이동 에이전트를 위한 효율적인 이주 정책 설계 및 구현

전 병 국† · 최 영 근††

요 약

최근 몇 년 동안 이동 에이전트(Mobile Agent) 기술은 분산 처리 시스템(Distributed Processing System)의 새로운 패러다임(Paradigm)으로 많은 관심이 되어왔다. 이동 에이전트는 통신망 노드에서 노드로 이주 가능한 자율적인 객체이다. 그러나, 통신망에 연동된 호스트(Host)나 노드 결손 등으로 인해 이동 에이전트는 계속 처리할 수 있는 다른 노드들이 있을 지라도 무한 대기하거나 파괴될 수 있다. 이를 위해서, 본 논문은 이동 에이전트의 **이주**를 보장하기 위한 경로 **재조정과 후위** 복구 기법을 통한 효율적인 이주 정책을 제안한다. 제안된 이주 정책은 가능한 한 자율적으로 이동 에이전트의 **이주 신뢰**를 제공하고, 자바(Java) 언어로 개발된 이동 시스템 모델인 MOS에서 이를 구현한다.

Design and Implementation of an Efficient Migration Policy for Mobile Agents

Byung-Kook Jeon† · Young-Keun Choi††

ABSTRACT

Mobile agent technology has received great attention in the last years as a new paradigm for distributed processing systems. Mobile agents are autonomous objects that can migrate from node to node of a computer network. But, due to hosts or communication nodes failures, mobile agents may be blocked or crashes even if there are other nodes available that could continue processing. To cope with above, this paper proposes an efficient policy by introducing the path reordering and backward recovery to ensure the migration of mobile agents. The proposed migration policy will be provided the migration reliability of mobile agents as autonomously as possible, and it is implemented in the MOS, a mobile object system model developed by the Java language.

1. 서 론

분산 시스템(Distributed System)의 새로운 패러다임(Paradigm)인 이동 에이전트(Mobile Agent)는 통신망 점유와 과부하를 효율적으로 감소시켜 기존 분산 시스템 문제들을 해결하고 있다[1,2,3,4], 이를 이용한 응용 분야는 매우 확장성이 있어 분산 처리뿐만 아니라 통신망 관리, 전자 상거래, 병렬처리 등 매우 다양하다. 이동 에이전트는 사용자를 대신하여 자율적이며 능동적으로 통신망(Network)을 이동하면서, 망에 연동된 노드(Node)의 호스트(Host) 시스템들 특성에 맞는 강력한 연산 기능이나 데이터베이스(Database) 정보 등을 자율적으로 가공 처리하여 사용자 요구에 따른

† 준 회 원 : 광운대학교 대학원 전자계산학과
†† 정 회 원 : 광운대학교 전자계산학과 교수
논문접수 : 1999년 3월 12일, 심사완료 : 1999년 5월 31일

결과를 제공한다. 또한, 이동 에이전트는 이동을 위해 이주를 지원하는 가상 장소(Virtual Place)를 필요로 한다. 이러한 환경을 제공하는 시스템을 이동 에이전트 시스템이라 하며, 이에 대표적으로는 Aglet[3], D'Agent [4], Mole[5], Odyssey[6] 등이 있다.

그러나, 대부분의 기존 시스템들은 이동 에이전트가 통신망에 연동된 다수의 이동 에이전트 시스템들로 이주할 때 발생할 수 있는 노드 혹은 호스트의 결점에 대한 이주 보장을 거의 제공하지 않는다. 즉, 노드 결손이나 호스트 장애 또는 데이터베이스 등 정보 서비스의 부재 등이 발생할 때 이동 에이전트는 무한 대기 상태이거나 고아(Orphan) 상태 혹은 파괴되어 쓰레기(Garbage)로 처리될 수 있다[5]. 이를 위해, 본 논문은 이동 에이전트의 이주를 보장하기 위한 경로 재조정과 이주 관리 기법을 통한 효율적인 이주 정책을 제안한다. 제안된 이주 정책은 이동 에이전트의 특성인 자율적 이주 능력을 가능한 한 극대화하기 위해 노드나 호스트 결점 등에 대응하는 기법이다. 또한, 이에 대한 구현은 자바(Java)로 개발된 이동 시스템 모델인 MOS [7,8]에 적용한다.

