이동 에이전트의 최적 이주 비용을 위한 플랫폼 설계

김완성* · 복경수*· 신재룡** · 유재수*

Platform Design for Optimal Migration Cost of Mobile Agents

Wan Soung Kim*, Kyoung Soo Bok*, Jae Ryong Shin**, Jae Soo Yoo*

이 논문은 2003년도 산업자원부 차세대신기술개발 및 한국과학기술평가원의 생물정보학 연구개발 2차 사업의 지원에 의해 수행되었음

요 약

네트워크의 발달과 다양한 서비스의 요구에 따라 새로운 소프트웨어의 패러다임에 대한 요구가 증가되고 있다. 이와 함께 이동 에이전트에 대한 많은 연구가 진행 중이다. 이동 에이전트의 수행에 있어 이주비용은 이동 에이전트의 성능에 많은 영향을 미친다. 본 논문에서 이동에이전트의 이주비용을 최적화하기 위한 기법을 제안한다. 제안하는 이주기법의 특징은 다음과 같다. 첫째, 네트워크 상태 및 플랫폼 상태변화에 적절하게 대응할 수 있는 동적 경로를 생성하여 에이전트 수행 효율을 높인다. 둘째, 수행할 코드를 프리패칭하여 이동 데이터량을 줄이고, 필요한 에이전트를 미리 인스턴스시켜 수행 시간을 단축한다. 셋째, 체크포인트 기법을 사용하여 에이전트 수행 중에 에러가 발생할지라도 에이전트는 재 수행을 하지 않고 에러 이전의 상태로 복구하는 방법을 사용하여 수행 효율을 높인다. 또한, 시뮬레이션을 통해 기존방법과 제안하는 방법을 비교 평가한다. 시뮬레이션 결과분석을 통해 이주관점에서 제안한 방법들이 기존방법들에 비해 성능이 매우 향상됨을 보인다.

ABSTRACT

A new software paradigm is required on the development of network and various service requirements. With this, many studies on a mobile agent have been made. For the execution of the mobile agent, migration is the most important factor that influences the performance of the mobile agent. In this paper we propose the method that leads to high migration efficiency in order to improve the performance. The features of our migration technique are as follows. First, the migration technique creates the dynamic itinerary that appropriately copes with the network conditions and the platform changes to improve the agent execution efficiency. Second, it prefetches an executed code to reduce the amount of the mobile data and reduces the execution time by instantiating the agent in advance. Third, it improves the execution efficiency by using the checkpoint-based recovery method that does not execute the agent again and recovers the process states even though the errors take place. Though the simulation, we compare the proposed method with the existing methods. The simulation result shows that the proposed method outperforms the existing methods in terms of migration.

키워드

이동 에이전트, 이주, 동적 이동 경로, 결함허용, 프리패칭

* 충북대학교 정보통신공학과 및 컴퓨터정보통신연구소 ** 광주보건대학 인터넷정보과

접수일자 : 2003. 9. 24

Ⅰ. 서 론

최근 컴퓨터, 통신기술의 발전은 인터넷이라는 거대한 통신망을 만들어 놓았다. 이러한 거대한 네트워크 통신망은 유지, 보수, 관리에 어려움이 따르고, 네트워크 환경의 불확실성 하에서 예측하지 못한 여러 가지 문제를 발생시킬 수 있다. 이로 인하여 특정 작업을 수행 할 수 없게 되거나, 필요 이상의 비용 발생과 자원 소모를 유발 할 수 있다. 따라서 이에 적합한 새로운 소프트웨어 기법에 대한 요구와 함께 이동 에이전트 기반 시스템 기술들이 개발되었다[1].

에이전트란 사용자를 대신하여 자율적으로 환경을 감지하고 목적에 맞는 행위를 수행함으로써 환경에 적응해 나가는 존재로 볼 수 있다. 즉 에이전트는 사용자를 대신하여 사용자가 원하는 작업에 자율성을 가짐으로써 능동적으로 해결해 주는 소프트웨어이다. 에이전트는 자율성, 지능성, 사교성 그리고 이동성 등의 특징이 있다. 첫째, 자율성은 사용자나 다른 에이전트의 직접적인 지시나 간섭 없이도 스스로 판단하여 행동하는 성질을 의미한다. 둘째, 지능(Intelligence)은 지식베이스와 추론 능력을 갖추고 사용자의 의도를 파악하여 계획을 세우고 학습을 통하여 새로운 지식을 스스로 터득하는 성질이다. 셋째, 사교성은 에이전트간 통신능력을 의미한다. 마지막으로 에이전트가 갖추어야 할 기본 특성중의 하나인 이동성은 기존의 클라이언트/서버의 개념과는 판이하게 다른 개념으로 서버의 내용을 클라이언트를 통해 전송 받아 정보를 얻거나 작업수행을 하는 것이 아니고, 클라이언트가 필요로 하는 작업을 위해 에이전트를 서버로 보내어 수행을 시킨다. 인터넷의 보급으로 컴퓨터 네트워크를 통해 제공되는 정보의 수가 급증하면서 이동 에이전트의 중요성이 강조되고 있으면 원격 통신 시에도 통신라인이 항상 접속되어 있을 필요가 없기 때문에 무선이동통신을 위한 작업수행 환경에서 큰 효과를 낼 수 있다[1].

이동성에 따라 에이전트는 정적 에이전트와 이동 에이전트로 구분되며, 이동 에이전트는 에이전트간에 코드 및 데이터 그리고 상태정보를 이동하여 다음 목적지 플랫폼의 에이전트에 인스턴스(Instance)를 생성한다. 이러한 이동 에이전트는 클라이언트/서버 형태로 네트워크를 이용해 작업을 수행하는 일반 에이전트와는 달리 시스템간에 코드와 현재의 에이전트 내부상태 자체가 이동하여 작업을 수행하는 에이전트로 네트워크 에이전트(Network agent), 순회 에이전트(Itinerant agent)라 불린다. 이동 에이전트는 에이전트 자체가 이동하기 때문에 보다 확장성 있게 응용할 수 있다[1].

이동 에이전트의 장점은 다음 7가지로 구분하여 설명할 수 있다. 첫째, 네트워크 부하와 트래픽을 줄여준다. 둘째, 대기시간을 줄여준다(수행할 실행코드와 데이터를 한 번에 이동하여 로컬 시스템에서 업무를 수행 시 리얼타임 응답이 필요한 지역 분산시스템에 응용이 가능하다). 셋째, 프로토콜을 캡슐화 할 수 있다(네트워크 상에서 데이터가 움직일 때 프로토콜을 구현한 객체를 캡슐화하여 이동하는 것이 가능하다. 따라서 이동 에이전트 사이의 독자적인 프로토콜 구현과 통신이 가능하다). 넷째, 자율적이고 비동기적으로 수행될 수 있다(자주 접속이 끊기거나 회선상태가 안 좋은 곳에서 효율성을 높일 수 있다). 다섯째, 동적인 적응성을 갖는다(주위 환경의 변화에 능동적으로 반응하고 동적으로 자기 자신을 이동시킬 수 있기 때문에 최적화된 성능을 낼 수 있다). 여섯째, 이 기종간에 수행이 용이하다(가상머신 위에서 수행되므로 하드웨어와 소프트웨어 버전에 무관하게 수행환경 구성이 가능하다). 일곱째, 강인하며 에러에 강하다(예상치 못한 상황과 예외사항에 대한 지능적 처리가 가능하므로 강인하고 에러에 영향을 덜 받는다).

