제19회 한국정보처리학회 춘계학술발표대회 논문집 제10권 제1호 (2003.5)

이동 에이전트를 위한 이주 스케줄 기반의 이주 기법 설계

김구수*, 엄영익*
*성균관대학교 정보통신공학부
e-mail : {gusukim,yieom}@ece.skku.ac.kr

Design of Itinerary Based Migration Scheme with Fault Tolerance for Mobile Agent

Gu Su Kim*, Young Ik Eom*
*School of Information and Communication,
Sungkyunkwan University

요 약

이동 에이전트란 컴퓨터 네트워크 상에서 사용자를 대신하여 특정 작업을 수행하는 프로그램이 독자적으로 여러 노드들을 이동하면서 필요한 작업을 수행하고 그 결과를 사용자에게 전달하도록 작성된 프로그램을 말하며, 이런 이동 에이전트를 수행할 수 있도록 컴퓨팅 환경을 제공하는 것을 이동 에이전트 컴퓨팅 환경이라고 한다. 본 연구에서는 이동 에이전트 운영에 필수 기능인 이동 에이전트 이주 기능을 이주 시작 이전에 이주 경로를 설정하여 이주 스케줄(Itinerary)로 관리하는 기법과 이동 에이전트가 이주할 시점에서 자율적으로 이주 목적지를 선정하고 선정된 목적지의 정보를 홈에 있는 Itinerary에 저장하여 관리하는 기법을 설계하였다. 그리고 이주한 방문지에서 수행 결과를 체크포인트로 저장하여 예외나 비정상적인 종료를 하였을 때 Itinerary의 내용을 보고 성공적으로 수행한 가장 마지막 플랫폼에게 이동 에이전트의 복원 및 다른 목적지로의 이주를 요청함으로서 결함 허용(fault-tolerance) 기능을 가지도록 설계하였다.

1. 서론

이동 에이전트란 컴퓨터 네트워크 상에서 사용자를 대신하여 특정 작업을 수행하는 프로그램이 독자적으로 여러 노드들을 이동하면서 필요한 작업을 수행하고 그 결과를 사용자에게 전달하도록 구성된 프로그램을 말한다[1]. 이동 에이전트 컴퓨팅 환경이란 이동 에이전트들이 각 노드들을 이주하면서 필요한 작업을 수행할 수 있도록 지원하는 프레임워크 구성, 운영하고 있는 컴퓨팅 환경을 말한다. 대표적인 이동 에이전트 플랫폼(mobile agent platform)으로는 Aglets, D'Agent, Tacoma, Voyager, Odyssey, Mole, Concordia 등이 있다[2]. 이들 플랫폼은 개발 언어에 따라 크게 자바 기반의 플랫폼과 비자바 기반의 플랫폼으로 나눌 수 있다. 이들 플랫폼은 개발 언어에 관계없이 이동 에이전트를 지원하기 위해 필수적으로 필요한 이동 에이전트의 이주 기능을 제공하고 있다.

2. 기존 연구

이동 에이전트의 이주 기법은 에이전트의 실행 상태를 어느 수준까지 이주시키느냐에 따라 strong 이주와 weak 이주 두 가지로 나눌 수 있다[3]. strong 이주 기법은 이주할 이동 에이전트의 실행 코드, 멤버 데이터, 프로그램 카운터, 스레드 스택, 리소스 등 실행 당시의 모든 정보와 자원을 이주시킨다. 반면에 weak 이주 기법은 이주할 이동 에이전트의 실행코드와 이동 에이전트의 멤버 데이터 만을 이주시킨다. 기존의 플랫폼들 중에서 Telescript, Ara와 같은 스크립트 기반의 플랫폼이나 Planet과 같은 오픈 소스 운영체제의 커널을 수정하여 만든 플랫폼에서는 strong 이주 기법을 구현한 사례가 있으나, Aglets, Voyager, Odyssey 등과 같은 자바 기반의 플랫폼에서는 strong 이주 기법을 지원하지 않고 weak 이주 기법만을 제공한다. 이는 자바가 스레드의 스택을 처리하는 API를 제공하지 않는데 기인한다. 하지만 자바 기반의 플랫폼 중에서 NOMADAS, WASP에서는 JVM의 소스를 수정하여 스레드의 스택에 접근하는 API를 만들어서 strong 이주 기법을 지원하고 있다[4].