본 논문의 구성으로 2장의 이동 에이전트 및 이동 시스템에서 이동 정책에 관한 기존 연구들을 분석하며, 3장에서는 이동 에이전트의 효율적인 이주 보장을 위한 정책을 제안한다. 4장에서는 이동 시스템 모델 MOS에서 제안된 이주 정책의 구현 사례를 보이고, 끝으로 5장에서는 앞으로의 연구 방향을 논한다.

2. 이동 에이전트와 시스템

2.1 기존 연구

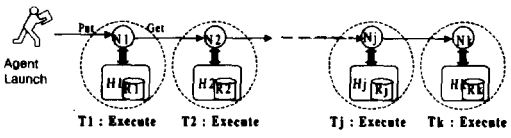
기존의 이동 에이전트 시스템들은 이동 에이전트의 이주를 지원하기 위해 이상적인 조건 즉, 시스템이나 노드에서의 결점이 없을 경우를 가정하고 있거나, 혹은 적절한 프로토콜(Protocol)을 제공하고 있다. 그러나, 만약 이동 에이전트가 적절한 라우팅(Routing) 기법[3,7,8]에 따라 해당 노드나 호스트(Host)들로 이동하고 있을 때, 이동 에이전트가 방문하고 처리하여야 할 임의의 노드가 고장 또는 접근 불가능할 경우 이동 에이전트의 이주에 대한 문제가 발생한다. 예를 들면, 노드 결점시 이동 에이전트는 해당 노드에 접근하기 위해 해당 메시지 큐(Queue)에 무한정 대기(Block)를 하거나, 이동 시스템이 있는 호스트가 장애나 쓰레기 수집을 수행하여 이동 에이전트를 파괴하는 경우가 발생할 수 있다[5]. 또한, 올바르게 이주되었다 할지라도, 이동 에이전트가 자율적으로 실행하는 과정에서 해당 호스트에서 지원하여야 할 자원중 접근할 데이터베이스 등과 같은 서비스 인터페이스(Interface)가 없을 경우 해당 에이전트는 오류를 범할 수 있다. 이러한 문제는 이동 에이전트가 목적지 노드들로 출발하여 임무를 완수한 마지막 노드까지 생명 주기(Life Span)에도 영향이 있다.

전형적으로 ORB[9]는 참조(Reference)가 없는 객체를 제거하기 위해 분산 쓰레기 수집을 수행하며, Voyager[10]는 이동 에이전트의 생명 주기를 위한 5가지의 정책을 제공한다. 또한, Mole[5]는 이동 에이전트에 대한 고아 찾기와 성공적인 종료를 위한 그림자(Shadow) 프로토콜을 제공하고 있다. 그림자 프로토콜은 이동 에이전트가 이동 중에 결점이 발생한 경우에 대해서 이를 찾아 처리하기 위한 프로토콜이며, 본 논문에서 제공하는 이동 에이전트의 이주 신뢰성을 보장하지는 않는다. 단순 프로토콜로서 트랜잭션 메시지 큐[11]를 이용한 방법이 있다. 이 방법은 메시지 전달자가 큐에 메시지를 놓고(Put) 수신자는 가져오는(Get) 방식으로서 프로세스들간의 비동기적 통신을 제공한다. 그러나 이 방법에 의하면 에이전트의 처리 과정을 감시하는 기능이 없어 자율적인 이동 에이전트가 실행시에는 문제가 생긴다. 예를 들면, 한 에이전트가 입력 큐에 있고 다음 노드의 큐로 이동되기 전에 노드 장애가 발생하면, 에이전트는 그 노드가 다운되어 복구되기 전까지 정지하고 있다. 이는 클라이언트/서버에서 생기는 문제와 다르다. 클라이언트는 서버 장애가 발생했을 때 같은 기능을 지원하는 다른 서버로 대체되어 사용자가 처리를 다시 요청할 수 있기 때문이다. 무결점(Fault Tolerance)을 지원하는 Mole[11]는 이동 에이전트를 전체 노드에 복사를 함으로써 투표(Voting)와 선택(Selection) 프로토콜을 이용하여 '정확히 한번(Exactly Once)' 이주를 위한 효율적인 기법을 제공하지만, 작업(Worker) 노드를 제외한 모든 관찰(Observer) 노드간의 상호 감시 기능과 통신이 필요할 뿐 아니라 망 접속에 결점이 없는 것을 가정하고 있다.