이러한 장점을 갖는 이동 에이전트의 성능은 이동 데이터 크기 및 네트워크의 상태 그리고 이동할 플랫폼의 상태에 따라 영향을 받는다[2]. 그러나 기존의 JAMES[3,4], AJENTA[5,6], Aglets[7], Grasshopper[8], Tacoma, Agent-Tcl, Concordia 그리고 Odyssey 등의 이동 에이전트의 플랫폼들은 동적인 환경변화(네트워크 상태, 플랫폼의 자원상태) 및 이주비용에 대한 특징(이동 데이터 크기)을 고려하지 못 하고 있다. 또한 일관성 있는 이주기법 즉 결함에 대한 처리를 제공하지

못하는 문제점 때문에 성능에 문제를 야기시켰다. JAMES는 프리패칭(Prefetching) 기법을 이용하여 이주할 때 이동 데이터 크기를 줄여 네트워크 트래픽을 개선하였지만 동적인 환경변화에 적절하게 대응하지 못하는 문제점과 결함에 대한 처리를 제공하지 못했다[3,4].

이러한 문제점을 해결하고자 본 논문에서는 최적의 이주 비용과 일관성을 제공하는 이주기법을 제안한다. 제안하는 기법은 정적, 동적 경로를 혼합한 방법, 프리패칭 및 체크포인트 기술을 사용하여 동적인 환경변화 및 이주 비용에 대한 특징을 고려한다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 이동 에이전트의 개념 과 이주기법 그리고 기존에 개발된 에이전트 시스템의 이주기법 및 특징을 기술한다. Ⅲ장에서는 제안하는 최적의 이주비용을 위한 에이전트 이주기법에 대해서 기술한다. Ⅳ장에서는 제안한 방법과 기존의 방법들을 시뮬레이션(Simulation)을 통해서 비교 평가한 결과를 제시하고 분석을 한다. 마지막으로 Ⅴ장에서 결론에 대해 기술한다.

Ⅱ. 관련연구

이 장에서는 전반적인 이동 에이전트 개념과 이주기법에 대해 기술하고, 기존에 개발된 에이전트 시스템의 이주 기법 및 특징을 기술한다.

1. 이주(Migration) 기법

이동 에이전트의 이주는 이동하는 정보에 따라 그림 1에서처럼 메시지 패싱, 원격 실행, 약 이주 그리고 강 이주로 구분된다. 메시지 패싱은 최초로 제안된 기법으로 프로세스간 메시지를 보내고 받음으로써 통신을 한다. 원격 실행의 개념은 노드에서 노드로 코드를 보내고, 가져오는 방법으로써 자바의 애플릿과 마이크로소프트의 액티브X가 이에 해당된다. 원격 실행은 코드만 이동하는 반면 이동 에이전트는 컴퓨터 네트워크상에서 노드에서 노드로 프로그램 실행상태를 이주한다. 이러한 이동 에이전트는 코드 및 데이터 그리고 실행상태를 이동객체에 저장하여 이주한다.

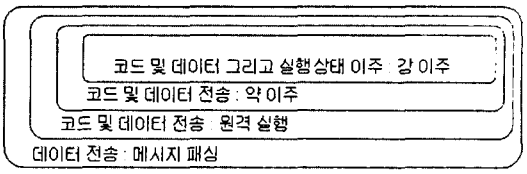


그림 1. 이동 등급
Fig. 1 Degrees of Mobility

이동 에이전트의 이주 기법은 그림 2에서 보는 것과 같이 이주되는 정보에 따라 약 이주와 강 이주로 구분된다. 강 이주는 목적지에 코드 및 데이터 그리고 실행상태를 이주하여 인스턴스를 생성하고 실행하는 기법이다. 이동객체에는 코드와 데이터상태 그리고 실행상태 또는 쓰레드 정보를 담고 있다[1].

이동 에이전트에서 이주 기법은 세 가지 관점으로 구분할 수 있다. 첫째, 프로그래머 관점에서 보면 약 이주와 강 이주로 구분되며, 둘째 에이전트 관점에서 보면 이주 전략 즉, 어떻게 이주를 할 것인가에 대한 방법으로 푸쉬와 풀 방식으로 구분된다. 셋째, 네트워크 관점에서 보면 코드와 데이터를 프로토콜(TCP/IP, UDP) 레벨에서 어떻게 보낼 것인가에 대한 전송전략으로 구분할 수 있다[9].

최근에는 이러한 객체이동을 위해 플랫폼 독립성을 장점으로 다양한 환경에 적용되고 있는 자바를 사용하여 개발하고 있으며, 자바는 코드이동 및 데이터 이동을 지원하지만, 스택에 직접 접근이 불가능하기 때문에 완벽한 쓰레드 이동을 지원하지 못한다. 즉 강 이주를 완벽하게 지원하지 못하기에 이 분야에서 많은 연구가 진행중이다. 강 이주 기법을 두 가지로 구분해 볼 수 있다. 자바 가상 머신의 바이트 코드를 수정하여 스택 정보를 가져오는 방법과 어플리케이션 레벨에서 실행 상태정보를 자바의 객체에 프레임별로 저장하여 이동시키는 방법으로 구분되어 개발되고 있다[10,11].

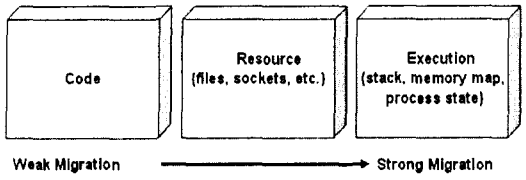


그림 2. 약 이주와 강 이주
Fig. 2 Weak Migration and Strong Migration

2. 기존 에이전트 시스템의 이주 기법

이동 에이전트의 이주기법에 대한 많은 연구가 진행중이며, 기존에 개발된 이동에이전트 플랫폼인 JAMES, AJENTA, Aglets 그리고 Grasshopper의 이주기법에 대해서 기술하고, 특징 및 문제점을 살펴본다.