이동 에이전트는 여러 플랫폼으로 이주하며 실행하게 되는데, 이주할 때 이주할 목적지의 선정을 어

떻게 할 것인지에 대한 기법이 필요하다. 이동 에이전트가 자율적으로 목적지를 결정할 수 도 있고, 이동 에이전트 개발자가 이동 에이전트를 개발할 때 직접 지정할 수 도 있다. 또 다른 방법으로 이동 에이전트가 이주할 목적지들의 전체 경로를 미리 지정하여 관리할 수 있는데 이주할 목적지의 스케줄에 관한 정보를 관리하는 객체를 Itinerary라고 한다[5]. 현재 많은 이동 에이전트 플랫폼들이 Itinerary를 사용하여 이동 에이전트를 이주시킨다.

이동 에이전트는 이주한 플랫폼에서 실행 중에 예외를 발생시키거나, 비정상적으로 종료할 수 있다. 이 경우 가능하다면 이동 에이전트를 종료 이전의 상태로 복원하여 다른 목적지로 이주시켜 실행을 계속할 필요가 있다. 비정상적 종료 상태에서 이동 에이전트의 실행을 복원하기위해서는 정기적으로 이동 에이전트의 실행 이미지를 저장해야한다. 플랫폼에서 저장하는 이동 에이전트의 실행 이미지를 체크포인트라고 한다[5].

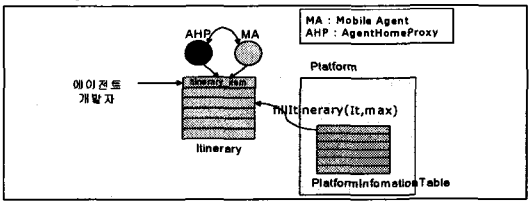
본 논문에서는 자바 기반의 이동 에이전트 플랫폼에서 이동 에이전트의 이주 경로를 관리하기 위해 Itinerary를 사용한 이주 기법을 설계하였다. 이주 경로를 사전에 모두 정하여 정해진 경로로 이동 에이전트를 이주 시키는 기법과 사전에 이주 경로를 결정하지 않고, 이동 에이전트가 이주 시점에서 자율적으로 이주 목적지를 결정하여 이주하는 기법을 설계하였다. 두 방법 모두 Itinerary를 이용하며, 체크포인트를 사용하여 결함 허용(fault-tolerance) 기능을 가지도록 하였다.

3. Itinerary 기반의 weak 이주 기법 설계

이동 에이전트가 이주를 할 때 어디로 이주할지 목적지를 결정해야한다. 이동 에이전트의 작업 성격에 따라 어떤 작업은 사전에 방문할 목적지들을 미리 정할 수 있다. 다른 작업에서는 이주 경로를 미리 설정하지 않고 이주 당시에 결정할 수 도 있다. 각각의 경우에 따라 Itinerary를 다르게 관리해야 한다. 본 논문에서 이 두 경우를 위한 이주 기법을 제시한다.