2.2 이동 에이전트의 수행과 결점

자율성을 가진 이동 에이전트는 라우팅 스케줄에 의한 이주를 수행함에 있어 (그림 1)과 같은 방식으로

실행한다. (그림 1)은 트랜잭션 메시지 큐를 대신한 객체 저장소를 사용하여 이동 에이전트가 수행하는 과정을 순차적으로 나타낸 것이다. 여기서 에이전트는 노드에서 노드(Na->...->Ni->..>Nk, Hi는 노드 Ni의 호스트, Ri는 에이전트 저장소(Repository))로 방문한다고 가정하자. Nk 노드를 제외하고는 모든 노드에서 트랜잭션 Ti에 대해 Get(Agent); Execute(agent); Put(agent); Commit; 같은 작업을 반복한다. 여기서 Get은 저장소에서 에이전트를 가져온다. Execute는 받은 에이전트를 수행하고, Put은 다음 방문될 저장소에 갖다 놓는다. 이러한 일련의 과정이 한 트랜잭션에서 수행되는 작업 단위이다.



(그림 1) 이동 에이전트 이주 경로

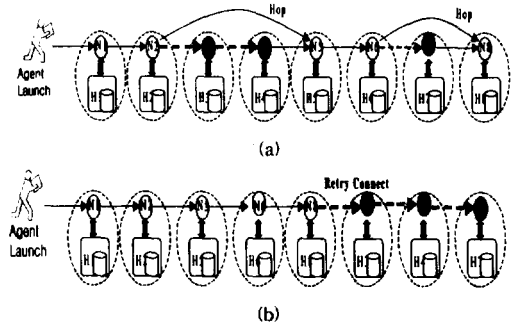
(그림 1)에서 임의 노드 Ni와 Nj간의 통신망 장애인 Nj에서 노드 결손이 발생했다고 가정하자. 그러면, 에이전트는 노드 Nj의 호스트가 동작중일 지라도 이주가 불가능한 상태가 된다. 역으로, Nj 노드가 연결중이라 할 지라도 해당 호스트에서 이동 에이전트가 시스템에 이주를 할 수 없는 상태, 즉 이동 시스템이 동작을 하지 않고 있는 상태가 있을 수 있다. 이러한 경우 자율적인 이동 에이전트는 목적지 노드까지 도달하기가 불가능해 지며, 사용자 지시를 이전 노드에서 기다릴 필요가 있다. 또한, 이동 에이전트가 경로상의 노드 결손이 없는 호스트 Hi에 이주하여 실행중일 때, 해당 시스템 Hi에 장애가 발생하여 더 이상 지원이 불가능한 경우이다. 이때, 자율성을 갖고 이주와 실행을 하던 이동 에이전트는 무한 대기 상태로 되거나, 파괴되어 올바른 목적 달성을 할 수 없다. 이같은 결점이 이동 에이전트가 방문하여야 할 이동 경로에 있을 때 이를 처리하지 못할 경우는 비효율적 운영이 될 것이다. 따라서 이동 경로상의 임의 중간 노드에서 결점이 생겨 더 이상 이동 에이전트가 대기 혹은 종료되는 것보다 가능한 한 남아 있는 나머지 경로에 대해서도 계속 수행을 처리해 주는 것이 효과적이다. 이러한 가정하에서 본 논문에서는 (그림 1)과 같은 경로를 가진 이동 에이전트의 이주 신뢰를 보장하는 정책을 3장에서 제안한다.

3. 이주 보장 정책

본 논문에서는 이동 에이전트의 이주 신뢰, 즉 호스트의 장애나 통신망 노드 결점 등이 발생할 때 이동 에이전트의 본질적 목적인 이주 보장을 위해 결점 유형에 따르는 효율적인 이주 보장 정책을 설계한다.

