이형질의 다중 에이전트 환경에서의 에이전트 동적 구동 기법

장지훈°, 강기영, 최중민
한양대학교 전자계산학과

Dynamic Agents Invocation in Heterogeneous Multi-Agent Environments

Jihoon Jang, Kiyoung Kang, Joongmin Choi
Dept. of Computer Science and Engineering, Hanyang University

요 약

최근 다중 에이전트에 대한 연구가 활발히 진행되고 있다. 특히 이질적이고 분산된 컴퓨팅 환경에 효율적이며 적응성이 향상된 여러 가지 형태의 다중 에이전트가 개발되고 있다. 하지만 아직 이질적인 에이전트의 동적구동을 통한 효율 향상과 유연성에 대한 연구는 초보단계이다.

본 논문에서는 자바로 된 다중 에이전트 기반 구조에서 서로 각기 다른 컴퓨터 언어로 만들어진 에이전트가 분산된 환경에서 상호 협력을 통해 작업을 수행하기 위해서 사용자나 다른 에이전트의 요청으로 동적으로 수행될 수 있는 기술을 설계하고 구현하였다. 또한 서로 다른 에이전트간의 협력을 가능하게 하는 통신기능을 다중 에이전트 기반기술에 적용함으로써 사용자의 요구에 더욱더 능동적으로 대처하고, 다양한 환경 변화에 강한 적응성과 개선된 협동능력을 제공하도록 구현하였다.

1. 서론

다중 에이전트 시스템에 있어 에이전트란 '분산 환경에서 상호 협력을 통해 작업을 수행하는 컴퓨터 프로그램'을 말한다. 이러한 다중 에이전트 시스템의 에이전트가 갖는 가장 큰 장점은 독립적인 응용 프로그램의 집합으로는 해결할 수 없는 보다 복잡한 서비스를 다른 에이전트와의 협력을 통해서 제공할 수 있다는 점이다. 그 이외에도 사용자는 컴퓨터에 있는 많은 에이전트의 존재나 사용법을 모르더라도 주로 이용하는 에이전트를 통해 자신도 모르게 다른 에이전트를 사용한다는 장점이 있다[1]. 그런데 이렇게 하나의 에이전트가 요청된 작업을 수행하기 위해서는 다른 에이전트의 도움이 필요한 경우가 있다. 다른 에이전트의 도움을 받기 위해서는 우선 해당되는 에이전트가 시스템 상에 구동되어 있어야만 하며, 구동된 에이전트는 AMS(Agent Management System)에 등록이 되며 등록된 에이전트가 제공하는 서비스에 대한 정보를 DF(Directory Facilitator)가 유지하게 된다. 그런데 대부분의 자바로 구현된 다중 에이전트 시스템에서는 자바이외의 다른 언어로 작성된 에이전트를 자동으로 구동시키지 못하기 때문에 이러한 여러 가지 일들을 수행하는 에이전트들을 일일이 사용자가 구동을 시킬 수밖에 없다. 많은 수의 에이전트에 대해서 이러한 사용자의 잦은 번거로움을 수반해야 한다. 그리고 다중 에이전트 시스템이 효율적으로 컴퓨터 시스템의 자원을 사용하기 위해서도 동적으로 해당되는 에이전트를 구동시키고 종료시키는 기능이 필수적이라 할 수 있다.

본 시스템에서는 다중 에이전트 시스템을 어느 플랫폼에서나 수정 없이 동작할 수 있도록 하기 위해서 자바 언어를 사용하여 구현하였으며, Java Native Interface를 이용하여 이질적인 에이전트의 자동 구동을 구현하였다. 또한 이질적인 에이전트들간의 통신을 위해서 KQML을 통신을 위한 언어로 사용하였으며, 에이전트 표준화 단체인 FIPA(Foundation for Intelligent Physical Agents)의 다중 에이전트 시스템의 표준을 수용하고 있다[5][6].

