KIPS Tr. Software and Data Eng.
Vol.3, No.3 pp.125~134 pISSN: 2287-5905



http://dx.doi.org/10.3745/KTSDE.2014.3.3.125

# A Multi-Agent Message Transport Architecture for Supporting Close Collaboration among Agents

Hai Jin Chang†

## ABSTRACT

This paper proposes a multi-agent message transport architecture to support application areas which need fast message communications for close collaboration among agents. In the FIPA(Foundation of Intelligent Physical Agents) agent platform, all message transfer services among agents are in charge of a conceptual entity named ACC(Agent Communication Channel). In our multi-agent message transport architecture, the ACC is represented as a set of system agents named MTSA(Message Transfer Service Agent). The MTSA enables close collaboration among agents by supporting asynchronous communication, by using Reactor pattern to handle agent input messages efficiently, and by selecting optimal message transfer protocols according to the relative positional relationships of sender agents and receiver agents. The multi-agent framework SMAF(Small Multi-Agent Framework), which is implemented on the proposed multi-agent message transport architecture, shows better performance on message transfer among agents than JADE(Java Agent Development Environment) which is a well-known FIPA-compliant multi-agent framework. The faster the speed of message transfer of a multi-agent architecture becomes, the wider application areas the architecture can support.

**Keywords : Multi-Agent Framework, Close Collaboration, Reactor Pattern, Message Transfer Service, JADE**

# 에이전트들 간의 밀접한 협력을 지원하기 위한 다중 에이전트 메시지 전송 구조

장 혜 진†

## 요 약

본 논문은 에이전트들 간의 긴밀한 협력을 위해 신속한 메시지 통신을 필요로 하는 응용 분야들을 지원하기 위한 다중 에이전트 메시지 전송 구조를 제안한다. 제안하는 구조는 FIPA(Foundation of Intelligent Physical Agents) 표준 에이전트 플랫폼 규격에서 에이전트들에게 메시지 전송 서비스를 제공하는 요소인 ACC(Agent Communication Channel)를 MTSA(Message Transfer Service Agent)라는 에이전트들의 집합으로 표현한다. MTSA는 비동기 메시지 통신을 지원하며, 메시지 수신을 효율적으로 처리하기 위해 반응자 패턴(reactor pattern)을 사용하며, 메시지 송신 에이전트와 수신 에이전트의 상대적 위치 관계에 따라 에이전트들 간에 최적의 통신 수단을 선택하여 메시지를 전송하여 에이전트들 간의 긴밀한 협력이 가능하도록 한다. 제안된 메시지 전송 구조에 따라 구현된 다중 에이전트 프레임워크 SMAF(Small Multi-Agent Framework)는 잘 알려진 에이전트 프레임워크 JADE(Java Agent Development Environment)와 비교하였을 때 향상된 메시지 전송 능력을 보인다. 다중 에이전트 구조의 메시지 통신 속도가 고속화되면 될수록 그 다중 에이전트 구조는 더 다양한 응용 분야들에 적용될 수 있을 것이다.

**키워드 : 다중 에이전트, 협력, 반응자 패턴, 메시지 전송 서비스, JADE**

## 1. 서 론

다중 에이전트 구조는 네트워크상에 분산된 복수개의 에이전트들이 에이전트 통신 언어로 표현된 메시지 교환을 통한 협력에 의해 크고 복잡한 문제를 푸는 것을 목적으로 하며, 다중 에이전트 구조상의 에이전트들은 분산성(distribution)과 자율성(autonomy)과 같은 장점을 갖는다. 다중 에이전트 구조는 에이전트 중심 소프트웨어 공학(agent-oriented software engineering), 분산 문제 풀이(distributed problem solving), 다중 에이전트 시뮬레이션(multi-agent simulation), 다중 로봇 시스템(collective robotics) 등과 같은 다양한 분야에서 사용되고 있다[1]. 에이전트에 대한 연구의 방향은 다양하다.

※ 본 논문은 상명대학교 교내선발과제로 연구되었음.
† 정 회 원 : 상명대학교 소프트웨어공학과 교수
논문접수 : 2013년 12월 10일
수 정 일 : 1차 2014년 1월 22일, 2차 2014년 1월 27일, 3차 2014년 2월 3일
심사완료 : 2014년 2월 3일
* Corresponding Author : Hai Jin Chang(hjchang@smu.ac.kr)

에이전트를 철학적이며 인공 지능적 관점에서 믿음, 욕망, 의도를 가진 구조로 보는 연구[2], 에이전트에게 계획 생성 능력과 같은 지적인 능력을 부여하고자 하는 연구[3], 그리고 에이전트를 소프트웨어 공학적 측면에서 소프트웨어 컴포넌트로 보는 연구[4]의 방향도 존재한다. 본 논문은 에이전트들이 서로 긴밀하게 협동하는 데 필요한 메시지 전송 능력을 중점으로 고려한다.

다중 에이전트 구조는 분산성을 강조한다. 다중 에이전트 구조의 에이전트들은 일반적으로 네트워크로 연결된 여러 개의 호스트들에 분산되어 존재하며, 에이전트들 간의 메시지 통신 프로토콜로는 TCP(Transmission Control Protocol)나 HTTP(Hypertext Transfer Protocol)와 같은 네트워크 통신 프로토콜들이 주로 사용되고 있다. 따라서 응용 시스템 구성 요소들이 네트워크상에 분산되어 있지 않거나, 응용 시스템 구성 요소들 간에 TCP나 HTTP와 같은 네트워크 통신 프로토콜보다 상대적으로 빠른 긴밀한 협력을 필요로 하는 분야에서는 다중 에이전트 구조가 적합하지 않을 수 있다는 평가가 존재한다[5]. 예를 들어, 지식 원천들과 블랙보드가 동일한 호스트에 존재하는 전통적인 블랙보드 시스템에서는 공유 메모리(shared memory)와 같은 프로세스 간 통신(interprocess communication) 방법이 사용되고 있으며, 그런 블랙보드 시스템에서는 상대적으로 느린 네트워크 통신 프로토콜을 사용하는 것이 바람직하지 않을 것이다. 하지만, 다중 에이전트 구조가 보다 빠른 에이전트 메시지 통신을 지원한다면 에이전트 구조가 가진 분산성, 자율성 등의 장점을 살리면서 지능형 로봇과 같은 보다 다양한 응용 분야에 에이전트 구조를 적용할 수 있을 것이다.