2.1 JAMES

기존의 클라이언트/서버 환경에서는 유연하게 코드를 지원하게 코드를 지원하지 못했고, 네트워크 트래픽 및 지연시간이 문제가 되었다. 이러한 문제를 개선하고자 이동 에이전트가 나오게 되었고 이런 이동 에이전트의 특징은 강건하고 쉬운 갱신 및 네트워크 트래픽 감소의 효과를 가져오게 되었다. 이에 JAMES 프로젝트는 이동 에이전트의 하부구조를 네트워크 관리 및 통신에 있어 성능을 향상시키고자 개발된 플랫폼으로 Siemens SA, University of Combra, Siemens AG가 공동으로 개발하였다. 주요 특징으로는 고성능의 수행효율, 효과적인 코드이주, 결함허용, 강건한 네크워크 관리의 특성이 있다. 효과적인 코드이주를 위해 코드 프리패칭, 캐쉬 스키마, 쓰레드 및 소켓 풀링의 기법을 적용하였다. JAMES의 이주기법은 클라이언트가 서비스를 요청하게 되면 중앙 호스트에서 이동 에이전트가 생성되고 정적 경로를 생성한다. 경로 정보에 따라 필요한 코드를 각 에이전트 플랫폼에 메시지 형태로 보낸다. 메시지를 받은 에이전트 플랫폼에서는 자신이 필요한 코드를 자신의 캐쉬메모리, 이전 에이전트 플랫폼의 캐쉬메모리, 자신의 보조기억장치에서 코드를 찾는다. 만약 코드가 없으면 JAMES 코드 서버에 필요한 코드를 요청하여 미리 코드를 받는다. 이러한 프리패칭 기법을 이용한 JAMES 플랫폼은 이주할 때 발생 할 수 있는 네트워크 트래픽을 개선하였다[3,4]. 그러나 JAMES의 이주기법은 동적인 환경변화 즉, 목적지 플랫폼의 자원 그리고 네트워크의 상태를 고려하지 못 하는 문제점과 결함에 대한 처리를 제공하지 못했다.

2.2 AJANTA

Minnesota 대학의 Anand Tripathi와 연구원들이 보안 및 강건한 하부구조를 만들 목적으로 이동 에이전트 플랫폼을 개발하였다. AJANTA는 자바의 직렬화, 반사(Reflection), RMI(Remote Method Invocation) 기능을 이용하여 이동 에이전트를 구현하였고, 네트워크상에서 플랫폼과 플랫폼의 이동에 있어 자바의 직렬화 기법을 이용하여 상태정보를 이주하였으며, 클래스 로딩 기법을 이용하여 요구한 클래스만을 이동시키는 경량의 약 이주 기법을 사용하였다. 에이전트 수행을 위해서는 Proxy Interposition을 통해 플랫폼 자원에 접근하여 수행하는 AJANTA의 이주방법은 세 가지 방법이 존재한다. AJANTA의 주요기법은 클래스 로딩 기법을 이용하여 수행에 필요한 코드만을 이동하는 방법을 사용하여 데이터의 양을 최소화함으로써 네트워크 부하를 줄였고 성능을 향상시켰다. 그리고 에이전트의 수행에 적합한 이주 패턴(Pattern)을 선택하고, 선택된 이주 패턴에 따라 에이전트를 수행함으로써 성능을 향상시켰다[5,6]. 그러나 AJANTA의 이주기법은 요구한 클래스만을 이동시키는 방법을 사용했지만, 이주 데이터의 크기를 고려하지 못하는 문제점과 결함에 대한 처리를 고려하지 못했다.

2.3 Aglet

Aglet는 IBM사의 Tokyo Research Lab에서 개발한 이동 에이전트 시스템이다. 실행환경과 이동 에이전트 프레임워크(Framework)를 자바 클래스 라이브러리 형태로 제공하여 개발을 쉽게 하였다. 실행 환경은 Tahiti라 불리는 GUI 기반의 자바 응용으로 이동 에이전트 관리를 쉽게 하려는데 초점을 두었고, 네트워크상에서 클래스 파일들을 이주하는 방법은 애플릿과 비슷하다. 애플릿과 다른

점은 상태 정보까지 이주한다는 것이다. Aglet에서 제공하는 이주 서비스는 dispatch, retractAglet, MobilityListener에 의해 서비스를 제공한다. dispatch는 에이전트를 새로운 호스트에 직렬화를 통해 전달하고 인스턴스를 생성한다. 이러한 기능은 run() 메소드를 통해 실행하는 역할을 한다. retractAglet는 dispatch와 같은 기능을 하지만 dispatch와는 다르게 풀(Pull) 방법을 이용하여 이전 호스트로 에이전트가 돌아가게 한다. 마지막으로 MobilityListener는 두가지의 메소드 dispatch, retractAglet의 이벤트를 제공한다. Aglet이 이주 시 클라이언트 정보 및 이동할 주소 그리고 Aglet에서 사용된 클래스 바이트 코드를 클론에 의해 생성된 바이트 배열에 저장하여 목적지에 이주시킨다. 즉 자바 객체 직렬화 기법과 비직렬화 기법을 사용하여 Aglet의 코드와 스택 정보는 보내지 않고 힙에 있는 상태 정보를 스트림으로 보내고 받는 약 이주 기법을 사용하였다[7]. Aglet의 문제점은 동적인 환경변화(목적지 플랫폼의 자원 및 네트워크의 상태)와 이주 데이터의 크기 그리고 결함에 대한 처리를 고려하지 못했다.

2.4 Grasshopper

Grasshopper는 GMD FOKU와 IKV++에서 이동 에이전트 실행환경을 개발한 것으로 FIPA (Foundation for Intelligent Physical Agents) 스펙에 따라 최초로 개발되었다. 소켓, RMI, CORBA, Socket+SSL(Secure Sockets Layer), RMI+SSL 기술과 자바를 사용하여 시스템이나 하드웨어를 수정하지 않고 현재 컴퓨터 시스템에 적합한 플랫폼을 개발하였다[8].

Grasshopper는 실행상태를 제외한 코드 및 데이터를 목적지 에이전시에 이동시키는 약 이주 기법을 사용했으며 에이전시 UI, 에이전시 API, 에이전트 API를 통해 이주 할 수 있다. 에이전시 UI을 이용하여 에이전시 관리자가 그래프 또는 텍스트 사용자 인터페이스를 통해 이동시키고 에이전시 API를 이용하여 다른 에이전트나 객체가 moveAgent() 메소드를 호출해서 이동시킨다. 마지막으로 에이전트 API를 이용하여 에이전트 스스로 이주한다. 커뮤니케이션 서비스가 서버 에이전트와 클라이언트 에이전트 이주 경로를 유지하고 있으며, 클라이언트의 요청에 의해 에이전트가 생성되어 수행하면서 에이전트가 이주할 때는 다음과 같은 절차를 통한다. 에이전트 이주 초기화 -> beforeMove() 메소드 호출 -> 에이전트 실행(에이전트 쓰레드) 일시정지 -> 이동 데이터 직렬화(serialization) -> 추가적인 정보 직렬화 -> 목적지 에이전시(agency)에서 에이전트 인스턴스 생성 -> 목적지 에이전시는 이동 데이터를 받았다고 메시지 전달 -> 목적지 에이전시는 자동적으로 afterMove() 메소드 호출 -> 목적지 에이전시는 에이전트를 실행하는 방법을 사용하였다[8]. 이러한 이주 기법을 사용한 Grasshopper는 동적인 환경변화(목적지 플랫폼의 자원 및 네트워크의 상태)와 이주 데이터의 크기 그리고 결함에 대한 처리를 고려하지 못했다.