3.1 경로 재조정

2.2에서 가정한 것처럼 노드 결점 혹은 호스트 장애로 인해 이동 에이전트가 중간에서 이주 자체가 불가능해 지는 경우가 있다. 예를 들면 (그림 2)(a)와 같은 라우팅 스케줄에 따른 에이전트 이주 경로가 있다고 가정하자. 이동 에이전트는 노드 N1, N2에 순차적으로 이주를 하여 수행하지만 노드 N3, N4, N7이 통신상의 결손이 생겨 해당 에이전트는 더 이상 이주하지 못하고 N2에서 무한 대기를 하게 된다. 이를 방지하기 위해 결손이 생긴 노드와의 연결은 건너 띄고(Hop) 해당 노드 주소를 라우팅 스케줄상의 맨 마지막으로 이동시킨다. 이후 연결은 노드 결손이 없는 것을 먼저 수행하게 되므로 이동 에이전트의 이주 경로가 (그림 2)(b)처럼 바뀐다. 이후, 결손이 생긴 노드에 대해서만 일정 시간(Timestamp)이 경과한 후 재연결을 시도하여 이주를 수행한다. (그림 2)는 이러한 방식에 의해 이동 에이전트가 이주되는 순서가 전체적으로 재조정된 것을 보여준다. 이와 같은 방법으로 (그림 3) 알고리즘은 노드 결점 혹은 이동 에이전트가 다음의 시스템에 도달하지 못할 때 연결 경로를 자동적으로 재조정하는 이주 정책을 제공한다.



(그림 2) 이주 경로상의 노드 결점과 재조정

제안된 (그림 3)의 알고리즘은 이동 에이전트와 이동 시스템간의 통신 연결을 수행한다. 목적지 노드와

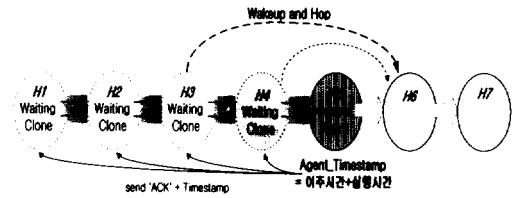
연결 설정이 안되었으면 다음 목적지 주소로 연결을 시도하고, 연결이 안된 주소를 라우팅 테이블의 맨 뒤로 이동시킨 후, 나중에 이 노드에 대해 다시 연결을 시도할 것이다. 이후, 결점이 생긴 목적지 주소들에 대해서 재연결을 시도할 때는 이동 시스템이 제공하는 Timestamp만큼 대기한 후 연결을 시도한다. 이후, 재차 실패되면 현 주소에 대한 연결은 무시하고 또 다음의 결점 노드로 연결 수행을 반복한다. 이때, 두 번 이상 실패된 노드에 대해 알고리즘 1을 계속 적용하면 이동 에이전트는 연결을 위한 무한 반복이 발생하므로 재시도 횟수를 제한하여 무시한다. 또한, 연결이 되었다 할지라도 해당 노드의 호스트에 이동 에이전트 시스템이 존재하지 않을 경우도 동일하게 적용한다.

```
for each agent's routing-table {
    extract a target address
                and fail_checked information;
    if (no more a target address)
      backward broadcast 'Agent_Fire' signal
                        to successed_target nodes;
    if (is it a fail_checked_address) {
        wait the agent during some system_timestamp;
        try to connect Socket to the address;
        if (success) { call goAgent; exit; }
        else { notify to user the address is unavailable;
               ignore the address;
        }
    }
    else if (not a fail_checked_address) {
          try to connect Socket to the destination node;
          if (success) { call goAgent; exit; }
          else { // fail to connect or send
                 notify to user;
                 move the current failed_address
                          to last in the routing-table;
                 set the fail_checked information;
          }
    }
}
```

(그림 3) 경로 재조정 알고리즘

3.2 후위 복구

이동 에이전트가 이주되어 실행 중인 상황에서 호스트 장애가 발생할 경우 즉, (그림 4)에서 노드 N5의 호스트 H5가 장애가 생기면 수행중인 에이전트들은 무한 대기 혹은 파괴된다. 이를 방지하기 위해 이동 에이전트들은 이전 노드 N4에서 실행을 끝낸 후 N5 노드로 이주할 때, 자신을 똑같이 복제(Clone)하여 다음 노드에서 'ACK' 신호가 올 때까지 무조건 대기한다. 만약 다음 노드 호스트인 H5에서 'ACK' 신호가 Timestamp내에 도착하지 않으면 N4에서 대기하고 있던 복제된 에이전트는 다음 노드 호스트 H5의 장애로 판단한 후, 자동적으로 활성화되어 N5 노드를 지나쳐 다음 노드인 N6로 건너뜀을 수행한다. 다음 호스트 H6에 이주된 에이전트도 실행중 장애가 발생하면 이 같은 작업을 반복한다. 그러나, 만약 호스트 H5에서 수행중이던 에이전트가 호스트의 장애로 인해 파괴되고, 동시에 이전 노드인 N4의 호스트 H4도 연속적으로 장애가 발생할 경우 그전 노드의 호스트인 H3에 복제되어 대기하고 있던 에이전트가 자동적으로 구동(Wakeup)되어 실행한다. 이를 후위 복구(Backward Recovery)라 한다. 이후, 최종 목적지 호스트에서는 결점 노드와 장애가 발생했던 호스트의 노드들을 제외한 경로상의 모든 복제된 이동 에이전트를 제거하는 신호(Fire)를 호스트들로 보내어 대기하고 있던 이전 복제 에이전트 모두를 제거한다. 여기서 원본 에이전트가 호스트 Hk에 있을 때 i번째 호스트 Hi에 복제 에이전트의 대기 시간 TSi는 System_Timestamp + $\sum_{x=i}^{k}$ Agent_Timestamp$_x$로 설정된다. 이와 같은 방법으로 제안된 알고리즘 2는 이동 에이전트가 호스트에서 다음 목적지 호스트로의 이주 관리를 위한 후위 복구 정책이다.