본 논문은 다음과 같이 구성된다. 2장에서는 기존에 구현되어 있는 시스템의 구조와 특징 그리고 각 시스템이 가지는 문제점들을 분석하고 이에 대한 해결책으로 제시된 본 시스템을 통해 보완하고자 하는 특성을 기술하였다. 3장에서는 본 시스템의 구체적인 설계와 구조에 대해 기술하였다. 4장에서는 실제적으로 구현된 프로그램을 바탕으로 실제적인 수행과정을 설명하였다. 5장에서는 결론과 앞으로의 방향을 제시한다.

2. 관련 연구

2.1 JATLite

Stanford Univ.에서 제작한 에이전트 기반구조 개발을 위한 자바 패키지로서, 자바 언어를 사용하며, KQML을 이용한 메시지 교환방식의 통신 툴을 제공한다. 또한 기반구조상에서 에이전트들간의 협력작업을 조정하기 위한 에이전트 라우터(Router)를 제공한다. 라우터를 통하여 등록된 에이전트들간의 전송하고자 하는 메시지를 메시지 큐에 저장하여, 메시지의 전송이 즉시 이루어지지 못하여도 자동적인 재 전송을 가능하게 한다.

시스템 구성은 자바 클래스들을 각각의 특성에 따라 4개의 층으로 구성하여 이를 통해 유지 보수를 용이하게 하는 구조를 가지고 있다.

이 시스템의 문제점은 라우터를 중심으로 하는 중앙집중형 구조이기에 병목현상이 일어날 경우 시스템이 효율적으로 가동되기 어렵게 될 수 있으며 모든 에이전트를 사용자가 직접 구동시켜야 하는 단점이 있다[7].

2.2 JAFMAS

자바 언어를 이용하여 일반적인 에이전트 구조의 이러한 에이전트를 구현하기 위한 클래스 집합을 제공한다. 모두 16개의 자바 클래스들을 통해 시스템에서 제공되는 기능들을 살펴보면, 먼저 통신(communication)에 관해 directed 통신과 broadcast 통신을 지원한다

다음으로 speech-act 언어(KQML)를 기반으로 하는 메시지 구조를 가진다. 그리고 협력작업에 관해서는 일정한 규칙(rule)을 기반으로 에이전트간의 대화를 통해 공동작업을 조정한다 이는 JAFMAS 내의 각 에이전트들이 가지고 있는 Plan 즉, 시스템내의 다른 에이전트에게 바라는 동작에 대한 선택적인 course 개념을 집합의 형태로 진달함으로 이루어진다 이러한 규칙(rule)의 수행을 위한 인터프라터가 지속적으로 작동한다. 마지막으로 다중 쓰레드를 지원하여 위에서 기술한 각각의 기능들을 동시에 수행할 수 있도록 각 작업에 대한 다중 쓰레드가 할당된다

이 시스템은 시스템내의 각 에이전트가 서버와 클라이언트의 기능을 상황에 따라 모두 수행할 수 있도록 내부에 모든 구조를 가지는 특성을 지닌다 따라서 비-중앙 집중적인 구조를 가지게 되나, 그를 위해 각 에이전트별로 많은 정보와 복잡한 구조를 가지게 되고, 이로 인해 시스템의 가동을 위한 부하가 커지게 되며 사용자가 일일이 각각의 에이전트를 콘솔(console) 윈도우에서 수행시켜야 하는 단점을 갖고 있다[8]

3. 다중 에이전트 동적 실행 시스템의 설계

3.1 시스템 구조

본 논문에서 설계하고 구현한 이질형 에이전트의 동적 구동을 위한 기반구조 시스템의 구성은 <그림1>과 같이 Agent Management System(AMS), Directory Facilitator(DF), Task Agents, Java Native Interface(JNI), User Interface Agent(UIA), Agent Invoking C++ file등으로 구성되었다.