FIPA(Foundation for Intelligent Physical Agents)는 에이전트 시스템에 대한 국제 표준을 규정하는 단체이다. 본 논문이 제안하는 에이전트 메시지 전송 구조는 FIPA 표준 규격에서 에이전트 플랫폼 내의 에이전트들 간의 메시지 전송 서비스를 담당하는 요소인 ACC(Agent Communication Channel)[6]를 MTSA(Message Transfer Service Agent)라는 에이전트들의 집합으로 표현한다. MTSA는 에이전트들 간 비동기 메시지 통신을 지원하며, 반응자 패턴(reactor pattern)[7]을 사용하여 메시지의 수신을 효과적으로 처리하며, 송신 에이전트와 수신 에이전트의 상대적 위치 관계에 따라 최적의 메시지 전송 프로토콜을 선택하여 메시지를 전송한다. MTSA가 사용하는 송신 에이전트와 수신 에이전트의 상대적 위치 관계의 기준은 프로세스와 호스트 기계이다. 송신 에이전트와 수신 에이전트는 동일 프로세스에 위치할 수도 있고, 동일 호스트에 위치할 수도 있으며, 서로 다른 호스트에 위치할 수도 있다.

본 논문이 제안하는 에이전트 메시지 전송 구조는 FIPA 표준 규격들[8]과의 완전한 호환을 목표로 하지 않지만 FIPA 표준 규격들로부터 많은 개념들을 빌려 사용한다. 본 논문은 FIPA 표준 규격 자체 및 그에 호환하는 잘 알려진 다중 에이전트 프레임워크의 하나인 JADE(Java Agent Development Environment)[9]와의 비교를 중심으로 제안 구조를 기술하고 성능 평가를 수행한다.

본 논문의 제 2장은 관련 기술 및 연구들에 대한 내용이다. 제 3장은 제안하는 에이전트 메시지 전송 구조를 기술한다. 제 4장은 잘 알려진 다중 에이전트 프레임워크인 JADE와 본 논문이 제안하는 구조에 따라 개발된 다중 에이전트 프레임워크 SMAF(Small Multi-Agent Framework)의 성능을 비교한다. 제 5장은 결론이다.

## 2. 관련 기술 및 연구

에이전트 시스템들 간의 상호 호환성(interoperability)을 보장하기 위한 에이전트 규격의 표준화에 대한 연구 단체에는 FIPA[8], KSE(Knowledge Sharing Effort)[10], OMG(Object Management Group)[11] 등이 있다. FIPA 표준 규격은 에이전트 통신 언어에 대한 규격[12]을 규정할 뿐 아니라 다른 표준 규격들에 비해 에이전트 시스템을 관리하는 데 필요한 AMS(Agent Management System), DF(Directory Facilitator)와 같은 시스템 에이전트들[13]을 규정하고 있다는 특징을 갖는다. JADE[9], FIFA-OS[14] 등의 다중 에이전트 프레임워크들이 FIPA 표준 규격을 따르고 있다. 모바일 에이전트 플랫폼 Grasshopper[15]는 FIPA와 OMG의 규격을 모두 따르고 있다.

FIPA의 에이전트 관리 시스템 규격[13]에 따르면 하나의 에이전트 플랫폼은 다음 Fig. 1과 같이 하나의 AMS, 하나 이상의 DF, 에이전트들 간의 메시지 전송 서비스(Message Transport Service)[6]를 전담하는 MTS(Message Transport System), 그리고 응용 에이전트들로 구성된다. MTS는 에이전트들 간의 메시지 전송 서비스를 전담하는 요소이다. MTS는 동일 에이전트 플랫폼 내의 에이전트들 간의 메시지 전송뿐 아니라 서로 다른 에이전트 플랫폼에 존재하는 에이전트들과의 메시지 전송도 담당한다. MTS는 FIPA 메시지 전송 서비스 표준 규격[6]에서는 ACC(Agent Communication Channel)로 표현되기도 한다. ACC는 MTS의 기능적인 측면을 강조하는 보다 구체적인 표현이라고 할 수 있다. 본 논문은 이후의 기술에서 MTS보다 ACC라는 표현을 사용한다.

FIPA 표준 규격과 무관하게 설계된 다중 에이전트 프레임워크인 EMAF(Extensible Multi-Agent Framework)[16]는 에이전트들에게 메시지 전송 서비스를 제공하는 ACC와

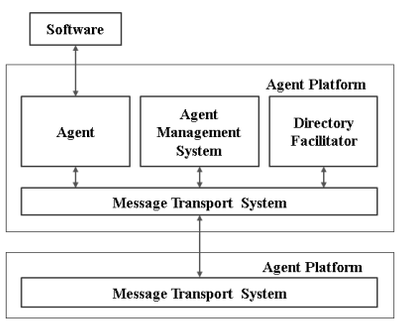


Fig. 1. FIPA Agent Platform

같은 별도의 요소를 갖지 않으며, 모든 에이전트는 서로 P2P(Peer-to-Peer) 방식으로 직접 통신한다. 하지만 모든 에이전트들이 직접 P2P 방식으로 통신한다면 에이전트들의 수가 많아짐에 따라 소켓과 같은 통신을 위한 자원들의 관리 부담이 급격히 증가할 수 있다. 에이전트들 간의 메시지 전송을 전담하는 ACC와 같은 요소를 사용하는 것은 응용 에이전트들이 다른 응용 에이전트들과의 네트워크 연결을 직접 관리하고 유지하는 부담을 감소시킨다.

FIPA 표준 에이전트 규격은 에이전트 시스템들 간의 상호운용성(interoperability)을 위한 외부적인 큰 틀만을 규정할 뿐 시스템의 내부적이고 세부적인 구현은 에이전트 시스템 개발자들의 재량에 맡기고 있다. FIPA 메시지 전송 표준 규격[6]은 ACC의 구조나 ACC가 사용할 수 있는 구체적인 메시지 전송 프로토콜들(message transport protocols)에 대하여 세부적으로 규정하고 있지 않다. 다만 에이전트 플랫폼간의 메시지 전송에 대해서는 IIOP(Internet Inter-ORB Protocol)를 사용하는 메시지 전송 프로토콜[17]과 HTTP를 사용하는 메시지 전송 프로토콜[18]를 ACC의 기본 메시지 전송 프로토콜로 지정하고 있으며, 성능 향상을 위하여 ACC가 메시지의 버퍼링 능력을 가질 수 있다고 규정하고 있을 뿐이다. FIPA 메시지 전송 표준 규격은 동일 에이전트 플랫폼 내의 에이전트들 간의 메시지 전송 방식에 대해서는 규정하고 있지 않다.

본 논문이 제안하는 에이전트 메시지 전송 구조는 FIPA 표준 규격과의 완전한 호환을 설계 목적으로 하지 않지만 FIPA 표준 규격들[6, 11, 12]로부터 다음과 같은 개념들과 규격들을 빌려다 쓰고 있다. (에이전트 메시지의 reply-with 필드와 in-reply-to 필드에 대한 설명은 제 3.1절에 나온다.)