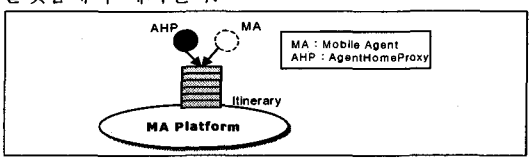
3.1 사전 이주 경로 결정에 의한 이주 기법

이동 에이전트의 이주 경로를 이주가 시작되기 전에 설정하기 위해 이동 에이전트는 홈에서 앞으로 방문할 목적지를 Itinerary 객체에 저장해야한다. Itinerary 객체의 한 아이템(Itinerary_Item)에는 한 개의 방문지에 대한 정보가 저장된다. 이 아이템에는 방문할 목적지, 방문한 후 수행할 함수, 방문 여부, 실행 성공 여부에 대한 정보가 유지된다. Itinerary 객체는 이동 에이전트가 현재 어디에서 수행하고 있는지를 가리키는 Itinerary_Item의 인덱스를 가진다. 이 인덱스로 이동 에이전트의 현재 위치를 알 수 있다. Itnerary에 아이템을 추가는 작업은 이동 에이전트 개발자가 이동 에이전트를 개발할 때 소스에서 직접 추가 하거나, 플랫폼의 fillItinerary(Itinerary it, int max) API를 호출하여 추가할 수 있다. fillItinerary API는 플랫폼이 유지하고 있는 다른 플랫폼에 대한 정보 테이블(PlatformInfomationTable)을 참조하여 최대 max개 만큼의 방문지를 결정하고 선정된 플랫폼에 대한 주소를 Itinerary에 채운다. PlatformInformationTable에 사용자가 요구한 max개 만큼의 플랫폼 정보가 없다면 fillItinerary API는 실제로 채워진 Itinerary_Item의 개수를 반환한다[그림 1].



[그림 1] Itinerary의 작성

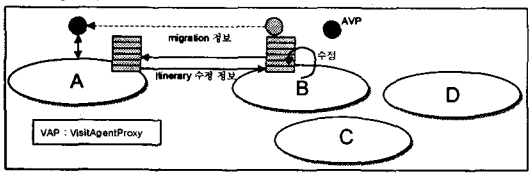
Itinerary 관리 및 보안을 위해 이동 에이전트 프락시를 사용한다. 일반적으로 프락시는 이동 에이전트가 다른 이동 에이전트의 public 메소드를 바로 호출하지 못하도록 차단하고 이동 에이전트 간 통신을 위해 사용된다. 본 기법에서는 일반적인 프락시의 기능에 Itinerary를 관리하는 기능을 추가한 AHP(AgentHomeProxy)를 두도록 하였다. AHP는 홈 플랫폼에서만 생성되며 이동 에이전트와 함께 Itinerary를 관리한다[그림 2]. AHP는 이동 에이전트가 다른 플랫폼으로 이주해가더라도 삭제되지 않고 홈 플랫폼에서 계속 존재하면서 Itinerary를 관리하여 이동 에이전트의 경로를 추적한다. 이동 에이전트는 홈에서 다른 플랫폼으로 이주할 때 Itinerary를 복사하여 Itinerary를 가지고 이주한다. 이동 에이전트가 방문한 플랫폼에서는 Itinerary 관리를 제외한 이동 에이전트 보호와 이동 에이전트 간 통신을 담당하는 AVP(AgentVisitProxy)가 생성된다. AVP는 이동 에이전트가 다시 다른 플랫폼으로 이주해갈 때 플랫폼에서 제거된다.



[그림2] Itinerary의 관리

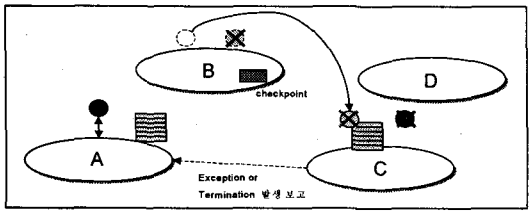
이동 에이전트가 Itinerary에서 지정한 다음 목적지로 이주했을 때 이주가 완료되면 이동 에이전트는 홈에 있는 AHP에게 이전 목적지에서 작업을 성공적으로 수행을 했는지 여부와 새로운 목적지로 성공적으로 이주했다는 사실을 AHP에게 통보한다. 그리고 Itinerary에 지정된 해당 목적지에서 수행할 함수를 호출한다. 필요하다면 새로운 플랫폼에서 Itinerary의 아이템을 추가할 수 있다. 이동 에이전트는 추가된 Itinerary의 아이템 정보들을 홈에 있는 AHP에게 보낸다. 이동 에이전트로부터 이주 완료 보고를 받은 AHP는 Itinerary의 해당 레코드를 갱신한다. 보고의 내용 중에 Itinerary 아이템 변경에 대한 정보가 있다면 AHP는 자신이 가지고 있는 Itinerary에 이들 아이템을 추가한다. 이런 과정을 통해 홈에서는 홈에서 생성된 이동 에이전트들이 현

재 어느 플랫폼에서 실행하고 있는지 위치를 추적할 수 있고, 언제라도 다시 홈으로 되돌아오게 할 수 있다 [그림 3].