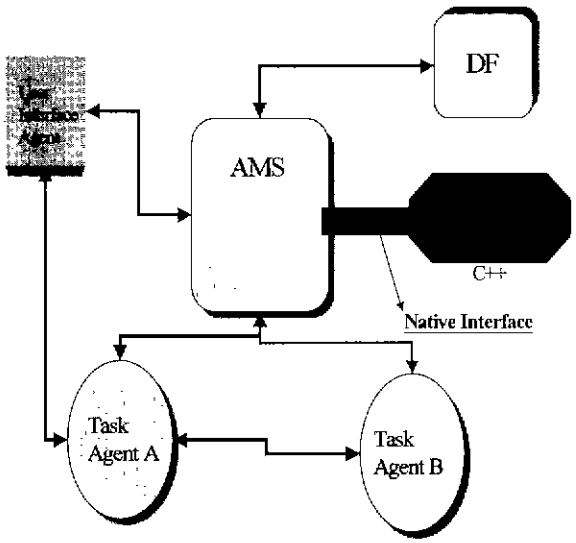
UIA(User Interface Agent)는 다중 에이전트 시스템을 사용하는 사용자와 대화를 하는 에이전트이다 이 에이전트는 각 사용자당 하나의 에이전트가 존재하게 되며 각 사용자를 관리하고 사용자의 요청을 접수하는 역할을 수행한다

AMS(Agent Management System)는 에이전트 플랫폼내의 에이전트의 등록, 동적 구동, 그리고 종료 등의 전반적인 에이전트의 관리를 수행한다 그리고 에이전트에 대한 상태정보(구동 및 종료)를 소유하고 있다.

DF(Directory Facilitator)는 에이전트들이 제공하는 서비스에 대한 정보를 각 에이전트에게 제공한다

Task Agents는 다수가 존재할 수 있으며 각각의 에이전트들은 고유의 작업을 수행할 수 있는 서비스를 제공한다 그리고 자신의 구동시에 필요한 정보와 통신을 위한 정보, 그리고 자신이 제공하는 서비스정보를 소유하고 있다 이러한 정보들은 Task Agent가 구동되면 자동으로 AMS에 등록이 된다

JNI(Java Native Interface)는 자바로 구현할 수 없는 부분을 Native Method로 선언하고 이것을 C나 C++로 구현하여 자바 프로그램에 통합시키기 위한 인터페이스이다 에이전트를 동적으로 구동시키기 위해서는 프로세서를 생성해서 생성된 프로세서 내에서 해당 에이전트를 구동시켜야 되는데 자바언어는 이러한 메쏘드를 지원하지 않는다 프로세서를 생성하기 위해서는 다른 언어인 C++를 사용하여 구현하고 기존의 자바 프로그램에 통합시켜야 한다



<그림 1> 시스템 구조.

3.2 Native Interface를 사용한 동적 구동

다중 에이전트 시스템에서 자바 언어를 사용하는 이유는 자바 고유의 특성인, 잘 정의된 표준 API와 자바 가상머신 때문에 자바로 작성된 소프트웨어가 어느 플랫폼에서나 수정 없이 동작할 수 있는 강력한 장점을 갖고 있기 때문이다. 언급한 것과 같이 시스템 환경의 영향을 거의 받지 않기 때문에 자바를 사용하는 것이 다른 언어를 사용하는 것보다 더욱 이질적인 환경에 적응할 수 있는 시스템을 만들 수 있다 하지만 현재의 자바언어도 몇 가지 약점을 가지고 있다. 자바 가상머신에서 수행되는 인터프리터형 언어이기 때문에 수행속도가 늦은 단점과 C++와 같은 언어에서 지원하는 여러 가지 기능들을 제공하지 않고 있다. 그 중에서 동적으로 에이전트를 구동시키기 위해서는 프로세서를 만들고 Command Line 명령을 사용하여 해당 에이전트를 구동시키는 것이 필요하다[11] 자바에서는 새로운 프로세서를 생성시킬 수 없다 이것을 해결하기 위해서 자바의 Native interface를 사용하였다[9][10]. 자바에서는 다른 언어와의 확장은 위해서 이 인터페이스를 제공하고 있다 본 시스템에서는 특정 에이전트를 동적으로 구동시키는 부분은 C++로 구현하였고 이를 Java Native Interface를 사용하여 본 시스템에 확장하여 사용하고 있다.