III. 최적의 이주 비용을 위한 에이전트 설계

이 장에서는 이동 에이전트에 대한 최적의 이주기법을 위한 방법을 제안한다. 제안하는 방법의 기본적인 목적은 플랫폼의 환경변화와 네트워크의 상태변화 그리고 시스템 결함에 적절하게 대응하여 성능을 낼 수 있도록 하는 것이다.

최적의 이주 비용을 위한 에이전트 이동을 위한 플랫폼 서비스들의 기능에 대해서 기술하고, 이주기법에 필요한 기술에 대해서 설명한다. 마지막으로 제안하는 최적의 이주 비용을 위한 에이전트 설계에 대해서 기술한다.

1. 이동 에이전트 이주를 위한 서비스의 종류

이동 에이전트의 실행을 위한 환경은 기능별로 크게 세 부분으로 나눌 수 있다. 첫째, 특정 목적의 에이전트를 생산하고 활동하는 영역이 있다. 둘째, 에이전트의 관리와 운영에 필요한 환경을 제공하고, 에이전트에게 필요한 서비스를 제공하는 영역이 있다. 셋째, 원격 플랫폼에 대한 발견과 물리적인 통신을 제공하는 영역이 있다.

이러한 영역 중에 이주에 관련된 서비스들은

다음과 같다. 사용자가 서비스를 요청하게 되면 이동 에이전트가 생성한다. 생성된 이동 에이전트는 컨테이너(Container)에 있는 로컬자원관리(Local Resource Management), 에이전트모니터링(Agent Monitoring), 에이전트스케쥴링(Agent Scheduling), 에이전트등록(Agent Registration), 통신(Communication) 서비스와 커뮤니케이션레이어(Communication Layer)의 원격플랫폼(Remote platform), 플랫폼 광고(Platform Advertisement)를 이용하여 최적의 이주를 제공한다.

로컬자원관리는 CPU 점유율, 메모리 점유율, 네크워크 사용량 등의 플랫폼의 자원에 대한 관리를 하며, 모니터링 자원에 대해서 일정간격을 두고 계속적으로 정보를 제공하거나, 모니터링 정보를 즉각 제공한다. 에이전트 모니터링은 플랫폼 상에 있는 에이전트에 대한 정보(에이전트 수, 상태, 기능)를 제공한다. 에이전트 스케쥴링은 플랫폼 상에 있는 에이전트들을 효율적으로 수행시키기 위해 쓰레드를 생성 할당시키며, 에너지 양을 확인하여 에이전트를 적절한 상태(활동상태, 휴면상태 등)로 변화시킨다. 에이전트 등록은 에이전트가 처음 생성 될 때, 플랫폼에 유일한 식별자를 부여하여 에이전트를 등록한다. 이는 플랫폼에 존재하는 에이전트들의 디렉토리 역할을 한다. 커뮤니케이션레이어의 원격플랫폼은 플랫폼의 식별자, 지원하는 서비스 종류 등을 바탕으로 외부 플랫폼을 탐색하고 플랫폼의 정보 및 에이전트 정보를 얻는다. 플랫폼 광고는 플랫폼의 정보를 네트워크 상에 브로드캐스팅하여 메시지를 받은 플랫폼은 해당 플랫폼 정보를 플랫폼 데이터베이스에 저장하여 다른 플랫폼들에 대한 정보를 유지하게 된다.

2. 에이전트 이동을 위한 기능

2.1 직렬화

이동 에이전트는 코드를 이동시키는 것이기 때문에 객체 직렬화를 잘 구현하는 것이 성능에 중요한 영향을 미친다. 직렬화란 데이터가 아니라 객체를 저장하고 읽어 들이는 방법으로 객체의 내용을 바이트 단위로 변환하여 파일 또는 네트워크를 통해서 송수신(스트림, Stream)이 가능하게 만들어 주는 것을 말한다. 즉 복잡한 데이터 구조를 갖는 객체를 직렬화 시켜 파일이나 배열에 저장하고 이를 객체로 다시 복원해내는 것이다. 이 기술은 더 나아가 네트워크 상에서 코바 기술을 이용하여 객체를 전송하는 데 활용 할 수 있다. 이주시 직렬화 포멧에 맞게 객체에 저장하여 이동한다. 그림 3은 객체 직렬화 기법을 보여주고 있다[12].

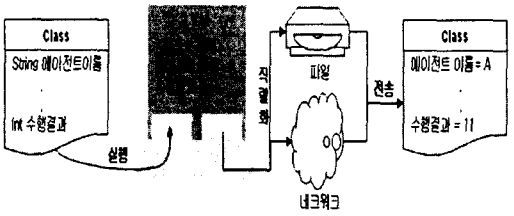


그림 3. 객체 직렬화
Fig. 3 Object Serialization

2.2 동적 클래스 로더(Dynamic class loader)

유닉스 시스템의 .os파일, 윈도우 시스템의 DLL파일은 동적으로 링크되는 파일이다. 자바가상머신의 경우에는 모든 프로그램이 동적으로 링크되어 있다. 다시 말해 모든 클래스들은 한 번에 하나의 클래스만 적재하고서 필요할 때마다 새로운 클래스를 적재해 나간다. 즉 프로그래머는 모든 프로그램이 동적으로 적재되므로 별다른 구분 없이 프로그램 내의 모든 부분들에 대해서 동일하게 접근할 수 있다. 그림 4 는 클래스 로더의 구조이다. 자바가상머신이 클래스 파일을 동적으로 로드할 수 있는 것은 java.lang. ClassLoader 라는 클래스에 의해서 가능하다.

클래스 로더는 적재(loading), 연결(linking), 초기화(initialization)의 과정을 거쳐 클래스를 사용할 준비를 한다. 적재는 클래스의 이름을 사용하여 클래스 파일 형태의 바이트를 찾고, 클래스의 구현 정보를 자바가상머신에게 알리는 작업을 수행한다. 적재 단계에 의해, 자바가상머신은 클래스의 이름과 클래스 계층도 상에서의 클래스 위치 및 그 클래스의 필드와 메소드의 종류를 알게 된다. 연결은 클래스가 기본적인 형태를 제대로 갖

추고 있는지, 자바가상머신의 보안과 관련된 제약 조건에 맞는지에 대한 클래스의 검증하는 단계, 그리고 스택 초기화를 위한 모듈을 수행한다. 이 검증과정의 부가적 영향으로 여러 클래스들이 함께 적재되며, 이 작업이 끝나면 클래스를 사용할 준비는 완료된다[10].