- Fig. 1의 FIPA 에이전트 플랫폼의 개념, 특히 메시지 전송 서비스를 전담하는 ACC의 개념.
- 에이전트 메시지를 구성하는 여러 필드들의 이름과 의미. 특히 reply-with 필드와 in-reply-to 필드의 의미.

FIPA 규격에 호환하는 대표적인 다중 에이전트 프레임워크인 JADE[9]에서 하나의 에이전트 플랫폼은 FIPA 규격에 따라 AMS, DF, 그리고 ACC를 포함한다. JADE는 FIPA 규격들을 준수하지만, FIPA 규격이 규정하지 않는 세부적인 설계와 구현에 있어서는 에이전트 컨테이너(container)[19]와 같은 독자적인 개념들을 사용하고 있다. JADE에서 하나의 에이전트 플랫폼은 하나 이상의 컨테이너들로 구성된다. JADE에서 ACC는 물리적으로 플랫폼상의 컨테이너들에 분산되어 있다. 컨테이너는 서로 밀접하게 협동해야 하는 에이전트들을 묶는 단위이며, 컨테이너는 에이전트의 한 종류가 아니다.

JADE는 본 논문이 제안하는 구조와 마찬가지로 메시지 전송 성능의 향상을 위하여 다양한 메시지 전송 프로토콜을 제공하며, 송신자 에이전트와 수신자 에이전트의 상대적 위치 관계에 따라 최적의 메시지 전송 프로토콜을 선택할 수 있도록 한다. JADE에서 송신자와 수신자의 상대적 위치 관계의 기준은 컨테이너와 플랫폼이다. JADE는 메시지의 송신자 에이전트와 수신자 에이전트가 동일 컨테이너 내부에 존재할 때에는 메시지 객체를 복사하고 그 복사본의 참조를 수신 에이전트에 전달하는 방식을 사용한다. JADE에서 하나의 컨테이너 내의 모든 에이전트들은 동일한 프로세스에 존재한다. 송신자와 수신자가 동일 에이전트 플랫폼의 서로 다른 컨테이너들에 존재하는 경우 JADE는 Java RMI(Remote Method Invocation)을 사용한다. 그리고 송신자와 수신자가 서로 다른 에이전트 플랫폼에 있을 때에는 FIPA 규격에 따라 Java 언어가 지원하는 ORB(Object Request Broker) 기반의 IIOP(Internet Inter-ORB Protocol), ORBACUS ORB[21] 기반의 IIOP, 또는 HTTP를 메시지 전송 프로토콜로 사용한다. JADE의 이러한 메시지 전송 체계는 송신자와 수신자의 위치 관계 정보를 사용하여 통신 성능을 개선할 수 있도록 하지만 다음과 같은 문제점들을 가질 수 있다.

첫째, JADE는 만일 송신 에이전트와 수신 에이전트가 동일 에이전트 플랫폼의 서로 다른 컨테이너에 존재한다면 두 에이전트가 동일한 프로세스 상에 존재하는 경우에도, 서로 다른 프로세스에 존재하는 경우와 마찬가지로, Java RMI를 사용하여 메시지를 전송한다는 문제점을 갖는다. Java RMI를 통한 메시지 전송은 기본적으로 메시지 객체를 직렬화(serialization)시켜 보내고 역직렬화(deserialization) 하여 복원하는 부담을 포함한다.

둘째, JADE가 사용하는 컨테이너는 에이전트의 한 종류가 아니다. JADE 컨테이너는 동일 에이전트 플랫폼 내의 컨테이너들 간의 메시지 전송을 위해서는 Java RMI 서버를 사용하고, 다른 플랫폼과의 메시지 전송을 위해서는 CORBA IIOP 서버를 내부적으로 사용한다. 에이전트들 간의 메시지 전송 서비스를 제공하는 JADE 컨테이너가 에이전트가 아니므로 JADE의 에이전트들은 다른 에이전트들뿐 아니라 에이전트가 아닌 컨테이너와 상호 작용해야 한다. 에이전트 개발자들은 에이전트 시스템의 개발을 위해 에이전트들뿐 아니라 에이전트의 구조와 API(Application Programming Interface)뿐 아니라 컨테이너의 구조와 API를 이해해야 한다.

본 논문이 제안하는 에이전트 메시지 전송 구조에서 ACC를 구성하는 MTSA들은 송신 에이전트와 수신 에이전트 간의 상대적 위치 관계를 프로세스와 호스트의 관점에서 파악한다. 그리고 그 파악된 관계에 따라 프로세스 내 MTP(Message Transfer Protocol), 프로세스간 MTP, 또는 호스트간 MTP를 사용하여 메시지를 전송한다. MTSA가 송신자 에이전트와 수신자 에이전트 간에 최적의 MTP를 선택하는 기준으로 프로세스와 호스트를 사용하는 것은 합리적이다. 왜냐하면 일반적으로 프로세스 내 통신이 프로세스간 통신보다 효율적이며, 동일 호스트상의 프로세스간 통신이 서로 다른 호스트간 통신보다 효율적이기 때문이다.

본 논문이 제안하는 구조에서 MTSA들은 프로세스와 호스트의 기준으로 메시지 송신 에이전트와 수신 에이전트의

상대적 위치 관계를 파악하므로 앞에서 기술한 JADE의 ACC의 첫 번째 문제점들을 갖지 않는다. 본 논문이 제안하는 구조에서는, 동일한 프로세스 상에 존재하는 에이전트들 간에는 항상 메시지 객체가 복사되고 그 참조가 직접 전달되는 방식으로 메시지 전송이 이루어지기 때문이다. 또한 본 논문이 제안하는 구조는 앞에서 기술한 JADE의 두 번째 문제점들을 갖지 않는다. 왜냐하면, 본 논문이 제안하는 에이전트 메시지 전송 구조의 MTSA는 에이전트의 한 종류이기 때문이다. 본 논문의 제 3.3절과 3.4절에서 기술되는 MTSA와 응용 에이전트는 모두 에이전트로서의 공통 API를 갖는다.

에이전트 메시지 전송 구조는 HTTP와 같은 통신 프로토콜뿐 아니라 JMS(Java Message System)과 같은 메시지 지향 미들웨어를 기반으로 개발될 수 있다. JMS는 분산 환경에서 RMI를 대신해 사용될 수 있다. JMS는 비동기 통신 모드를 지원하며 메시지 큐를 사용하여 RMI보다 효율적이고 안정적인 메시지 통신을 지원한다. 따라서 JMS와 같은 잘 정의된 메시지 지향 미들웨어를 에이전트 메시지 전송 구조의 하부 구조로 사용하는 것은 시스템의 성능, 다른 시스템과의 호환성 등의 면에서 유리하다. 하지만 JMS 자체만으로는 에이전트 메시지 전송 구조의 기능을 제공할 수는 없다. 왜냐하면 JMS는 본 논문이 제안하는 메시지 전송 구조와 달리 프로세스 내 통신, 프로세스간 통신, 호스트간 통신을 구분하여 지원하지 않으며, JMS 자체는 에이전트 식별자, AMS와 같은 시스템 에이전트들과의 협력, 그리고 reply-with 메시지 필드와 in-reply-to 메시지 필드를 이용한 에이전트 질의응답 프로토콜 등의 기능을 지원하지 않기 때문이다.

