OMG MAF 명세를 지원하는 이동 에이전트시스템의 개발

유양우° · 김진홍 · 안건태 · 문남두 · 박양수 · 이명준
울산대학교 컴퓨터 · 정보통신공학부

A Mobile Agent System supporting OMG MAF Specification

Yang-Woo Yu° · Jin-Hong Kim · Geon-Tae Ahn · Nam-Doo Moon · Yang-Soo Park · Myung-Joon Lee
School of Computer Engineering · Information Technology, Univ. of Ulsan.

요 약

인터넷 환경에서 분산 애플리케이션을 개발하는데 있어서 다양한 기술과 방법이 사용되고 있다. 그 중 이동 에이전트를 이용하는 기술은 서버의 인터페이스를 바꾸지 않고 클라이언트의 다양한 요구를 융통성 있게 서비스할 수 있고, 네트웍 트래픽(traffic)을 줄이는 패러다임 특성 때문에 현재 널리 각광받고 있다. 하지만, 대부분의 이동 에이전트시스템들은 그 구조와 구현이 매우 상이하여, 이동 에이전트들이 이종의 시스템에서 실행이 되지 않고 있다. 이러한 문제를 해결하기 위하여 OMG(Object Management Group, Inc)는 이질적인 에이전트시스템간의 상호운용성(interoperability)과 이식성을 증진시킬 목적으로 MAF(Mobile Agent Facilities) 명세를 제안하였다. 본 논문에서는 OMG MAF 명세를 만족하는 이동 에이전트시스템을 Java 프로그래밍 언어를 이용하여 개발하였다.

1. 서 론

네트웍 중심의 프로그래밍이나 어플리케이션들에 대한 관심은 인터넷 사용자수의 증가와 WWW의 폭발적인 인기 때문에 최근 몇 년 사이 급진전하고 있다. 이에 부흥하여 이 같은 어플리케이션을 생성하기 위한 새로운 기술, 언어 ,그리고 패러다임들이 쏟아져 나오고 있다[2].

이동 에이전트(Mobile Agent): 에이전트라고 하는 것은 사용자를 대신해서 자발적으로 행동하는 컴퓨터 프로그램을 의미한다. 따라서 이동 에이전트라고 하면 컴퓨터 네트웍에서 어떤 계산을 수행하기 위하여 노드와 노드사이를 자율적으로 이동할 수 있는 사용자 대행 프로그램을 말한다. 그들의 일은 에이전트 프로그래밍에 의해서 결정되고, 그 응용은 인터넷 전자 상거래에서 실시간 디바이스 컨트롤, 과학적인 계산을 요하는 분산처리에 이르기까지 다양하다[1].

이동 에이전트시스템은 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 홈사이트로 돌아오게 된다[8].

이동 에이전트는 새로운 연구를 제공하는 비교적 신기술이다. 이러한 기술과 연구는 새롭기 때문에, 이동 에이전트 시스템들은 그 구조와 구현에서 너무 상이하다. 이런 시스템들간의 차이는 에이전트 기술의 빠른 확산과 상호운용성(interoperability)[1]을 가로막고, 에이전트시스템의 적용을 어렵게 하고있다. 상호운용성과 시스템의 상이성을 개선시키기 위하여 이동 에이전트 기술의 몇 가지 양상은 표준화되어야만 한다. OMG(Object Management Group, Inc)에서는 이동 에이전트 시스템의 상호운용성 기능에 대하여 표준 구조로 MAF(Mobile Agent Facilities) 명세를 제안하였다[1]. 이로서, 에이전트 시스템들 간의 상호운용성을 제공할 수 있으며, 시스템의 상이성을 해결할 수 있다. 본 논문에서는 MAF 명세를 만족하는 이동 에이전트시스템인 SMART 시스템을 Java 프로그래밍 언어를 이용하여 개발하였다.