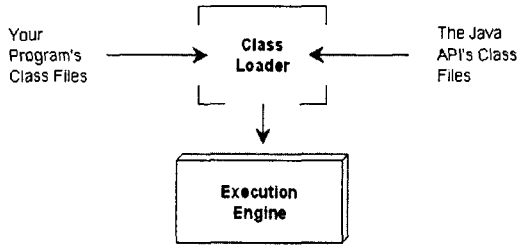


그림 4. 자바 클래스 로더 구조
Fig. 4 Java's class loader Architecture

2.3 프리패칭 기법

프리패칭은 가까운 미래에 참조될 가능성이 높은 데이터 블럭을 예측하여 이를 미리 프로세서에 인접한 메모리로 가져와서 메모리 접근 시간을 줄인다.

프리패칭의 목적이 네트워크 왕복을 최대한 줄이고, 업무처리를 최소화하는 것이 목적인 만큼, 이것이 적용된 네트워크 내에서는 빠른 응답시간과 지연시간 감소로 로컬 영역 및 글로벌 영역에서의 높은 업무 효율을 기대할 수 있다. 지연시간 감소는 로컬영역의 스토리지, 메모리, 네트워크량 등의 자원을 절약할 수 있으며, 메인 서버 측으로의 데이터 접근 횟수와 처리량을 늘릴 수 있다. 물론 로컬영역으로의 데이터 프리패칭으로 인해 로컬 영역의 부담이 늘어나는 것은 어쩔 수 없다. 즉 팻 클라이언트 시스템이라고 볼 수 있는데, 이는 업무특성에 따라서 원거리 네트워크 상에서의 업무 적용 시 적용되는 형태이다. 반면 근거리 네트워크에서는 로컬 영역에 데이터를 패치하여 작업하는 것보다는 중간 미들웨어나 캐쉬 서버 측에 프리패칭을 적용시키는 것이 일반적인 형태이다. 이는 각 업무별 작업 효율에 따라서 프리패칭의 위치를 알맞게 적용하는 것이 가장 올바른 형태일 것이다. 즉 데이터의 패치 크기에 따라서 캐쉬의 적용이 달라지게 된다. 물론, 원거리의 다량 데이터 보다, 근거리의 소량 데이터들의 캐싱이 훨씬 간편하며 이는 곧 프리패칭의 적용 예를 소량의 데이터에 초점을 맞추는 것이 훨씬 효율적이라는 것을 보여준다. 이러한 프리패칭 기법을 이용하여 네트워크상에서 이동 에이전트에 적용하여 성능을 높이고자 한다. 이동 에이전트에서 에이전트는 플랫폼을 이동하면서 에이전트를 수행한다. 수행 시 필요한 코드를 미리 이동할 플랫폼으로 이동 후 인스턴스화 하면 에이전트 이주 시 이동정보 양이 줄어 이주시간을 줄일 수 있고, 성능을 향상시킬 수 있다[4].

2.4 체크포인트 기법

시스템 결함이 발생했을 때 기록된 로그를 참조하여 복구하게 된다. 그런데 이 때 전체 기록된 로그를 검색해야 한다면, 로그 탐색 시간 및 복구 시간이 매우 길게 된다. 서버를 빨리 복구해야 하는 경우 이러한 긴 복구시간은 바람직하지 않다. 그러므로 주기적으로 체크포인트를 실행함으로써 복구시간을 줄이려는 노력이 필요하다. 체크포인트는 시스템의 안정성에 따라 주기가 결정되어야 한다. 시스템이 불안정하다면 체크포인트를 자주 하는 것이 유리할 것이다. 결함이 전혀 발생하지 않는 시스템에서는 체크포인트를 하는 것이 오히려 불필요한 오버헤드가 된다. 체크포인트를 자주하면 할수록 시스템을 복구하는데 걸리는 시간은 짧아지게 된다. 이는 로그의 분석 시간이 짧아지기 때문이다. 에이전트가 실행 중 또는 이주 시 에러에 대한 처리를 높이고자 이동 전, 후에 실행 상태 및 수행정보를 안전한 저장장치에 주기적으로 저장하여 에러에 대한 처리를 효율적으로 높이는 기법을 사용하여 에이전트를 재 수행하지 않고 저장한 내용을 인스턴스화 하여 에이전트 수행을 연속적으로 할 수 있게 함으로써 성능 향상을 가져온다.

3. 제안하는 이주 기법

최적의 이주를 위한 이주 API, 이주 기법, 동적 경로 생성방법, 결함허용 에이전트, 네트워크 부하 및 지연시간 해결방법 그리고 설계된 이주 기법의 특징에 대해서 기술한다.

3.1 이주 기법

에이전트 수행과정은 그림 5 에서 보는 것과 같이 4 단계의 과정을 거쳐 수행한다. 1 단계에서는 클라이언트의 서비스 요청에 의해 에이전트가 생성되고, 2 단계에서는 에이전트를 수행하며, 실질적인 이주단계는 3 단계와 4 단계에서 이루어지며, 에이전트 생성 시에만 1단계를 수행하고, 생성 후 에이전트 수행이 완료될 때까지 2 단계에서 4 단계까지 반복 수행하면서 에이전트의 임무를 수행한다.

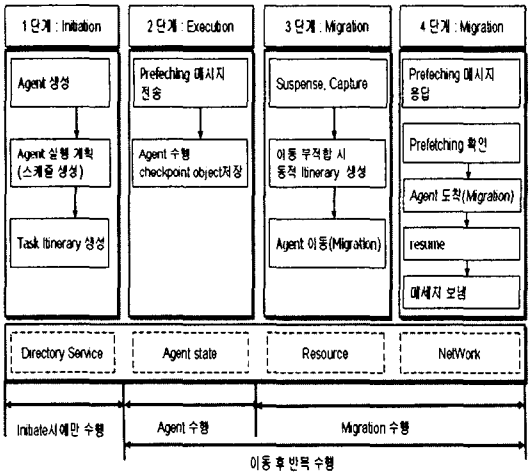


그림 5. 에이전트 수행 과정
Fig. 5 Agent Execution Mechanism

3.2 이주 API

이동 에이전트의 이주를 위한 API는 아래와 같이 다섯 부분으로 구성되어 있으며, 각각의 API에 대한 기능과 파라미터 표 1과 같다.