JADE에 JMS 기반의 MTP를 추가하기 위한 연구[22]에 따르면 JMS 기반의 MTP는 4.1절에 기술된 RTT(Round Trip Time) 테스트에서 송신 에이전트와 수신 에이전트의 쌍이 100개인 경우 IIOP 기반의 기존 MTP보다 약 3.5~3.8배 정도의 통신 향상을 보이며, 기존 MTP 중 가장 빠른 HTTP 기반의 MTP보다는 약 1.7~1.8배 정도의 통신 성능 향상을 보이고 있다. 하지만 본 논문이 제안하는 메시지 전송 구조를 구현하는 SMAF는 4.2절의 Table 2에 기술된 바와 같이 송신 에이전트와 수신 에이전트의 쌍이 100개인 경우 '2 프로세스, 2 호스트, 2 플랫폼' 환경에서 JADE의 HTTP 기반의 MTP보다 약 27.6배 이상의 빠른 성능을 보인다.

본 논문이 제안하는 구조는 JADE의 메시지 전송 구조와 마찬가지로 다양한 종류의 통신 프로토콜들과 JMS와 같은 다양한 메시지 지향 미들웨어에 근거한 MTP들을 지원할 수 있다. 본 논문이 제안하는 구조는 메시지 전송의 성능을 높이기 위하여 반응자 패턴과 비동기 통신을 지원한다. 반응자 패턴에 대한 내용은 제 3.5절에서 기술된다. 비동기 통신에 대한 내용은 제 3.6절에 나온다.

## 3. 제안 멀티 에이전트 메시지 전송 구조

### 3.1 에이전트의 식별 및 에이전트 메시지

에이전트들이 사용하는 에이전트 통신 언어는 에이전트 메시지의 형태 및 의미를 규정한다. FIPA가 규정하는 에이전트 통신 언어 규격[12]은 에이전트 통신 언어인 KQML(Knowledge Query Manipulation Language)[23]로부터 많은 개념을 빌려왔다. 다음 Fig. 2는 본 논문에서 제안하는 에이전트 메시지 전송 구조가 사용하는 에이전트 통신 메시지의 구조이다. sender 필드와 receiver 필드는 송신자 에이전트와 수신자 에이전트의 고유한 식별자를 담는다. content 필드와 language 필드는 각각 전송하고자 하는 내용과 그 내용을 표현하는 언어의 종류를 담는다. conversation-id 필드는 메시지들의 그룹들을 서로 구분하기 위한 대화 그룹 식별자를 담는다.


Fig. 2. Agent Message Format

reply-with 필드와 in-reply-to 필드는 에이전트가 다른 에이전트에게 보낸 요청 메시지와 그 요청에 대한 응답 메시지의 짝을 구분하기 위한 필드이다. 요청 메시지란 reply-with 필드 값이 null이 아닌 메시지를 의미하고, 응답 메시지란 in-reply-to 필드 값이 null이 아닌 메시지를 의미한다. 만일 에이전트 $A$가 reply-with 필드 값이 null이 아닌 어떤 값 $t$를 가진 요청 메시지 $m1$을 송신한 후, in-reply-to 필드의 값이 $t$인 응답 메시지 $m2$를 수신하였다면, 메시지 $m2$는 요청 $m1$에 대한 응답으로 해석된다. 요청 필드 값과 응답 필드 값의 쌍들을 조합하면 전자 상거래 등에 필요한 다양한 프로토콜들을 지원할 수 있다. 모든 에이전트는 고유한 reply-with 필드 값을 생성할 수 있는 기능을 갖는다.

### 3.2 에이전트 플랫폼

본 논문이 제안하는 에이전트 메시지 전송 구조에서 하나의 에이전트 플랫폼은 Fig. 3과 같이 하나의 AMS(Agent Management System), 하나 이상의 DF(Directory Facilitator), MTSA들, 그리고 응용 에이전트들로 구성된다. MTSA는 에이전트들 간의 메시지 전송 서비스를 전담하는 에이전트이다. MTSA들은 플랫폼 내부에 존재하는 에이전트들 간의 메시지 전송 서비스뿐 아니라 및 다른 플랫폼에 존재하는 에이전트들과의 메시지 전송 서비스를 제공한다. 응용 에이전트들은 다른 응용 에이전트들과 직접 통신하지 못하며 MTSA를 통해서만 다른 응용 에이전트와 통신할 수 있다. 제안하는 구조는 Fig. 1에서 기술된 FIPA 에이전트 플랫폼 상의 메시지 전송 시스템(message transport system)을 하나의 에이전트 플랫폼에 물리적으로 분산된 MTSA들의 집합으로 표현한다. 본 논문이 제안하는 구조에서 AMS와 DF는 MTSA의 서비스를 이용하는 응용 에이전트의 한 종류로 구현될 수 있다.

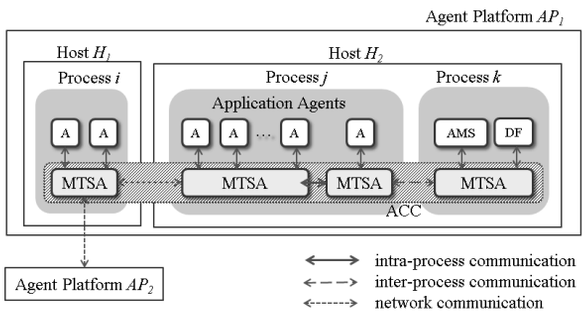


Fig. 3. An Agent Platform in The Proposed Architecture

MTSA들은 송신 에이전트와 수신 에이전트가 존재하는 프로세스들과 호스트들의 상대적 위치 관계를 이용하여, 프로세스 내 MTP(Message Transport Portocol), 프로세스간 MTP, 그리고 호스트간 MTP중 최적의 통신 프로토콜을 선택할 수 있다. 제안된 구조는 메시지 전송 서비스를 MTSA라는 에이전트들이 담당한다는 점에서, 에이전트들 간 메시지 전송 서비스를 에이전트가 아닌 컨테이너가 제공하는 JADE[10]와 구분된다.