본 논문의 구성은 다음과 같다. 2장에서는 이동 에이전트시스템간의 상호운용성을 제공하기 위하여 OMG에서 제안한 MAF 명세의 구현에 대하여 소개하고, 각 메쏘드들이 동작하는 방법에 대해 살펴본다. 3장에서는 SMART 시스템의 전반적인 구조와 각 모듈의 역할에 대해 설명한다. 끝으로 4장에서 결론 및 향후 연구방향에 대해 살펴본다.

2. OMG MAF 명세

이동 에이전트 기술에서 가장 중요한 목적은 다양한 에이전트시스템 사이의 상호운용성이다. 에이전트 전송과 클래스 전송, 그리고 에이전트 관리에서 표준화가 된다면 상호운용성은 더욱더 쉽게 이루어질 수 있다. OMG에서 제안한 MAF 명세는 이종의 이동 에이전트시스템에 대하여 상호운용 가능한 인터페이스를 정의하였다. MAF 모듈의 구성은 MAFAgentSystem과 MAFFinder 두 개의 클래스로 이루어져 있다. MAFAgentSystem 클래스는 에이전트를 생성하고, 목적지 시스템으로 전송하는 작업을 수행한다. 또한 에이전트의 실행을 중지하고, 종료시키는 에이전트 다양한 오퍼레이션을

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

정의한다. 명명(Naming) 서비스를 지원하는 MAFFinder 클래스는 에이전트와 플레이스(Place), 그리고 에이전트시스템을 등록시키고, 등록 해제 그리고 검색하는 오퍼레이션을 정의한다.

2.1 MAFAgentSystem 클래스

에이전트 관리 작업을 지원하는 메쏘드와 객체를 정의하는 MAFAgentSystem 클래스가 있다[1]. 주로 에이전트시스템 이름을 구하고, 에이전트를 받아들이는 작업을 한다. 이러한 메쏘드와 객체들은 에이전트 전송에 관한 기본적인 몇 가지 오퍼레이션을 제공한다.

· *create_agent()* : 에이전트 시스템은 원격 클라이언트의 요청에 의해 에이전트를 생성하기 위하여 create_agent() 오퍼레이션을 수행한다. 리턴 값은 생성된 에이전트의 실제 이름을 반환한다. 에이전트시스템이 아닌 클라이언트 또한 이 메쏘드를 이용하여 명시한 에이전트시스템 내에 특정 에이전트를 생성시킬 수 있다.

· *fetch_class()* : 이 메쏘드는 에이전트시스템에서 에이전트가 작업을 수행할 때, 클라이언트의 클래스 또는 코드를 불러올 수 있으며, 명시한 코드 베이스와 클라이언트로부터 클래스를 검색하는데 주로 사용한다. 리턴 값은 하나 이상의 클래스 정의를 반환한다

· *get_MAFFinder()* : 에이전트와 플레이스 그리고 에이전트 시스템들을 검색하기 위하여 MAFFinder에 대한 레퍼런스를 반환한다. MAFFinder에 대한 레퍼런스를 얻자마자, 에이전트와 플레이스 그리고 에이전트시스템을 찾기 위하여 MAFFinder 클래스의 메쏘드를 사용할 수 있다.

· *receive_agent()* : 에이전트시스템은 에이전트를 받아들이고 인스턴스화 시키기 위하여 receive_agent()를 사용한다. 이 메쏘드에 대한 알고리듬은 먼저, 에이전트를 인스턴스화 시키는데 필요한 클래스가 입력 파라미터에 있는지 아니면 에이전트 시스템 플랫폼에 캐쉬되어 있는지를 검사한다. 에이전트를 실행시키는데 필요한 클래스가 요구되면 fetch_class()를 호출한다. 마지막으로, 명시한 플레이스에서 에이전트를 직렬화(deserialize) 시키고 인스턴스화 시킨다.

2.2 MAFFinder 클래스

에이전트와 플레이스, 그리고 에이전트시스템의 동적인 이름과 위치(Location)정보를 유지하기 위한 메쏘드를 MAFFinder 클래스에서 제공한다[1].

