전달한다.

AgletsHomePlace에 존재하는 AgletsLoader는 요구에 따라 해당 SMA(Real Time Analyser, System Infor Checker, File System Checker, User Info Checker)를 생성하고 대상호스트로 파견한다. 파견된 에이전트는 호스트에 도착하여 실시간 점검, 시스템 정보 점검, 계정점검, 파일시스템 점검 등의 기능을 수행한다.

각 SMA는 AgletsHomePlace에 존재하는 SMAReporter에게 보안정보의 점검 내용을 보고한다. SMAReporter는 보고 내용을 Sec_DB에 저장하고 결과를 SMAReportMgr에게 알린다. SMAReportMgr는 보고 받은 결과를 SMAMonitor에 알린다. SMA는 대상호스트가 끝날 때까지 연속 이동한다.

또한 SMAMonitor 는 사용자가 설정한 표현방법에 따라 SMA_DB에 저장된 각 호스트의 보안상태 정보를 가공하여 관리자에게 디스플레이 하게 된다.

SMAViewer는 관리자가 SMA를 제어하고 결과를 보고 받을 수 있는 사용자 인터페이스 기능만 제공하며 파견하고자 하는 에이전트의 영역설정, 보안 정책, 보안 속성 값들의 표현에 대한 설정은 실제로 SMAMonitor에서 수행된다.

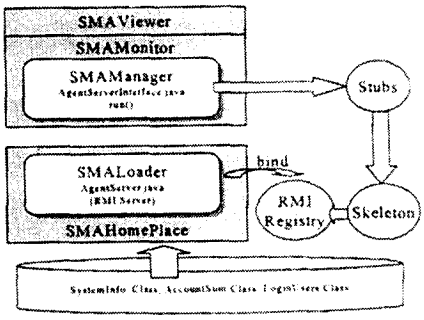
SMAViewer와 SMAMonitor는 자바로 작성된 응용 프로그램 계층이며, SMAHomePlace는 Aglet의 서브클래스로 작성되며, Aglets Context에서 실행된다. SMA의 생성, 이동, 관리는 Aglets Context에서, 네트워크 이동시의 프로토콜은 ATP에서 담당한다. SMA는 한 호스트에서 수행을 마치면 다음 호스트로 이동하고 자신의 스케줄에 따라 이동하며 임무를 끝마친 SMA는 삭제된다.

3.3 모듈간의 인터페이스

3.3.1 인터페이스 구조

SMAViewer와 SMAMonitor는 Java로 작성된 Java Application이다. 반면에 SMAHomePlace는 Java로 작성되지만 Aglets API를 사용한 Java Aglet이다. Java Applet이 appletviewer나 웹브라우저에서 실행되는 것과 같이 Java Aglet은 Aglet Context를 제공하는 Tahiti와 같은 Aglet Runtime을 필요로 한다. 따라서 본 보안 관리시스템은 Java Application과 Java Aglet간에 편재한 모듈간의 인터페이스를 정의할 필요가 있다.

별도의 프로세스가 협조하여 작업하기 위해서는 IPC, TCP/IP 소켓, Java RMI와 같은 방법을 사용해야한다. 이를 도식화하면 <그림 9>와 같다.



<그림9> 모듈간의 인터페이스

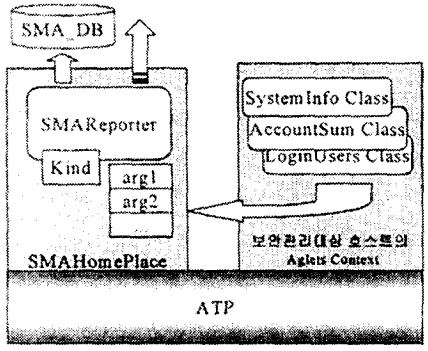
3.3.2 SMAHomePlace와 대상 호스트와의 인터페이스

이동 에이전트는 동기적 호출을 사용하는 RPC와는 달리 비동기적 호출을 사용한다. 다시 말하면 Aglet Context에 로드된 SMA는 독자적으로 실행되

며, 결과를 리턴하지 않는다. SMA를 로드한 SMALoader역시 SMA의 실행을 기다리지 않는다.

보안사항의 결과를 리턴하기 위해서 SMA에서는 Aglet의 메시지를 사용해야 한다. Aglet은 동기적 및 비동기적 메시지 전달을 지원하며, 다수의 Aglet에 대한 방송 기법을 지원하기도 한다.

보안 관리 대상 호스트에 이동한 각 에이전트는 sendMessage()를 사용하여 검사결과를 송신하며, SMAHomePlace의 SMAReport는 Message Class의 handleMessage() 메소드를 override하여 이를 처리한다. SMAReport는 전달받은 결과를 SMA_DB에 저장하고 결과를 SMAMonitor에 알린다.



<그림 10> SMAHomePlace와 대상 호스트의 인터페이스

4. 결 론

이동 에이전트를 이용한 보안 관리 시스템의 기본 모델을 설계하였다. 또한 IBM Aglets Workbench의 기능에 대하여 살펴보았다. 본 연구를 통하여 이동 에이전트를 이용한 보안 관리를 응용의 하나로서 제시하고자 한 것에 본 연구의 의의가 있다.

이동 에이전트를 이용한 프로그래밍 패러다임이 일반화되는 시점에서 본 연구에서 제안한 보안 관리 시스템이 이동 에이전트의 고유한 장점을 그대로 이어 받으며 네트워크 환경의 컴퓨터들의 보안 관리에 이용될 수 있을 것으로 기대된다.

고정된 보안 관리자가 네트워크상의 각 컴퓨터에 항상 존재해야 하는 기존의 방식과 달리 보안 감시를 위한 컴퓨터에만 시스템을 설치하고 기타의 컴퓨터에는 일반적인 이동 에이전트의 가상기계만이 설치된다면 소프트웨어 분배의 효율성 등에 많은 이익을 기대할 수 있으리라고 본다.

참고 문헌

[1] 김현배, "보안서비스를 제공하기 위한 이동 에이전트 시스템의 설계에 관한 연구", 한국정보교육학회, 2권1호, 1998.8.

[2] 박지은, 김상욱, "이동 에이전트를 지원하는 프로그래밍 언어", 정보처리학회, 4권 5호, 1997.9.

[3] Mitsuru Oshima, Guenter Karjoth, "Aglets Specification(1.0)", IBM Research Division, http://www.trl.ibm.co.jp/aglets/, 1997.5.

[4] Danny B. Lange, Yariv Aridor, "Agent Transfer Protocol -- ATP/0.1", IBM Research Division, http://www.trl.ibm.co.jp/aglets/, 1997.3.

[5] Danny B. Lange, "Java Aglet Application programming Interface(J-AAPI) White Paper - Draft 2", IBM Research Division, http://www.trl.ibm.co.jp/aglets/, 1997.2.

[6] Danny B. Lange, Daniel T. Chang, "IBM Aglets Workbench, A White Paper, Draft", IBM Research Division, http://www.trl.ibm.co.jp/aglets/, 1996.9.

[7] Danny B. Lange and Mitsuru Oshima, "Programming Mobile Agents in Java", IBM Research, http://www.trl.ibm.co.jp/aglets/.