### 3.3 응용 에이전트의 구조

응용 에이전트는 에이전트 메시지의 전송에 대한 처리를 MTSA에게 전담시킴으로써 단순한 구조를 갖고 적은 자원을 사용하도록 설계되었다. 응용 에이전트는 자신이 사용하는 MTSA를 통해서만 다른 에이전트들과 통신하며, 자신이 사용하는 MTSA와는 프로세스 내(intra-process) MTP를 사용하여 통신한다. 응용 에이전트의 구조는 다음 Fig. 4와 같다.

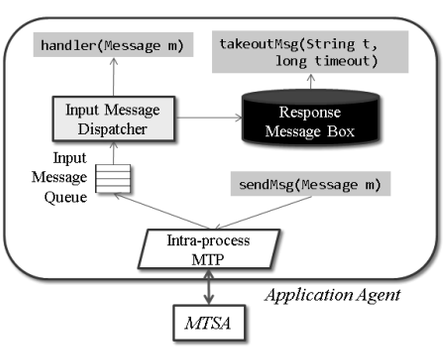


Fig. 4. Application Agent

MTSA가 보낸 메시지들은 프로세스 내 MTP를 통하여 응용 에이전트의 입력 메시지 큐(input message queue)에 삽입된다. 입력 메시지 분배기(input message dispatcher) 쓰레드는 입력 메시지 큐에서 메시지를 꺼내어 그 메시지가 응답 메시지이면 응답 메시지 함(response message box)에 넣고, 아니면 메시지 처리기 hander(Message m)을 호출하여 메시지를 처리하는 작업을 반복한다. 메시지 처리기는 응답 메시지가 아닌 메시지가 도착할 때마다 그 메시지의 처리를 위해 자동 호출되는 콜백 메서드(callback method)이다. 응용 에이전트의 기능은 메시지 처리기의 내부 코드 구현에 따라 달라진다. 메시지 처리기는 그 코드 구현에 따라 수신 메시지의 처리를 늦추거나, 수신 메시지를 다른 에이전트에게 전달하여 처리되도록 하거나, 심지어는 수신 메시지를 무시할 수 있도록 작성될 수 있으므로, 메시지 처리기는 에이전트의 자율성(autonomy)의 기초가 된다. Fig 4의 응답 메시지 함과 takeoutMsg(String s, long timeout) 메서드에 대한 내용은 제 3.6절에서 기술된다.

응용 에이전트 객체 $s$가 자신이 사용하는 MTSA $r$에게 메시지 $m$을 보내기 위한 sendMsg(Agent $s$, MTSA $r$, Message $m$)의 알고리즘은 Fig. 5와 같다.

```
void sendMsg(Agent s, MTSA r, Message m) {
    String t = the value of the reply-with field of m;
    if (t is not null) {
        /* m is a request message */
    put the pair <t, null> into the response message box
    of s;
}
    /* use an intra-process MTP(Message Transfer
    Protocol) */
    Message cm = make a copy of m;
    put cm into the input message queue of r by using
    an intra-process MTP;
}
```

Fig. 5. Algorithm sendMsg(Agent **s**, MTSA **r**, Message **m**)

위 알고리즘은 매우 간결하고 빠른 코드로 구현될 수 있다. 왜냐하면 응용 에이전트는 MTSA와만 통신하며, 프로세스내 MTP(Message Transfer Protocol)만을 사용하기 때문이다.

### 3.4 MTSA의 구조

MTSA의 구조는 다음 Fig. 6과 같다. 프로세스 내 MTP (Message Transfer Protocol)만을 사용하는 응용 에이전트와 달리, MTSA는 프로세스간 MPT와 호스트간 MTP도 사용한다.

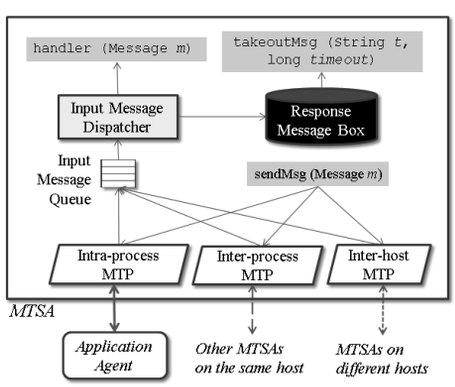


Fig. 6. MTSA

MTSA의 입력 메시지 분배기 쓰레드와 메시지 처리기의 기능은 응용 에이전트의 해당 기능들과 기본적으로 유사하다. MTSA의 비동기 통신 및 그를 지원하기 위한 응답 메시지 함과 takeoutMsg(String s, long timeout) 메서드에 대한 내용은 제 3.6절에서 기술된다.

MTSA *s*가 응용 에이전트 *r*에게 메시지 *m*을 보내는 sendMsg(MTSA *s*, Agent *r*, Message *m*)의 동작은 다음 Fig. 7과 같다.

```
void sendMsg(MTSA s, Agent r, Message m) {
  String t = the value of the reply-with field of m;
  if (t is not null) {
    put the pair <t, null> into the response message
    box of s;
  }

  if (the s and the MTSA of r is on the same
    process) {
  Message cm = make a copy of m;
  put directly cm into the input message queue of the
  MTSA of r by using an intra-process MTP;
  } else if (s and the MTSA of r are on the same
    host) {
    send m to the MTSA of r by using an
    inter-process MTS;
  } else {
    send m to the MTSA of r by using an inter-host
    MTS;
  }
}
```

Fig. 7. Algorithm sendMsg(MTSA ***s***, Agent ***r***, Message ***m***)

위 sendMsg(MTSA *s*, Agent *r*, Message *m*) 알고리즘에서, 만일 수신 에이전트가 MTSA와 동일한 프로세스 내에 존재하면 MTSA는 전송해야 하는 메시지의 복사본을 프로세스 내 MTP(Message Transfer Protocol)를 통하여 수신 에이전트에게 전달한다. 만일 수신 에이전트가 MTSA와 동일한 호스트 내에 존재하면 프로세스간 MTP를 사용하여 메시지를 전송한다. 프로세스간 MTP로 UNIX 소켓, 파이프 등과 같은 다양한 방법을 사용할 수 있다. 위 알고리즘은 만일 송신 에이전트와 수신 에이전트가 서로 다른 호스트에 존재한다면 호스트간 MTP를 사용한다. 호스트간 MTP로는 UDP, TCP, HTTP와 같은 방법들을 사용할 수 있다.

MTSA들이 구체적으로 어떤 종류의 프로세스간 MTP들과 호스트간 MTP들을 구현하는가는 에이전트들이 활동하는 환경에 사용되는 장비들의 운영 체계의 특성이나 지원 통신 프로토콜들에 따라 달라질 수 있다.