표 1. 이주 API
Table. 1 Migration API

requestMigration(AgentID, Itinerary)	
기능	에이전트 요청 또는 플랫폼의 상태에 따라 에이전트를 다른 플랫폼으로 이동시키는 기능을 수행한다.
파라미터	AgentID : 이주를 요청한 에이전트 식별자 Itinerary : 해당 플랫폼에서 생성된 에이전트의 이동 경로
requestPrefetching(AgentID, Itinerary, N)	
기능	에이전트의 이동 경로가 생성되면 다음에 이동할 플랫폼에 필요한 코드를 미리 프리패칭 하라는 메시지를 전송한다. 옵션 형태로 사용할 수 있으며 프리패칭 기능을 수행하여 에이전트의 이동 후 수행에 필요한 전처리 시간을 줄일 수 있다.
파라미터	AgentID : 프리패칭 메시지를 요청한 에이전트 식별자 Itinerary : 해당 플랫폼에서 생성된 에이전트의 이동 경로 N : 프리패칭 메시지를 전달할 플랫폼의 개수
receivePrefetching(PlatformID,AgentID, Code, Flag)	
기능	프리패칭 메시지를 수신한 플랫폼에서 각 플랫폼의 캐쉬를 검사하여 필요한 코드를 수신하여 컨테이너에 등록한다. 프리패칭 메시지의 우선 순위에 따라 필요한 코드를 메모리에 인스턴스화한다.
파라미터	PlatformID : 프리패칭 메시지를 전달한 플랫폼 식별자 AgentID : 프리패칭 메시지를 전달한 에이전트 식별자 Code : 프리패칭할 코드 Flag : 인스턴스를 생성할 것인지에 대한 여부
receiveMigration(PlatformID, AgentID, Code, Data, Itinerary)	
기능	에이전트 또는 플랫폼의 요청에 따라 이동된 에이전트의 코드와 데이터를 수신하고 수행 가능한 상태로 생성한다.
파라미터	PlatformID : 에이전트가 이동하기 전 플랫폼 식별자 AgentID : 에이전트 식별자 Code : 전송되어온 코드 Data : 이동하기 전에 만들어진 데이터 Itinerary : 에이전트의 이동 경로
checkpointAgent(AgentID)	
기능	에이전트의 수행 상태를 안전한 저장 장치에 기록하고 에러가 발생할 경우 복구 할 수 있는 기능을 제공한다. 체크포인트에 의해 기록된 내용은 로그 형태로 기록되며 응용 프로그램 개발자에 의해 선택적으로 사용될 수 있다. 그러나 에이전트가 생성되거나 이동할 때에는 의무적으로 체크포인트가 내부적으로 수행된다.
파라미터	AgentID : 에이전트 식별자

3.3 동적 경로 생성

기존의 에이전트 시스템들은 처음 에이전트 생성 시 디렉토리서비스를 통해 각 플랫폼에서 수행 가능한 서비스 검사, 통신비용, 플랫폼 상태, 자원 할당 비용검사를 하여 최적의 에이전트 플랫폼 리스트를 생성하는 정적인 경로를 사용하였다. 이러한 정적 경로는 이동 에이전트 생성 시 한번만 생성하여 사용한다는 장점이 있고 에이전트 수행 중, 네트워크의 환경변화(네트워크 부하, 지연시간) 및 다음 목적지의 플랫폼 상태변화(플랫폼 자원 부족)에 따라 적절하게 대응하지 못하는 단점이 있다. 반면 동적 경로는 에이전트가 생성되어 다음 목적지 플랫폼으로 이동하기 전에 네트워크 및 플랫폼의 상태를 검사하여 에이전트 수행가능 여부를 확인하고 에이전트 실행환경에 적합한 경로를 생성한다. 동적 경로는 이주 시 매번 네트워크 및 플랫폼의 상태를 검사하여 새로운 경로를 생성하기 때문에 경로생성에 따른 소요시간이 증가하게 되는 단점이 있다[14]. 이러한 정적 및 동적 경로의 특징을 고려한 경로 생성방법을 제안한다. 에이전트를 생성 할 때 컨테이너의 에이전트 스케쥴링, 에이전트 모니터링, 로컬 자원 관리, 에이전트 등록 서비스를 통하여 각 플랫폼에서 수행

표 2. 정적 경로와 동적 경로 비교
Table. 2 A comparison between static itinerary and dynamic itinerary

	정적 경로	동적 경로
생성시기	에이전트 생성	에이전트 이주
장 점	한번만 생성	이주 시 마다
단 점	시간에 따라 네트워크 및 플랫폼 고려하지 않음	경로 생성에 따른 수행시간 증가

가능한 서비스 검사, 통신비용, 플랫폼 상태, 자원 할당 비용검사를 하여 최적의 이주경로를 생성한 후 에이전트가 이동하면서 각각의 플랫폼에서 에이전트 실행 후 다음 이주할 플랫폼의 상태 및 네트워크상태(네트워크 부하, 지연시간)를 컨테이너의 서비스들을 통해 검사한 후 에이전트가 수행 가능할 경우 정적 경로에 따라 에이전트를 수행하고, 수행 불가능 할 경우 컨테이너의 서비스들을 통해 새롭게 최적의 경로를 생성한 후 새로운 경로정보에 따라 에이전트를 수행한다.

3.4 결함허용(Fault-Tolerant) 에이전트

이동 에이전트는 여러 플랫폼으로 이주하면서 실행하기 때문에 결함은 에이전트 수행 중 시스템 결함과 이주 시 네트워크에 따른 결함으로 구분할 수 있다. 이와 같은 결함의 발생으로 인해 컴퓨팅 손실 시간 및 에이전트 수행시간 증가 그리고 에이전트 소멸을 초래한다. 제안하는 에이전트는 그림 6에서 같이 결함에 대해 체크포인트 에러복구 기법을 사용하여 결함에 따른 에이전트 복구시간을 줄여 전체적인 에이전트 수행 효율과 일관성을 높이고자 한다.

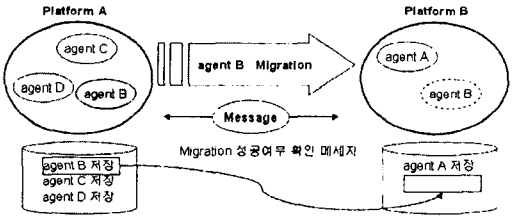


그림 6. 이주 시 체크포인트 메카니즘
Fig. 6 A checkpoint mechanism on migration

결함허용 에이전트는 결함에 대한 처리를 다음과 같이 처리한다. 첫째, 에이전트 수행 중 시스템 결함에 대한 처리는 에이전트의 수행결과를 주기적으로 체크포인트 하여 에이전트를 재 수행하지 않고 결함 이전의 상태로 복구하는 방법을 사용하였다. 둘째, 이주 시 네트워크에 따른 결함은 이주 시 에이전트의 데이터 및 상태정보를 주기적으로 체크포인트하고 있기 때문에 안정된 디스크에 저장되어 있다. 즉 이전 플랫폼에서는 체크포인트시 저장한 정보를 삭제하지 않고 기다렸다가, 다음 목적지 플랫폼으로 완벽한 이주가 되어 인스턴스를 생성 한 후 실행할 준비가 되었다는 메시지를 받으면, 삭제하기 때문에 이주 시에 따른 결함을 해결할 수 있다. 이주가 완료되었다는 메시지를

받으면 이전 플랫폼에 체크포인트 시 저장한 정보를 삭제한다. 삭제하는 이유는 디스크의 활용을 높이는데 있다. 이러한 체크포인트 기법을 이용하여 에러에 대한 처리 및 복구시간을 단축하여 성능을 향상시킬 수 있다.