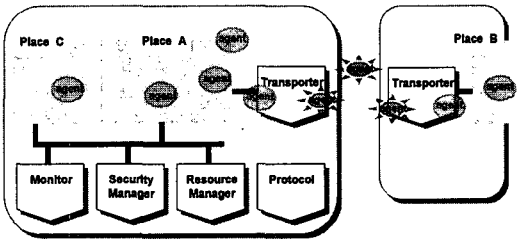
· *lookup_agent()* : 파라미터에 명시한 에이전트의 위치를 반환한다. 이 메쏘드는 이름으로 특정 에이전트를 검색할 수 있다. 또한 특정 에이전트 프로화일(Profile)과 일치하는 에이전트들을 검색할 수 있다. 이 메쏘드를 사용하여 다른 에이전트 또는 통신하고자 하는 에이전트를 찾을 수 있다.

· *register_agent()* : MAFFinder에 등록된 에이전트의 리스트에 에이전트를 추가시키는 메쏘드이다. 이동 에이전트는 여러 곳을 돌아다니기 때문에, 이 오퍼레이션은 에이전트에 의해서 매우 빈번하게 호출될 수 있다. MAFFinder 내의 이미 존재하는 에이전트 이름으로 호출된다면, 이 오퍼레이션은 관련 정보를 가장 최근에 호출된 정보와 대치시킨다.

· *unregister_agent()* : MAFFinder에 등록된 에이전트의 리스트에 명시된 에이전트를 제거한다. 이 메쏘드는 에이전트 시스템 내의 에이전트의 리스트를 유지하는데 사용할 수 있다.

3. SMART 시스템의 구조

SMART 시스템은 [그림1]에서 보는 바와 같이 크게 5개의 패키지로 구성되어있다. 에이전트를 실행시키고 종료시키는 에이전트 서버와 에이전트를 목적지 에이전트시스템으로 전송하는 트랜스포터, 에이전트의 라이프사이클을 감시하는 모니터, 그리고 에이전트에게 목적지 에이전트시스템의 자원을 할당하고 회수하는 자원 관리자, 마지막으로 에이전트시스템의 보안을 유지하는 보안 관리자로 구성되어있다. 3장에서는 이들의 역할에 대해 간략하게 살펴본다.



[그림 1] SMART 시스템의 전체적인 구조

3.1 에이전트

에이전트는 사용자를 대신하여 자발적으로 행동하는 컴퓨터 프로그램이다. 각각의 에이전트는 수행을 위하여 자신의 스레드를 가지도록 설계되었다. 에이전트의 구현은 인터페이스로 정의되어 있으며 에이전트의 기본적인 속성을 가진 Teplete 추상 클래스를 상속받아 실제 에이전트를 구현한다.

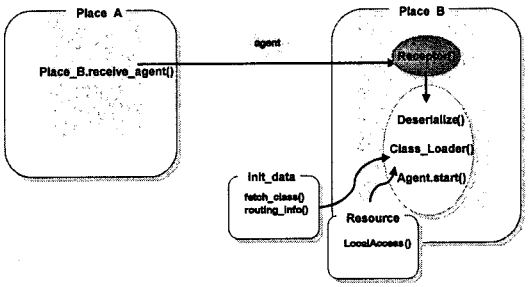
3.2 에이전트 서버 : 플레이스(Place)

에이전트 서버의 주된 임무는 에이전트를 받아들이고 실행시키는 것이다. 본 논문에서는 에이전트가 수행되는 실행 환경을 플레이스라고 호칭한다. 이 플레이스는 하나의 시스템에 실행될 수 있으며, 또한 한 시스템에 여러 개의 플레이스가 존재할 수 있다[4]. SMART 시스템에서는 클라이언트가 목적지의 플레이스를 동적으로 생성할 수 있는 기능을 제공한다.