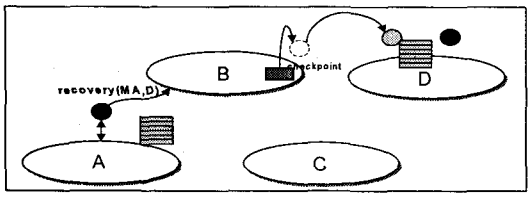
[그림 3] 이주 후 홈으로 통보

이동 에이전트는 다른 플랫폼으로 이주해갈 때 먼저 자신의 실행 상태를 체크포인트로 저장한다. 이 체크포인트는 이동 에이전트가 예외를 발생시켰거나 비정상적으로 종료했을 때 이동 에이전트를 복원하기위해 사용된다. 이동 에이전트의 복원은 AHP의 요구에 의해 이루어지며 복원된 이동 에이전트는 지정된 다음 목적지로 이주하게 된다. 이러한 체크포인트의 운영으로 이동 에이전트가 결함 허용 기능을 가지도록 하였다. [그림 4]에서 이동 에이전트는 플랫폼 B에서 플랫폼 C로 이주한다. 플랫폼 B에서 생성된 AVP는 제거되고 플랫폼 C로 이주하기 전에 이동 에이전트의 체크포인트를 플랫폼 B에 저장한다. 플랫폼 B는 후에 AHP로부터 삭제 요청이 올 때까지 이동 에이전트의 체크포인트를 유지하고 있어야 한다. 이동 에이전트가 플랫폼 C로 이주한 후 실행을 하다가 예외나 비정상적 종료를 발생하면 플랫폼 C는 이동 에이전트의 AHP에게 통보한다.



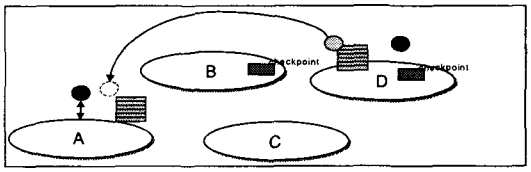
[그림 4] 예외발생을 에이전트의 홈에 보고

[그림 5]에서 예외 발생이나 비정상적 종료 메시지를 받은 AHP는 Itinerary를 조사하여 이동 에이전트가 성공적으로 수행한 마지막 플랫폼(B)과 다음 방문 목적지(D)를 찾아낸다. AHP는 플랫폼 B에게 recovery(MA, D) 메시지를 보낸다. 여기서 MA는 복원할 이동 에이전트의 ID이며 D는 이동 에이전트를 복원하여 이주시킬 목적지가 된다. 이 메시지를 받은 플랫폼 B는 해당 이동 에이전트를 복원하여 플랫폼 D로 이주시킨다.



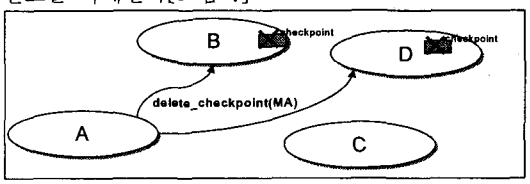
[그림 5] 이동 에이전트 복원 및 이주

이동 에이전트는 Itinerary상에 있는 모든 방문지를 방문하였거나 소유자의 복귀 명령에 의해 자신의 홈으로 복귀한다 [그림 6].



[그림 6] 수행 완료 후 홈으로 복귀

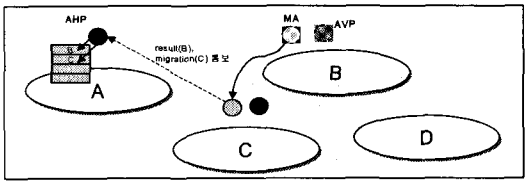
홈으로 복귀한 이동 에이전트는 Itinerary상에 성공적으로 수행을 마친 모든 플랫폼에게 즉, 체크포인트를 저장하고 있는 모든 플랫폼들에게 delete_checkpoint(MA) 메시지를 보낸다. 이 메시지를 받은 플랫폼들은 지정한 이동 에이전트의 체크포인트를 삭제한다[그림 7]