### 3.5 반응자 패턴

클라이언트의 요청마다 하나의 서비스 프로세스 또는 쓰레드를 사용하는 전통적인 블로킹(blocking) 방식의 서버에서는 서버에 연결된 클라이언트의 수만큼의 서비스 쓰레드들이 서버에 생성되고, 각 쓰레드는 해당 클라이언트로부터의 요청을 기다리는 동안 블로킹 된다. 하지만 이런 방식은 비싼 자원인 쓰레드들을 낭비하게 된다. 반응자 패턴(reactor pattern)[7]은 여러 원천으로부터 발생되는 이벤트들 또는 메시지들을 하나의 처리기에게 모아서 효과적으로 처리하기 위한 소프트웨어 패턴이다. 반응자 패턴은 서버에 연결된 클라이언트들의 다수의 요청들을 모아 하나 또는 소수의 쓰레드들로 처리할 수 있게 하며, 서버의 쓰레드가 클라이언트의 메시지를 기다리는 동안 블로킹되는 현상을 없앨 수 있도록 한다. 반응자 패턴은 서버 클라이언트 구조에서 논블로킹(non-blocking) 서버를 만드는 데 이용되고 있다.

MTSA는 메시지의 수신 및 처리 효율을 추가적으로 높이기 위해 반응자 패턴을 지원한다. 본 논문이 제안하는 구조에 따라 구현된 다중 에이전트 플랫폼 SMAF 2.2는 반응자 패턴을 적용한 소켓 통신을 사용한다. MS Windows, Linux 등의 많은 종류의 운영체계들이 반응자 패턴을 이용한 논블로킹 방식의 입출력을 지원하고 있다. Java 언어는 JDK 1.4 이후부터 반응자 패턴을 지원하고 있다.

### 3.6 비동기 통신의 지원

JADE[10], Jadex[24]와 같은 에이전트 플랫폼은 비동기 통신을 지원하고 있다. 본 논문이 제안하는 구조도 비동기 통신을 지원한다. 제안 구조에서 비동기 통신은 요청 메시지를 송신한 응용 에이전트 *A*가 그에 대한 응답 메시지의 도착을 기다리지 않고 바로 자신에게 필요한 어떤 다른 작업을 수행할 수 있도록 하며, 이후 원하는 시점에 takeoutMsg (Agent *A*, String *t*, long *due_time*) 메서드를 호출하여 응답 메시지 함에서 자신의 요청에 짝이 되는 응답 메시지를 비동기적으로 꺼내갈 수 있도록 한다.

비동기 통신 기법은 응답 메시지 함, 입력 메시지 분배기, 그리고 takeoutMsg(Agent *A*, String *t*, long *due_time*) 메서드를 바탕으로 동작한다. Fig. 5와 Fig. 7의 sendMsg 알고리즘에 따르면, 에이전트 *A*가 sendMsg(*A*, *B*, *m*) 메서드를 호출하여 에이전트 *B*에게 reply-with 필드의 값 *t*가 null이 아닌 요청 메시지 *m*을 보내면, <*t*, null> 레코드가 *A*의

```
Message takeoutMsg(Agent A, String t, long due_time)
{
  while (due_time <= current time) {
    if (there exists a record <t, a message r whose
      in-reply-to field equals t> in the response
      message box of A)
    {
      delete the record <t, the message r> from the
      response message box of A;
      return the message r;
    }
  }
  return null;      /* no response in timeout */
}
```

Fig. 8. Algorithm takeoutMsg
(Agent *A*, ***String t, long due_time)***

응답 메시지 함에 삽입된다. 이후 in-reply-to 필드의 값이 *t*인 응답 메시지 *r*이 에이전트 *A*에게 도착하면 *A*의 입력 레코드가 <*t*, *r*>으로 수정된다. 응용 에이전트 A가 응답 메시지 분배기 쓰레드에 의해 입력 메시지 함의 <*t*, null> 메시지 함에서 응답 메시지 *r*을 꺼내가는 메서드 takeoutMsg(Agent *A*, String *t*, long *due_time*)의 동작은 다음 Fig. 8과 같다.

takeoutMsg(Agent *A*, String *t*, long *due_time*) 메서드는 만일 주어진 *due_time* 시각 이내에 입력 메시지 분배기 쓰레드가 in-reply-to 필드의 값이 *t*인 메시지를 응답 메시지 함에 넣으면 그 응답 메시지를 즉시 반환하고 아니면 *due_time* 이후 null을 반환한다. 비동기 통신을 이용하면, 요청 메시지를 송신한 에이전트가 그에 대한 응답 메시지를 기다리는 동안 발생시킬 수 있는 부담을 줄일 수 있다.

## 4. 성능 평가

본 논문이 제안하는 구조의 성능을 평가하기 위하여 본 논문이 제안하는 구조에 따라 구현된 SMAF(Small Multi-Agent Framewrok) 버전 2.2를 JADE 버전 4.2.0과 동일한 환경에서 동일한 성능 측정 방법으로 성능을 비교하였다. 성능 측정 방법으로는 다중 에이전트 프레임워크인 JADE의 성능 측정을 위하여 사용되었던 RTT(Round Trip Time) 테스트[25]를 그대로 사용하였다.

### 4.1 성능 평가 환경 및 방법

RTT 테스트는 Fig. 9와 같이 송신 에이전트(sender agent)와 수신 에이전트(receiver agent)의 쌍들 간에 수행된다. RTT 테스트는 송신 에이전트가 메시지를 보내고 그 메시지를 받은 수신 에이전트가 다시 송신 에이전트에게 응답 메시지를 돌려주는 데 걸리는 시간들의 평균을 측정하는 테스트이다. 송신 에이전트들과 수신 에이전트들이 총 1쌍, 10쌍, 100쌍, 1,000쌍이 존재하는 경우들을 테스트하였으며, 송신자 에이전트와 수신자 에이전트의 쌍이 1쌍인 경우는 10,000번 메시지 왕복(roundtrip)을 수행하고, 송신자 에이전트와 수신자 에이전트의 쌍이 10쌍, 100쌍, 1,000쌍인 경우는 1,000번의 메시지 왕복을 수행하여 평균을 구했다. RTT 테스트에 사용되는 모든 메시지의 content 필드의 값은 7글자의 짧은 문자열이다.

송신자 에이전트와 수신자 에이전트의 상대적 위치를 다음 3가지 경우로 구분하여 테스트하였다.

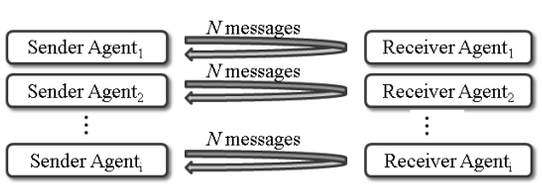


Fig. 9. RTT(Round Trip Time) Test

**[case 1] 1 프로세스, 1 호스트, 1 플랫폼 환경**

송신자와 수신자가 동일 프로세스 내에 존재할 때의 메시지 전송 성능을 비교하기 위하여, JADE의 경우 동일한 컨테이너에 존재하는 송신자와 수신자 쌍들을 사용하였고, SMAF의 경우 동일한 MTSA를 사용하는 송신자와 수신자의 쌍들을 사용하였다. 이 경우, JADE와 SMAF 모두 송신자와 수신자가 동일 프로세스에 존재하게 되며, 송신자가 보낸 메시지는 복사되어 같은 프로세스 내에 존재하는 수신자의 입력 메시지 큐에 바로 전달되게 된다.