3.3 전송 관리자(Transporter)

SMART 이동 에이전트는 미리 설계된 경로를 따라 이동하거나, 그들 자신이 모은 정보에 의해 직접 동적으로 경로를 설정하여 돌아다닌다. 이동 시, 목적지 에이전트시스템의 위치가 설정되면, 이동 에이전트는 목적지 에이전트시스템으로 이동하기 위하여 현재의 에이전트시스템에게 요청한다. 이 때, 보내는 에이전트시스템은 다음과 같은 행동이 일어난다. 현재의 에이전트 수행을 중단시키고, 전송할 에이전트의 상태를 Transport 클래스의 wrap_up() 메쏘드를 이용하여 저장한다. 그리고 에이전트 클래스와 상태를 네트웍을 통해 전송할 수 있도록 바이트 스트림으로 변환시키는 serialize() 메쏘드를 호출한다. 직렬화된 에이전트는 주어진 프로토콜로 전송되기 위하여 암호화된다. 마지막으로 요청자의 인증 과정을 거친 후 해당 에이전트를 목적지 에이전트시스템으로 전송한다. 목적지 에이전트시스템에서는 에이전트를 기다리는 Receptor가 에이전트를 받아들인다. 도착한 에이전트는 인증 과정을 거친 후에 에이전트 클래스와 상태를 역 직렬화(deserialize) 시킨다. 그리고 에이전트를 인스턴스화 시키고 에이전트 상태는 저장된다. 그런 후에 에이전트를 실행시킨다.

SMART 시스템에서 에이전트를 보내는 정책은 세 가지 방법을 지원한다[1]. 첫째로는, 에이전트를 전송 시 필요한 클래스 모두 전송하는 방법. 둘째로는, 에이전트 클래스만을 전송하고, 다른 클래스들은 필요할 때마다 전송한다. 마지막으로, 에이전트 생성 시 필요한 모든 클래스의 이름 목록을 전송한다. 에이전트 실행 도중에 필요한 클래스들은 먼저 로컬 캐시에 저장되어 있는지 검사하고, 없으면 해당 호스트와 근원지 에이전트시스템을 검사하는 방식이다.



[그림 2] SMART 시스템에서 에이전트 전송

3.4 모니터(Monitor)

이동 에이전트들은 주어진 연산을 수행하기 위하여 한 노드에서 노드로 광범위한 영역을 돌아다닌다. 이러한 에이전트들은 특정 노드에서 수행 도중 시스템 또는 다른 요인으로 인해 그들의 수행이 중단될 수 있다. 에이전트시스템은 기본적으로 에이전트의 라이프사이클 정보를 유지해야 한다[3]. SMART 시스템은 모니터 클래스 내의 alive_agent() 메쏘드가 이러한 임무를 담당하고 있으며, 또한 하나의 에이전트시스템에서 생성된 에이전트들이 그들의 임무를 잘 수행하고 있는지 여부를 검사하는 기능을 제공한다. list_all_agent()는 현재 에이전트시스템 내에 존재하는 에이전트의 이름을 반환한다. 또 다른 기능으로, 원거리에 있는 에이전트시스템에게 특정 플레이스에 어떤 에이전트를 만들 수 있도록 요청할 수 있다.

3.5 자원 관리자(Resource Manager)

이동 에이전트는 클라이언트의 코드를 목적지 에이전트 서버에서 실행하는 패러다임이다. 도착한 에이전트는 목적지 시스템에서 인증과정을 거친 후 자신에게 할당된 양의 자원을 사용할 수 있다. SMART 시스템은 이동 에이전트가 에이전트시스템 자원에 대하여 접근 할 때, 자바 보안 관리자를 사용하여 안정성을 제공하였다. 그리고 자원에 대한 접근 안정성을 위하여 Resource와 AccessProtocol 인터페이스 클래스를 정의하였다[5].