3.3 통신 형태

본 시스템에서 통신을 위한 언어로는 현재 널리 사용되고 있는 KQML(Knowledge Query and Manipulation Language)형태의 언어를 사용한다 본 시스템에서 이 KQML은 이형질의 에이전트들 간의 통신, 에이전트들과 User Interface Agent(UIA)간의 통신, UIA와 AMS간의 통신에 사용된다. 이형질의 에이전트간의 통신을 위해서는 소켓(socket) 연결을 통한 KQML형태의 메시지 신송을 통해서 이루어진다[2][3][4][12].

모든 구성요소들 간의 통신 형태는 다음과 같다.

1 메시지 전송의 요청이 발생.

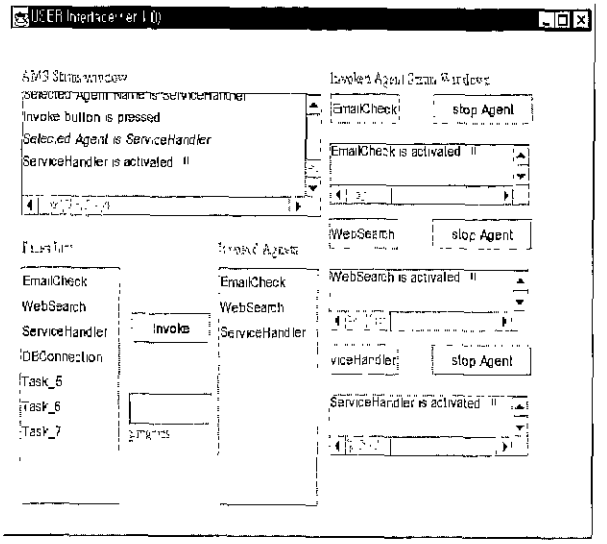
2 구성요소들간의 connection 요청
3 connection이 실립되면 단계 4로 간다 connection 실립 실패 시에 전송할 메시지 저장 후 일정 시간이 경과 후 단계 2로 간다
4 메시지 전송.
5 connection 종료

3.4 시스템의 작업 흐름도

다음은 본 시스템에서의 작업 수행 과정을 나열한 것이다

① 먼저 AMS를 구동시킨다 구동된 AMS는 사용자 인터페이스 에이전트(UIA)를 구동시킨다
② 사용자는 UIA를 통해서 수행할 작업을 지시하며, 이 작업 요청 메시지는 AMS에 전달된다.
③ AMS는 전달된 메시지로부터 요청된 작업에 맞는 에이전트에 대한 정보를 DF에게 묻는다
④ DF는 해당 에이전트의 정보를 AMS에게 전달한다
⑤ AMS는 DF로부터 받은 정보로 해당 에이전트를 구동시킨다.
⑥ 사용자는 구동된 Task Agent A에게 작업 요청 메시지를 전달된다.
⑦ Task Agent A의 작업이 수행되다
⑧ Task Agent A기 작업 수행 중에 자신이 처리할 수 없는 추가적인 작업을 처리할 수 있는 에이전트에 대한 정보를 AMS에게 문의한다
⑩ 단계 ③, ④, ⑤ 순서로 수행(동일)된다
⑪ 새로이 구동된 Task Agent B에게 작업을 지시한다
⑫ 새로이 생성된 Task Agent B의 작업이 끝나면 결과를 통합해서 결과를 출력한다
⑬ 위와 같이 서비스의 종류에 따라 이에 해당되는 에이전트들이 동적으로 구동된다

4. 다중 에이전트 동적 실행 시스템의 구현



<그림 2> User Interface Agent

본 논문에서 이질적인 다중 에이전트의 동적 실행 기능을 시험하기 위해 구현된 시스템은 NT 운영체제에서 개발되었고 기본 기반구조인 AMS, DF 그리고 사용자 인터페이스는 자바 언어로 작성되었으며 에이전트 자동실행 부분은 C++로 개발되었다 그리고 이형질의 에이전트를 시험하기 위해서 자바와 C++를 사용하여 게개의 Task Agent로 만들었다 <그림 2>는 사용자 인터페이스의 결과 화면이다.