(그림 4) 후위 복구의 예

(그림 5) 알고리즘에서 waitingAgentCheck 모듈은 이동 에이전트 시스템에서 주기적으로 기다리고 있는 모든 복제된 에이전트에 대해 구동시키는 기능이다. 또한, 모듈 goAgent는 해당 에이전트 전송 후 도착했다는 'ACK' 신호가 올 때까지 기다린 후, 신호가 도착하면 복제된 에이전트를 기다리게 한다. 만약 일정 시간동안 신호가 오지 않을 경우는 호스트 장애로 간주하여 arrangePath 메소드를 수행하여 다음 노드와 연결을 수행한다. arriveAgent는 이전 노드에 'ACK' 신호를 보내고, 도착한 에이전트에 대해 실행하도록 한다. sleepAgent는 에이전트 자신의 Timestamp와 시스템에서 자원을 처리하는데 걸리는 기본 Timestamp를

합한 시간 동안만큼 복제된 에이전트를 머물게 한다. wakeupAgent는 waitingAgentCheck와 goAgent 모듈에 의해 호스트로 이주된 에이전트가 더 이상 동작할 수 없는 상황으로 판단하여 복제된 에이전트를 찾아 삭제 후 목적지 노드 경로를 재조정한 후 3.1의 경로 재조정하는 알고리즘을 수행한다. 그래서, 알고리즘 1처럼 더 이상 에이전트가 이동할 곳이 없으면 기다리고 있던 이전 노드의 모든 복제 에이전트를 제거한다.

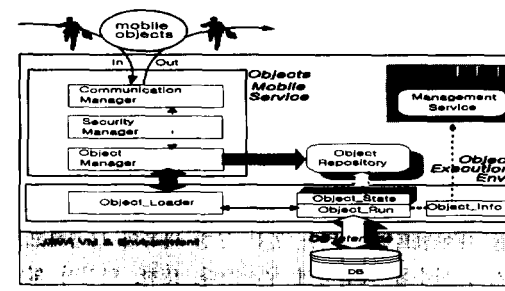
```
//주기적으로 대기 에이전트 Timestamp 검사
watingAgentCheck :
 for each sleeped_agent
    if (empty a agent_timestamp)
       notify to user;
       call wakeupAgent;

//에이전트 Go 및 복제
goAgent :
  send the agent;
  wait the agent's 'ACK' signal during
                                send_timestamp;
  if ('ACK') {
        clone the agent;
        call sleepAgent;
  }
  else call wakeupAgent;

//에이전트가 도착하면 이전노드에 알림과 실행
arriveAgent :
  send 'ACK' to the previous_node;
  start the agent;

//Timestamp만큼 복제된 에이전트 대기
sleepAgent :
 for each cloned_agent
    add agent_timstamp to system_timestamp;
    add the agent to the sleeped_list;
    sleep the agent;

//대기중인 에이전트 재구동
wakeupAgent :
 for the sleeped_list
   find a cloned_agent;
 remove it from the sleep_list;
 if (Agent_Fire) remove the cloned_agent;
 else { move the current failed_address
           to last in the routing-table;
        set the fail_checked information;
        call arrangePath;
      }
```

(그림 5) 후위 복구 알고리즘

4. 구 현

4.1 구현 환경

제안된 정책을 구현하기 위해 이미 개발된 이동 시스템 모델로서 MOS[7,8]를 사용한다. MOS는 자바 이동 객체 모델을 대상으로 입/출 기능이 있는 추상적인 장소를 제공하는 환경이다. 즉, MOS는 이동 에이전트가 목적지 노드로 이동되어 실행될 때 해당 노드에서의 시스템 성능과 데이터베이스 자원 등을 접근할 수 있는 환경으로 (그림 6)과 같은 구조로 되어 있다.