3.5 네트워크 부하 및 지연시간 해결방법

이동 에이전트는 플랫폼에서 플랫폼으로 이동하면서 에이전트의 임무를 수행한다. 이주 시 코드 및 데이터 그리고 상태정보를 이동하여 수행한다[6]. 에이전트의 경로가 정적 또는 동적으로 생성되면 그림 7 과 같이 경로 정보에 따라 이동할 에이전트 플랫폼(메세지를 보내는 플랫폼의 다음 플랫폼을 제외한 플랫폼)에 필요한 코드 메시지를 보낸다. 메시지를 받은 플랫폼은 송신측 플랫폼에게 메시지를 받았다고 전달해 주고 필요한 코드를 원격 디스커버리(Remote Discovery)를 통해 정보를 얻어 코드를 받아온 후 인스턴스를 생성한다. 다음 이동할 플랫폼에 메시지를 전달하지 않는 이유는 지금 수행 후 바로 이동해서 수행하기 때문에 성능 향상에는 효과가 없으므로 보내지 않으며, 핸드셰이크를 통해 메시지 전달 결과를 알 수 있다. 이동 에이전트가 이주 후 새로운 플랫폼에서 제일 먼저 프리패칭 메시지를 통해 받아온 코드가 있는가를 검사하게 된다. 메시지가 제대로 전달되었지만, 미리 코드를 받아오지 못했을 경우 필요한 코드를 다시 받아오게 된다. 이러한 프리패칭 기법을 이용하여 필요한 코드를 미리 보내어 네트워크 부하 및 네트워크 지연시간을 줄여 수행 성능을 높일 수 있다.

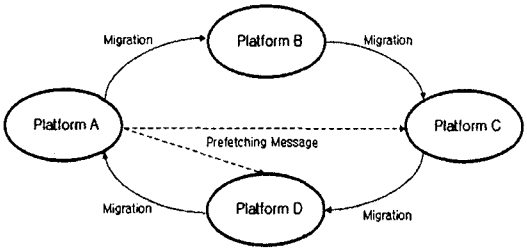


그림 7. 프리패칭 메시지
Fig. 7 Prefetching Message

3.6 설계된 이주 기법의 특징

설계된 최적의 이주 비용을 위해 이동 에이전트는 이동 데이터 크기 및 네트워크의 상태 그리고 이동할 플랫폼의 상태를 고려하여 이주기법을 설계한다. 이러한 이주기법을 사용함으로써 다음과 같은 이점이 있다. 첫째, 각 플랫폼에서 수행 가능한 서비스 검사, 통신비용, 플랫폼 상태 그리고 자원할당 등의 비용검사를 통해 최소의 비용으로 에이전트 수행 가능한 플랫폼의 리스트를 생성하여 에이전트 생성 시 정적 경로를 구성하고, 에이전트 수행 중 네트워크 부하 및 플랫폼 자원 부족 등의 문제가 발생 시 동적 경로를 생성함으로써 실행환경변화에 적절하게 대응할 수 있으며 성능향상을 가져온다. 둘째, 코드 및 데이터 그리고 실행상태를 이주 후 새로운 플랫폼의 안정된 저장장치에 체크포인트 기법을 이용하여 에이전트 정보(코드, 데이터, 실행상태)들을 저장 후 이전 플랫폼에 이주에 대한 메시지를 보내어 플랫폼 결함이 발생하거나 이주 시 네트워크 에러에 대해 에이전트를 재 수행하지 않고 결함전의 상태로 복구하여 수행효율을 높였다. 셋째, 프리패칭 기법을 이용하여 필요한 코드를 미리 각 플랫폼에 이동하여 코드의 이동 비용(네트워크 트래픽, 네트워크 지연시간)을 최소화하였다.

Ⅳ. 시뮬레이션

지금부터는 제안한 방법과 기존의 에이전트 플랫폼에서 사용한 이주서비스 2가지를 시뮬레이션을 통해서 비교 평가한다. 시뮬레이션 환경은 윈도우 2000서버 운영체제에 Intel Pentium(R) Ⅲ 1.80GHz CPU, 256MB의 메인 메모리를 갖는 시스템이 사용되었으며, AweSim 3.0 student version을 이용하여 시뮬레이션을 수행하였다.

성능테스트는 이주관점에서 테스트를 하였다. 제안하는 알고리즘과 기존의 플랫폼의 이주 성능에 대해 테스트를 했다. 그리고 제안하는 알고리즘을 프리패칭 기법, 동적 경로, 프리패칭+동적경로 모듈별로 구분하여 성능을 테스트했다. 이 실험에서 사용한 성능관련 파라미터 및 실행 파라미

터는 표 3 과 같다. 이주는 2번으로 정하여 실험을 수행했고 도착율(에이전트 수행시간, 시스템 부하)은 식 1의 지수분포를 따른다. 우리는 이 수식의 λ를 3 ~ 20 으로 변화시켜 가면서 실험을 수행하였다.

$$f(x) = \lambda e^{-\lambda x} \qquad \text{(식 1)}$$

표 3. 성능평가 파라미터
Table. 3 A performance evaluation parameter

파라미터		값
실행 파라미터	플랫폼 수	2, 4
	네트워크 상태	지수분포(λ: 0.1 ~ 0.25)
	코드 사이즈	4KB
성능 파라미터	에이전트 도착율	지수분포(λ: 3 ~ 20)
	이주 횟수	2

그림 8은 각각의 에이전트 플랫폼에서 사용하는 이주기법을 사용한 시뮬레이션 결과와 제안하는 이주기법의 시뮬레이션 결과를 보여준다. 제안하는 이주방법은 AJANTA보다 평균 85%, JAMES보다는 평균 120% 성능이 향상되었다.