사용자는 Task List에서 원하는 작업을 선택하면 이것이 AMS로 전달되고, 이 전달된 사항으로 DF에서 이 작업을 서비스할 수 있는 에이전트의 정보를 AMS에게 보내주면 AMS는 이 정보를 받아서 해당 에이전트를 수행시키게 된다 수행된 에이전트들의 상태정보는 위의 'Invoked Agent List'에 나타나게 된다 그리고 사용자는 구동된 에이전트로 작업을 수행한다 작업의 수행 중에 여러 Task Agent들이 현재 작업을 수행하는 에이전트에 의해 자동적으로 구동되며 구동된 에이전트간의 협동을 통해서 사용자의 요청을 수행하게 된다

5. 결론 및 향후 연구 계획

기존의 자바 언어를 사용한 다중 에이전트 기반구조는 고유의 서비스를 제공하는 Task Agent들을 동적으로 구동시키지 못함으로써 시스템 자원을 효율적으로 사용하지 못했으며, 또한 필요한 에이전트를 동적 구동시키지 못함으로 인하여 Task Agent들을 효과적으로 관리하지 못하였다 본 논문에서 제안된 시스템은 이러한 문제점을 Java Native Interface를 사용하여 C++로 작성된 동적 구동부분을 다중 에이전트 기반구조에 포함시킴으로써 해결하였고, 더불어 이형질의 에이전트들을 동적으로 구동시킬 수 있게 하였다 또한 에이전트간의 통신 언어인 KQML을 사용함으로써 다른 언어로 만들어진 어떠한 에이전트들 간의 통신도 가능하게 함으로써 이질적인 환경에서의 호환성과 적응성을 향상시켰으며 이로 인하여 사용자에게 다양한 서비스를 제공할 수 있다

향후 연구 계획은 현재 AMS 자체에 구동될 에이전트들의 정보가 정적으로 저장되어 있는데, 이를 에이전트들의 등록을 통해서 동적으로 관리함으로써 기존에 미리 개발된 에이전트들에 대한 재사용 성을 높이고자 한다

[참고문헌]

[1] 최중민, "에이전트 개요와 연구방향", 정보과학회지 Vol 15 No 3, pp 7-16, 1997
[2] Labrou, Y, Finin, T, "A Proposal for a New KQML Specification", Technical Report TR-CS-97-03, CSEE, UMBC, 1997
[3] Finin, T, "KQML as an Agent Communication Language", In The Proceedings of the Third International Conference on Information and Knowledge Management, ACM Press, 1994
[4] 노현칠, 이근배, 이종혁, "KQML에 기반한 멀티에이전트 통신 환경", 정보과학회지 Vol. 15 No 3, 1997
[5] FIPA, "Agent Management(TC1)", FIPA '97 Draft Specification, 1997
[6] FIPA, "Application Design Test Personal Travel Agent (TC4)", FIPA'97 Draft Specification, 1997
[7] Jeon, H C, "JATLite Feature", (available at http://java stanford.edu), 1996
[8] Chauhan, D Baker, B, "JAFMAS Software Architecture", (available at http:// www eecs uc edu/~abaker/JAFMAS/), 1997
[9] Stearns, B, "Using the Java Native Interface(JNI)", (available at http://java sun com/docs/books/tutorial/native1 1/index html), 1998
[10] Stearns, B, "Java Native Interface Programming", (available at http://java sun com/docs/books/tutorial/native1 1/implementing/), 1998
[11] Hart, J. M, "Win32 System Programming", Addison Wesley Developers Press, pp 143~178, 1997
[12] Harold, E R, "Java Network Programming", O'REILLY, pp 149~211, 1997