(그림 6) 이동 에이전트 시스템 모델

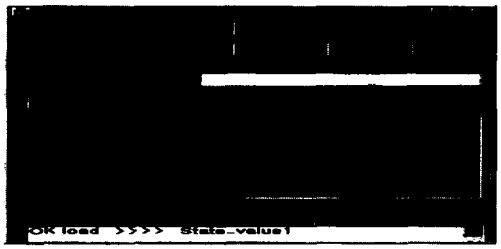
제안된 시스템 모델은 (그림 6)처럼 사용자 인터페이스를 제공하기 위한 GUI 서비스 모듈과 통신망을 통해 객체의 이동을 제공하는 객체 이동 지원(Objects Mobile Service) 모듈, 그리고 이동되어진 객체들의 등록 관리 및 자원 제공, 객체 자동 실행을 수행하는 객체 실행 환경(Objects Execution Environment) 모듈로 구성되어 있다. 또한, 객체 이동의 투명성을 위해 객체 저장소(Object Repository)를 구현해 객체 이름 투명성(Naming Transparency)을 제공한다. 이같은 시스템에서 이동 에이전트를 수행한 실제 인트라넷상에 연동된 이동 시스템들의 각 호스트의 주소와 운영체제는 <표 1>과 같다.

〈표 1〉 호스트 주소와 운영체제

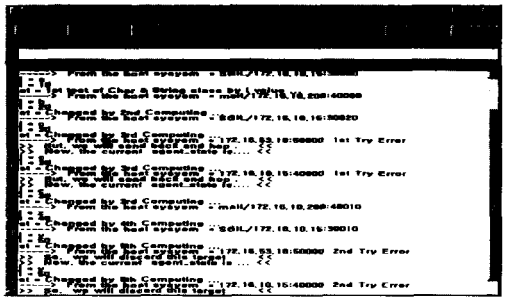
IP 주소	Port #	운영체제	Host 명
172.16.10.200	40000	Unix	mail
	40010		
172.16.10.15	30000	Unix	SGIL
	30010		
	30020		
172.16.53.18	30000	Windows NT	web-server

4.2 구현 실례

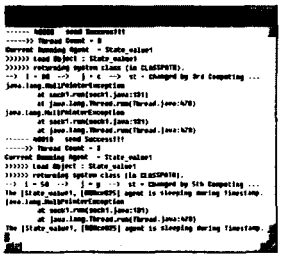
다음의 그림들은 단순한 분산 처리 작업을 수행하는 이동 에이전트를 모델로 적용된 결과 화면이다. (그림 7)은 사용자가 에이전트를 가져오고, 라우팅 스케줄을 명시한 것이다. 여기서 결점 처리를 위해 2개의 잘못된 목적지가 있다. (그림 8)은 이주 결점을 자동적으로 관리한 것을 보여준다.



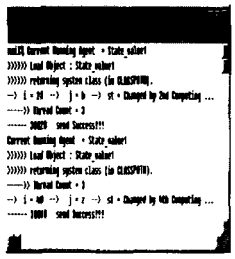
(그림 7) 에이전트 적재



(a) 사용자 결과 화면



(b) 서버 SGIL의 수행



(c) 서버 Mail의 수행

(그림 8) 실행 결과 화면

(그림 7)처럼 에이전트의 적재 작업과 라우팅 스케줄을 결정하면 (그림 8)(a) 사용자 창에서 'Go' 명령을 실행한다. (그림 8)(a)와 (c)에 나타난 바대로 노드 결점이 없는 것은 올바른 수행을 하였으며, 노드 결손 혹은 이동 시스템이 동작하지 않는 노드에 대해서는 (그림 8)(a)와 (b)처럼 1차와 2차 연결 설정을 수행한 결과를 보여준다. (그림 8)의 (b), (c)는 이동 에이전트가 해당 시스템들로 이동해서 실행중인 과정을 화면 캡처한 것이며, (그림 8)의 (b)에서 보여진 에러 메시지는 실제 결점을 나타내기 위해 보여준 것이다. 이때, (그림 8)의 (b)는 해당 에이전트 쓰레드가 Timestamp 만큼 재연결을 위해 대기하고 있음을 나타내고 있으며, 대기하고 있는 에이전트에 관한 상태도 결손에 따른 영향이 없는 것을 보이기 위해 해당 에이전트의 현 상태 정보를 매번 출력한 것이다.