Resource 인터페이스 클래스는 시스템이 지원하는 모든 자원에 접근하기 위한 일반적인 기능을 제공한다. 이 클래스에서 제공하는 getResource() 메쏘드는 자원에 대한 참조를 에이전트에게 전달한다. 그 결과 자원에 참조를 가진 에이전트는 그 자원을 접근 할 수 있다.

AccessProtocol 인터페이스 클래스는 자원에 대한 접근 허용을 확인하는 checkAccess() 메쏘드, 이미 에이전트에게 접근이 허용된 자원의 집합을 나타내는 배열 successMethods를 제공하며 또한, 자원 접근을 동적으로 제어하기 위하여 enable() 와 disable() 메쏘드를 제공한다.

3.6 보안 관리자(Security Manager)

전송될 모든 에이전트는 악의의 에이전트들로부터 피해가 없도록 자신을 철저히 보호해야한다. SMART 시스템에서는 에이전트를 전송할 때, 에이전트 클래스와 상태는 모두 암호화시켜서 전송한다[7]. 그리고, 이동 에이전트를 목적지 에이전트시스템으로 안전하게 전송하기 위하여 두 에이전트 서버는 두 단계의 메시지 교환을 한다. 첫 단계에서, 송신 서버는 수신 서버에게 에이전트 소유자의 사인을 포함한 이동 요청 메시지를 보내고, 수신 서버는 요청 메시지에 포함된 사인을 바탕으로 이동하려는 에이전트가 안전한 것임을 검증하여 그 결과를 송신 서버에게 전달한다. 두 번째 단계에서는 송신 서버는 인증이 성공적일 경우 에이전트를 전송하며 전송이 성공적이라는 확인 메시지를 받는다. 실패일 경우 예외처리를 한다.

4. 결 론

에이전트 패러다임은 자신의 코드를 이동시켜 목적지 시스템에서 실행시킨다는 특성 때문에, 현재 인터넷 환경에서 분산 애플리케이션을 개발하는데 널리 이용되고 있다. 하지만 대부분의 이동 에이전트 시스템들은 그 구조와 구현이 매우 달라 이종의 시스템에서 생성된 에이전트의 실행은 지원하지 않는다. 이러한 시스템들간의 차이는 에이전트 기술의 빠른 확산과 상호운용성을 가로막고, 에이전트 시스템의 적용을 어렵게 하고있다.

본 논문에서는 서로 다른 에이전트시스템간의 상호운용성과 이식성을 위하여 OMG MAF 명세를 만족하는 SMART 시스템을 Java 프로그래밍 언어를 이용하여 개발하였다. 그 결과 MAF 명세를 만족하는 이질적인 에이전트시스템간의 통신이 가능하며, 다른 시스템의 에이전트를 수행할 수 있게 되었다.

[참고문헌]

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

[2] Alfonso Fuggetta, Gian Pietro Picco, "Giovanni Vigna, Understanding Code Mobility", IEEE Transaction On S/W Engineering, Vol. 24, NO. 5, May 1998.

[3] William Li, David G. Messerschmitt, Java-To-Go Mobile Agent System, University of California at Berkeley, 1998

[4] William Li, David G. Messerschmitt, "Java-To-Go Itinerative Computing Using Java", University of California at Berkeley, 1998.

[5] Anand R. Tripathi, Neeran M. Karnik, Manish K. Vora, Tanvir Ahmed, and Ram D. Singh, "Ajanta - A Mobile Agent Programming System", 1998.

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

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

[8] B.Venners, "The Architecture of Aglets", JavaWorld, http://www.javaworld.com/javaworld/jw-04-1997/jw-04-hood.html, April 1997.

[9] Rocco De Nicola, Gian Luigi Ferrari, Rosario Pugliese, "KLAIM: A Kernel Language for Agents Interaction and Mobility", IEEE Transactions On SE, VOL. 24, NO. 5, May 1998.

[10] IBM Aglets Workbench-Home Page, 1999. URL address: http://www.trl.ibm.co.jp/aglets/