**[case 2] 2 프로세스, 1 호스트, 1 플랫폼 환경**

송신자 에이전트와 수신자 에이전트가 동일한 프로세스(즉 동일 JVM)상에 존재하지는 않지만 동일 호스트에 존재하는 경우이다. 이런 경우에 JADE는 송신자 에이전트와 수신자 에이전트가 동일한 플랫폼에 있다면 Java RMI를 사용하며, 다른 플랫폼에 있다면 HTTP 또는 IIOP를 사용하게 된다. SMAF와의 공정한 비교를 위하여 JADE의 경우 보다 빠른 RMI를 사용하는 경우의 성능을 측정하였다. 일반적으로 송신자와 수신자가 동일한 호스트에 존재하면 HTTP 또는 IIOP보다 Java RMI가 빠르다. SMAF의 경우 메시지 전송 프로토콜로 TCP를 사용하였다. TCP는 프로세스간 통신의 경우에도 효과적이며, 반응자 패턴을 적용할 수 있는 프로토콜이기 때문이다.

**[case 3] 2 프로세스, 2 호스트, 2 플랫폼 환경**

FIPA 규격은 HTTP와 IIOP를 플랫폼간 기본 메시지 전송 프로토콜로 규정하고 있다. 송신자 에이전트의 플랫폼과 수신자 에이전트의 플랫폼이 서로 다른 호스트에 존재하는 경우 JADE는 HTTP와 IIOP를 지원한다. JADE는 버전 3.2부터 HTTP를 플랫폼간 기본 메시지 전송 프로토콜로 사용하고 있다. JADE 벤치마크 자료[25]를 보면 Java 언어가 기본 지원하는 ORB(Object Request Broker)를 사용하는 IIOP보다 HTTP가 약 1.7배 정도 성능이 높다. 따라서 이 벤치마크에서는 JADE는 HTTP를 기준으로 성능을 측정하였다. SMAF의 경우 TCP를 사용하였다.

측정에 사용된 기계 및 환경은 다음 Table 1과 같다. 두 개의 기계 Host1과 Host2는 무선 공유기를 통하여 무선랜 카드들로 Wi-Fi로 연결되어 있다. Host1과 Host2에서 RTT 테스트를 위한 충분한 주기억 장치 공간의 확보를 위하여 jvm -Xms512m 옵션을 사용하였다.

Table 1. Benchmarking Environment

| | Host1 (desktop pc) | Host2 (notebook) |
|---|---|---|
| **CPU** | Intel i7-2600K @3.4GHz | i7-3517u @1.9 GHz |
| **RAM** | 4G bytes | 8G bytes |
| **OS** | Windows 7 (32bit) | Windows 7 (64bit) |
| **JDK** | JDK 1.6 | JDK 1.6 |

Table 2. The Results of Benchmarking

| | couples | iterations | JADE 4.2 | | SMAF 2.2 | | average performance ratio (%) |
|---|---|---|---|---|---|---|---|
| | | | average RTT(msec) | average # of RTs /sec | average RTT(msec) | average # of RTs /sec | |
| case 1 | 1 | 10000 | 0.07 | 13608.3 | 0.03 | 32051.3 | 235.5 |
| | 10 | 1000 | 0.32 | 3151.8 | 0.067 | 15197.1 | 482.2 |
| | 100 | 1000 | 3.22 | 310.9 | 0.54 | 1869.2 | 601.1 |
| | 1000 | 1000 | 42.94 | 23.3 | 7.59 | 131.7 | 565.4 |
| case 2 | 1 | 10000 | 0.43 | 2300.5 | 0.12 | 8183.6 | 355.7 |
| | 10 | 1000 | 1.42 | 705.5 | 0.40 | 2483.1 | 351.9 |
| | 100 | 1000 | 13.15 | 76.1 | 2.74 | 365.3 | 480.2 |
| | 1000 | 1000 | 153.93 | 6.5 | 25.74 | 38.9 | 598.1 |
| case 3 | 1 | 10000 | 5.23 | 192.2 | 4.20 | 238.1 | 123.9 |
| | 10 | 1000 | 37.42 | 26.8 | 5.58 | 179.3 | 668.4 |
| | 100 | 1000 | 403.15 | 2.5 | 14.58 | 68.7 | 2768.4 |
| | 1000 | 1000 | 3945.48 | 0.3 | 115.56 | 8.7 | 3414.5 |

### 4.2 성능 평가 및 분석

다음 Table 2는 JADE와 비교하여 SMAF의 RTT 테스트를 수행한 결과를 보여준다. SMAF는 JADE에 비교하여 송신 에이전트와 수신 에이전트가 동일한 프로세스 즉 동일한 JVM 상에 존재할 때 2.3배에서 5.6배 정도의 높은 성능을 보이며, 송신 에이전트와 수신 에이전트가 동일한 호스트에 존재하지만 서로 다른 프로세스 즉 서로 다른 JVM 상에 존재할 때 3.5배에서 약 6배 높은 성능을 보인다. 송신 에이전트와 수신 에이전트의 플랫폼이 서로 다른 호스트에 존재할 때에는 SMAF가 1.2배에서 34배 정도 높은 성능을 보이고 있다.

JADE 4.2와 SMAF 2.2는 두 프레임워크의 특성상 서로 동일한 메시지 전송 프로토콜들의 집합을 지원하고 있지 않으므로, 성능 비교에 동일한 메시지 전송 프로토콜들을 사용하지 못했다. 하지만 Table 2의 성능 평가 결과는 에이전트 메시지 표현 포맷의 세부적 차이와 메시지 전송 프로토콜들 이외에는 동일한 조건에서 이루어진 것이므로 상당한 의미를 갖는다고 할 수 있다. SMAF는 JADE에 비교하여 특히 송신 에이전트와 수신 에이전트의 쌍들의 수가 많아질수록 RTT 테스트에서 더 좋은 성능을 보이고 있으며, 특히 송신 에이전트의 플랫폼과 수신 에이전트가 서로 다른 호스트들에 존재하는 경우 상대적으로 더 좋은 성능을 보이고 있다. 그 이유는 본 논문이 제안하는 에이전트 메시지 전송 구조 및 SMAF가 가진 다음과 같은 특성들 때문으로 판단된다.