5. 결 론

본 논문에서는 이동 에이전트가 방문하고 처리하여야 할 임의의 노드가 고장 또는 접근 불가능할 경우 자율적인 이동 에이전트의 이주 신뢰에 대한 문제 해결을 위해 Timestamp를 이용한 경로 재조정과 후위 복구 정책을 설계하고 구현하였다. 구현된 알고리즘은 노드 결손 혹은 이동 시스템 장애시 이동 에이전트의 본질인 자율적 이동성을 보장해 줌으로써 무한 대기나 파괴되는 것을 회피하였을 뿐 아니라 에이전트 생명 주기도 가능한 한 향상시켰다. 또한 부수적으로 망 결손이나 호스트 장애를 자동 발견하는 효과도 제공한다. 향후 연구 방향으로는 협력 작업을 위한 그룹 에이전트 처리를 할 때 동일한 이주 보장 기법의 보완이 필요하며, 향후 전자 상거래나 분산 처리, 통신망 관리 등 이동 에이전트를 이용한 응용 분야 개발이 필요하다.

참 고 문 헌

[1] K.A. Baharat, L. Cardelli, "Migratory Applications," Proc. of the 8th Annual ACM Symp. on UIS Tech., November 1995.

[2] J. Vitek and Christian Tschudin, "Mobile Object Systems: Towards the Programmable Internet," Springer-Verlag, April 1997.

[3] IBM, "The Aglets Workbench." URL: http://www.trl.ibm.co.jp/aglets/

[4] Robert S.G, "Agent Tcl: A flexible and secure mobile-agent system," TR98-327, Dartmouth Col. June 1997.

[5] J. Baumann, "A Protocol for Orphan Detection and Termination in Mobile Agent Systems," TR-1997-09, Stuttgart Univ. July, 1997.

[6] General Magic, "Odyssey," URL : http://www.genmagic.com/agents/

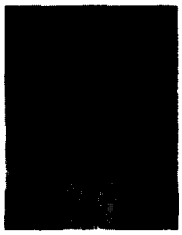
[7] 전병국, 이근상, 최영근, "Java언어를 이용한 객체 이동시스템의 설계 및 구현", 정보처리논문지, 6권, 1호, Jan. 1999.

[8] 전병국, 최영근, "인트라넷상에서 자바 객체의 이동시스템 설계 및 구현", 정보과학회논문지(C), 5권 2호, Arp., 1999.

[9] OMG, "Mobile Agent Facility Interoperability Facilities Specification(MAF)," OMG.

[10] ObjectSpace Voyager, GeneralMagic Odyssey, IBM Aglets: A Comparison, June, 1997.

[11] K. Rothermel, M. Strasser. "A Fault-Tolerant Protocol for Providing the Exactly-Once Property of Mobile Agents," Proc. 17th IEEE SRDS'98, Oct. 1998.



전 병 국

e-mail : jeonbk@sky.wonju.ac.kr

1985년 광운대학교 전산과 졸업(이학사)

1991년 광운대학교 대학원 전산과 졸업(이학석사)

1991년~1993년 KINITI 연구원

1993년~현재 원주대학교 사무자동화과 조교수

1992년~현재 광운대학교 대학원 전자계산학과 박사과정

관심분야 : 이동에이전트, 분산처리, 전자상거래

최 영 근

e-mail : ygchoi@daisy.kwangwoon.ac.kr

1980년 서울대학교 사범대학 수학교육과(이학사)

1982년 서울대학교 대학원 계산통계학과(이학석사)

1989년 서울대학교 대학원 계산통계학과(이학박사)

1983년~현재 광운대학교 전자계산학과 교수

1997년~현재 광운대학교 전자계산원 원장

관심분야 : 병렬 컴파일러, 병렬 프로그래밍 언어, 객체지향 프로그래밍 언어, 객체지향 분산 컴퓨팅