[그림 7] 체크포인트 삭제

3.2 이동 에이전트의 자율적 목적지 결정에 의한 이주 기법

이동 에이전트의 이주 경로를 항상 미리 설정하여 이주할 필요는 없다. 이동 에이전트가 다른 플랫폼으로 이주하려 할 때 이동 에이전트는 자율적으로 목적지를 선정하여 이주할 수 있다. 이 경우 AHP만 Itinerary를 가지고 이동 에이전트 자신은 Itinerary를 가지지 않는다. 초기에 이동 에이전트가 홈에 있을 때 AHP는 비어 있는 Itinerary를 가진다. 이동 에이전트가 홈을 떠나 다른 플랫폼으로 이주하게되면 최초의 Itinerary_Item이 하나 추가된다. 이동 에이전트가 다시 다른 플랫폼으로 이주하려 할때 이주할 목적지를 플랫폼에서 제공하는 플랫폼 정보를 보고 목적지를 선정하고 현재의 실행 이미지를 체크포인트로 저장한 후 선정된 목적지로 이주한다. 이동 에이전트가 이주를 완료하면 AHP에게 이전 플랫폼에서의 수행 성공 여부와 새로운 플랫폼으로 이주한 사실을 통보한다. 통보 메시지를 받은 AHP는 이전 플랫폼에서의 수행 성공 여부와 새로운 플랫폼에 대한 정보를 Itinerary에 추가한다. 이 과정을 이동 에이전트가 다른 플랫폼으로 이주할 때마다 수행함으로서 HAP가 관리하는 Itinerary에 이동 에이전트가 이주한 이주 경로들과 수행 성공 여부에 대한 정보들이 저장된다.



[그림 8] 이동 에이전트의 자율적 목적지 결정시에 Itinerary 갱신

이동 에이전트가 특정 플랫폼에서 예외나 비정상적인 종료를 발생시켰다면 이 사실을 HAP에게 통보한다. 이 통보 메시지를 받은 HAP는 Itinerary에 해당 플랫폼을 찾아 에러 발생 사실을 체크하고 성공적으로 수행한 가장 마지막 플랫폼을 찾아 이동 recovery(MA, failsites) 메시지를 보내어 에이전트의 복원을 요청한다. 복원 요청 메시지를 받은 플랫폼은 지정한 이동 에이전트를 복원하여 비정상적 종료를 발생한 플랫폼(failsites) 이외의 새로운 플랫폼을 선정하여 이주시킨다.

이동 에이전트 개발자가 이동 에이전트를 개발할 때 3.1의 방법과 3.2의 방법을 선택적으로 사용할 수 있으므로 이동 에이전트의 이주 정책 수립에 유연성을 가지도록 설계하였다.

3.3 이주 기법 비교

[표 1]에서 본 논문에서 제시하는 사전 이주 경로 결정에 의한 이주 기법과 이동 에이전트의 자율적 목적지 결정에 의한 이주 기법을 비교하였다. 사전 이주 경로 결정에 의한 이주 기법의 가장 큰 특징은 전체 이주 경로를 알 수 있고 이동 에이전트의 현재 위치를 파악하여 전체 진행 사항을 파악할 수 있다. 이동 에이전트의 자율적 목적지 결정에 의한 이주 기법의 특징은 이동 에이전트가 가장 최신의 플랫폼 정보를 이용하여 이주 목적지를 자율적으로 선택할 수 있도록 한 것이다.

	사전 이주 경로 결정에 의한 이주 기법	이동 에이전트의 자율적 목적지 결정에 의한 이주 기법
장점	• 전체 이주 경로를 파악할 수 있다. • 결함 허용 기능을 가진다. • 이동 에이전트의 위치를 추적할 수 있다.	• 이동 에이전트의 자율성이 높다. • 결함 허용 기능을 가진다. • 가장 최신의 플랫폼 정보를 이용할 수 있다. • 이동 에이전트의 위치를 추적할 수 있다.
단점	• 이동 에이전트의 자율성이 낮다. • 이주 경로가 고정되어 있다. • 이주 할 때마다 AHP에게 보고해야한다. • 실행 이미지를 홈 이외의 플랫폼에 저장하므로 보안 문제를 가진다.	• 이동 에이전트가 어디로 이주할지 예측할 수 없다. • 이주 할 때마다 AHP에게 보고해야한다. • 실행 이미지를 홈 이외의 플랫폼에 저장하므로 보안 문제를 가진다.