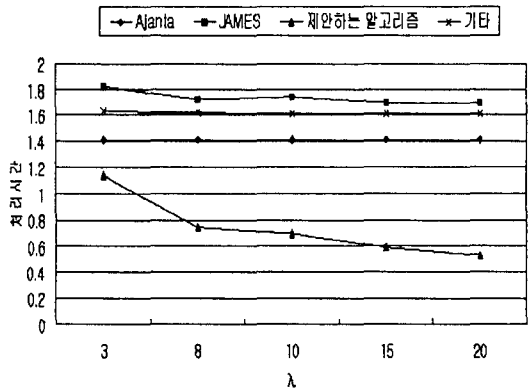


그림 8. 에이전트 도착율 vs 처리시간
Fig. 8 An arrival rate of agent vs execution time

그림 9는 제안하는 알고리즘을 모듈별로 구분하여 성능을 테스트한 결과이다. 제안하는 기능을 모두 고려한 경우, 프리패칭만 고려했을 경우, 동적 경로만 고려했을 경우의 결과이다. 제안하는 알고리즘을 모두 고려했을 경우 프리패칭만 사용한 경우보다 평균 60%, 동적 경로만을 사용한 경우보다 평균 35% 성능이 좋다.

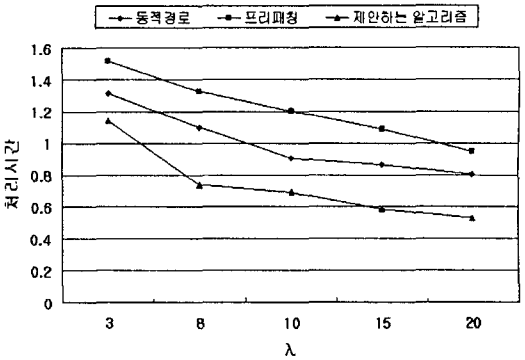


그림 9 . 에이전트 도착율 vs 처리시간
Fig. 9 An arrival rate of agent vs execution time

V. 결 론

기존의 이동 에이전트 시스템의 이주는 네트워크 트래픽 및 일관성 있는 에이전트 이주를 보장하지 못했다. 본 논문에서는 이러한 문제점을 해결하기 위해 정적, 동적 경로를 혼합한 방법, 프리패칭 및 체크포인트 기술을 사용하여 이주성능을 향상시키기 위한 기법을 제안하였다. 제안된 이주기법은 이동 데이터 크기 및 플랫폼 상태 그리고 네트워크 상태 변화에 따라 성능이 향상되었다. 향후에는 제안한 이주기법을 실질적인 에이전트 플랫폼에서 구현하여 활용될 수 있도록 할 것이다.

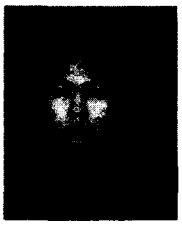
참고 문헌

[1] Kurt Rothermel and Radu Popescu-Zeletin, "Mobile Agents" , Proc. First International Workshop, 1997

[2] 조수현, 김영학, "결함 허용을 고려한 효율적인 이동 에이전트 전송방법" , 한국정보과학회 추계학술대회 논문집(III) 제28권 제2호,

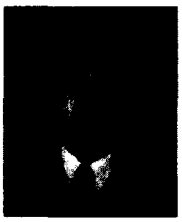
pp.550~552, 2001.
[3] Luis Moura Silva and Paulo Simoes, " JAMES : A platform of Mobile Agents for the Management of Telecommunication Networks" , Proc. International Workshop on Intelligent Agents for Telecommunication Applications, pp.76-95, 1999
[4] Luis Moura Silva, "Optimizing the Migration of Mobile Agents" , Proc. Mobile Agents for Telecommunication Applications, 1999
[5] Anand Tripathi and Tanvir Ahmed, "AJANTA " , http://www.cs.umn.edu/ajanta/
[6] Anand R. Tripathi and Neeran M. Karnik and Manish K. Vora and Tanvir Ahmed and Ram D. Singh, "Mobile Agent Programming in Ajanta" , Proc. 19th International Conference on Distributed Computing Systems, pp.190-197, 1999
[7] Mitsuru Oshima, Guenter Karjoth and Kouichi Ono, "Aglets Specification 1.1 Draft" , http://www.trl.ibm.com/aglets/spec11.htm, 1998
[8] C. BAUMER and M. BREUGST and S. CHOY and T. MAGEDANZ, "Grasshopper Programmer' s Guide" , http://www.grasshopper.de/
[9] K. Koukoumpetsos and N.Antonopoulos, " Mobility Patterns: An Alternative Approach to Mobility Management" , Proc. 6th World Multi-Conference on Systemics, pp.14-18, 2002
[10] T. Illmann, T. Krueger, F. Kargl and M. Weber, "Migration of Mobile Agents in Java : problems, Classfication and Solutions " , Proc. MAMA'00, 2000
[11] Eddy Truyen, Bert Robben, Bart Vanhaute, Tim Coninx, Wouter Joosen and Pierre Verbaeten, "Portable Support for Transparent Thread Migration in Java" , Proc. International Symposium on Agent Systems and Applications/Mobile Agents (ASA/MA'2000), pp.29-43, 2000
[12] Sara Bouchenak, Daniel Hagimont and Noel De Palma, "Techniques for Implementing Efficient Java Thread Serialization " , Proc. ACS/IEEE International Conference on Computer Systems and Applications, 2003
[13] S. Bouchenak and D. Hagimont, "Approaches to Capturing Java Threads State" , Proc. IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing, 2000
[14] Christian Erfurth, Peter Braun and Wilhelm Rossak, "Some Thoughts on Migration Intelligence for Mobile Agents" , Technical Report No. 09/01, Computer Science Department, 2001

저자 소개



김완성(Wan-Soung Kim)

1999.02 : 한밭대학교 화학공학과 공학사
2004.02 : 충북대학교 정보통신공학과 공학석사
※ 관심분야 : 에이전트, 데이터베이스 시스템, 네트워크 등



복경수(Kyoung-Soo Bok)

1998.02 : 충북대학교 수학과 이학사
2000.02 : 충북대학교 정보통신공학과 공학석사
2000.03 - 현재 : 충북대학교 정보통신공학과 박사과정
※ 관심분야 : 자료 저장 시스템, 멀티미디어 데이터베이스, 이동객체 데이터베이스, 고차원 색인 구조, 시공간 색인구조 등

신재룡(Jae-Ryong Shin)

1996.02 : 충북대학교 정보통신공학과 공학사
1998.08 : 충북대학교 정보통신공학과 공학석사
2002.08 : 충북대학교 정보통신공학과 공학박사
2003.02 - 현재 : 광주보건대학교 인터넷정보과 전임강사
※ 관심분야 : XML, 데이터베이스 시스템, 실시간 시스템, 멀티미디어 데이터베이스 등

유재수(Jae-Soo Yoo)

1989 : 전북대학교 컴퓨터공학과 공학사
1991 : 한국과학기술원 전산학과 공학석사
1995 : 한국과학기술원 전산학과 공학박사
1995 - 1996.8 : 목포대학교 전산통계학과 전임강사
1996.8 - 현재 : 충북대학교 정보통신공학과 및 컴퓨터정보통신연구소 부교수
※ 관심분야 : 데이터베이스 시스템, 정보검색, 멀티미디어 데이터베이스, 분산 객체 컴퓨팅 등