[표 1] 제시 기법의 장단점 비교

[표 2]에서 자바 기반의 대표적인 플랫폼인 Aglets, Odyssey, Voyager, Concordia의 이동 에이전트 이주 기법과 본 논문에서 제시한 이주 기법을 비교하였다. 각 플랫폼의 이주 기법을 Itinerary로 목적지를 미리 결정하는지, 에이전트가 자율적으로 목적지를 선정할 수 있는지, 이동 에이전트의 위치 추적(trace)이 가능한지, 프락시를 사용하는지, 체크포인트를 운영하는지, 그리고 이주 후 실행할 함수(entry point)가 한 개로 고정되어있는지 목적지 별로 다르게 부여할 수 있는지 비교하였다. 대부분의 플랫폼에서 Itinerary를 운영하고 있으며 이동 에이전트를 개발 할 때 Itinerary를 구성하도록 하고 있다. 본 논문에서 제시하고 있는 이동 에이전트가 이주할 당시에 자율적으로 목적지를 결정하고 이를 홈에 있는 Itinerary에 보관하는 기법은 다른 플랫폼에서는 없는 기능이다.

플랫폼 / 항목	Aglets	Odyssey	Voyager	Concordia	제안 기법
Itinerary로 목적지를 미리 결정	O	O	O	O	O
agent가 목적지를 자율적으로 결정	X	X	X	X	O
agent tracking	O	X	O	O	O
proxy 사용	O	X	O	X	O
checkpoint	X	X	O	O	O
entry point	one	one	multiple	multiple	multiple

[표 2] 이동 에이전트 플랫폼 별 이주 기법 비교

4. 결론 및 향후 계획

본 논문에서 이동 에이전트의 이주 경로를 사전에 결정하여 Itinerary로 관리하는 기법과 이동 에이전트가 자율적으로 목적지를 선정하여 이주하면서 AHP가 이주 경로를 Itinerary에 저장하는 기법을 설계하였다. 두 가지 이주 기법을 선택적으로 사용할 수 있도록 유연하게 설계하였다. 이동 에이전트는 이주 완료 시에 자신의 방문 여부를 AHP에게 보고함으로서 홈 플랫폼은 자신이 생성한 이동 에이전트들이 현재 어느 플랫폼에서 실행하고 있는지 추적할 수 있다. 이동 에이전트가 각 방문지에서 다시 다른 플랫폼으로 이주를 할 때 체크포인트를 저장함으로서 이동 에이전트가 비정상적인 종료가 발생하더라도 Itinerary의 이주 경로를 추적하여 종료 전에 방문한 플랫폼의 체크포인트를 이용하여 이동 에이전트를 복원할 수 있다. 하지만 이동 에이전트의 실행 이미지를 홈이 아니 다른 플랫폼에 저장함으로서 보안 문제에는 취약하다. 향후에 이러한 보안 문제를 해결 할 수 있는 기법에 대한 연구가 필요하다.

참고문헌

[1] V. A. Pham and A. Karmouch, "Mobile Software Agents: An Overview," IEEE Communications Magazine, Jul. 1998.

[2] N. M. Karnik and A. R. Tripathi, "Design Issues in Mobile-Agent Programming Systems", IEEE Concurrency, Jul.-Sep. 1998.

[3] T. Illmann, T. Krueger, F. Kargl, and M. Weber, "Transparent Migration of Mobile Agent Using the Java Platform Debugger Architecture," Mobile agents 5th International Conference, MA 2001, Atlanta, GA, USA, December 2-4, 2001 proceedings / Gian Petro Picco (ed.), LNCS 2240, p198-212

[4] "The Mobile Agent List," http://mole.informatik.uni-stuttgart.de/mal/preview/preview.html

[5] R. Broos, B. Dillenseger, P. Dini, T. Hong, A. Leichsenring, M. Leith, E. Malville, M. Nietfeld, K. Sadi, and M. Zell, "Mobile Agent Platform Assessment Report," MIAMI Project, http://www.fokus.gmd.de/research/cc/ecco/climate/ap-documents/miami-agplatf.pdf