(1) SMAF의 응용 에이전트는 자원 소모를 줄이는 가벼운 구조를 갖도록 설계되었다. 실제로 성능 평가에 사용된 SMAF 2.2에서 응용 에이전트의 구현에는 단 하나의 쓰레드가 사용되고 있다. 응용 에이전트가 가벼운 구조를 가지면 결과적으로 에이전트 시스템의 부담이 적어지고 응용 에이전트들의 수가 많아져도 에이전트 시스템의 성능이 덜 줄어든다.

(2) SMAF에서 메시지 전송을 담당하는 MTSA는 에이전트의 한 종류이며 내부적으로 Fig. 6에 표현된 입력 메시지 큐의 버퍼링 능력을 사용하여 메시지의 수신 및 처리의 성능을 높일 수 있도록 설계되었다. 반면에 JADE에서 메시지 전송 서비스를 담당하는 컨테이너는 수신 메시지의 버퍼링을 위한 명시적인 입력 메시지 큐를 사용하지 않는다.

(3) SMAF의 MTSA는 반응자 패턴을 이용해 메시지의 수신시 발생하는 쓰레드들의 블럭킹 문제를 없애고 적은 수의 쓰레드로 높은 메시지 수신 성능을 낸다. 반면에 JADE 4.2는 반응자 패턴을 도입하고 있지 않다고 분석된다. Java RMI는 반응자 패턴을 지원하지 않고 있다.

위 Table 2의 성능 평가 결과에는 반영되지 않았지만, JADE의 경우 송신자와 수신자의 컨테이너가 다르면 그들이 동일 프로세스에 존재하는 경우에도 전송 메시지 객체의 직렬화 및 역직렬화 절차를 수반하는 상대적으로 느린 Java RMI 통신이 사용된다. 하지만 본 논문이 제안하는 구조에서는 에이전트들이 동일 프로세스에 존재하는 경우에 항상 프로세스 내 통신을 통해 신속하게 메시지 전송이 처리된다.

## 5. 결론 및 향후 연구

본 논문은 에이전트들 간의 긴밀한 협력을 필요로 하는 응용 분야들을 지원하기 위한 다중 에이전트 메시지 전송 구조를 제안하고 제안된 구조에 따라 구현된 다중 에이전트 프레임워크 SMAF 2.2의 성능을 잘 알려진 다중 에이전트 프레임워크인 JADE 4.2.0의 성능과 비교하였다. 성능 평가 결과 SMAF는 JADE보다 명확히 높은 성능을 보인다.

본 논문이 제안하는 에이전트 메시지 전송 구조 및 그것을 기반으로 구현된 다중 에이전트 프레임워크 SMAF는 FIPA 표준 규격을 완전하게 만족하는 것을 목표로 설계되지 않았지만, 그것으로부터 많은 것들을 참조하여 설계되었으며, 부분적으로 FIPA 표준 에이전트 규격을 만족한다. 제안된 다중 에이전트 구조는 FIPA 표준 에이전트 플랫폼 규격에

서 메시지 전송 서비스를 담당하는 ACC(Agent Communication Channel)를 MTSA(Message Transfer Service Agent)라는 에이전트들의 집합으로 표현한다. 그 MTSA들은 비동기 메시지 통신을 지원하여 응답 메시지 수신의 부담을 줄이며, 반응자 패턴을 사용하여 에이전트 메시지 수신에 필요한 쓰레드 자원의 소모를 줄이고, 송신 에이전트와 수신 에이전트의 상대적 위치에 따라 프로세스 내 통신, 프로세스 간 통신, 그리고 호스트 간 통신을 선택하여 에이전트들 간의 긴밀한 협력이 가능하도록 한다. 다중 에이전트 구조의 메시지 통신 속도가 고속화되면 될수록 그 다중 에이전트 구조는 더 긴밀한 협력을 필요로 하는 더 다양한 응용 분야들에 적용될 수 있을 것이다.

향후 연구로는 에이전트들의 메시지 전송 성능을 더욱 향상하기 위하여 MTSA가 사용할 수 있는 메시지 전송 프로토콜들을 다양화하는 연구가 필요하다. 예를 들어, 동일 호스트상의 서로 다른 프로세스에 존재하는 에이전트들의 메시지 전송을 위하여 UNIX Socket이나 파이프를 사용하는 것을 검토할 수 있다. 또한 다중 에이전트 기술을 모바일 환경에 적용하기 위해 필요한 기술들에 대한 연구가 필요하다. 제안된 에이전트 구조에 표현된 개념들은 FIPA 표준 에이전트 규격을 만족하는 에이전트 프레임워크뿐 아니라 FIPA 표준을 만족하는 것을 목표로 하지 않는 다중 에이전트 프레임워크를 개발할 때에도 사용될 수 있을 것이다.

## Reference


[1] Jacques Ferber, Multi-Agent System: *An Introduction to Distributed Artificial Intelligence*, Addison Wesley Longman, Feb., 25, 1999.

[2] Anand S. Rao, Michael P. Georgeff, "BDI Agents: From Theory to Practice", in *Proceedings of the First International Conference on Multi-Agent System*, San Francisco, CA, pp.312-319, 1995.

[3] Gowang-Lo Lee, Sang-Kyu Park, Myong-Wuk Jang, Byung-Eui Min, Joong-Min Choi, "A Method of Extending a Multiagent Framework with a Plan Generation Module", *The Transactions of the Korea Information Processing Society*, Vol.4, No.9, 1997.

[4] Jennings, Nicholas R., "An Agent-Based Approach for building Complex Software Systems", *Communications of the ACM*, 44(4), pp.35-41, April, 2001.

[5] Daniel D. Corkill, "Collaborating Software Blackboard and Multi-Agent Systems & the Future", In *Proceedings of the International Lisp Conference*, New York, October, 2003.

[6] FIPA Agent Message Transport Service Specification [Internet], http://www.fipa.org/specs/fipa00067/SC00067F.pdf.

[7] Schmidt, Douglas et al, *Pattern-Oriented Software Architecture Volume 2: Patterns for Concurrent and Networked Objects*, Wiley, 2000.

[8] Foundation for Intelligent Physical Agents [Internet], Specifications, http://www.fipa.org.

[9] Java Agent DEvelopment framework [Internet], http://jade.tilab.com/

[10] Patil RS, Fikes RE, Patel-Scheneider PF, McKay D, Finn T, Gruber T, Neches R, "The DARPA knowledge sharing effort: Progress report", in *Proceedings of the Third Conference on Principles of Knowledge Representation and Reasoning*, Cambridge, MA, 1992, pp.103-114.

[11] Object Management Group Middleware Specifications [Internet], http://www.omg.org/spec/#MW.

[12] FIPA ACL Message Structure Specification [Internet], http://www.fipa.org/specs/fipa00061/SC00061G.pdf.

[13] FIPA Agent Management Specification [Internet], http://http://www.fipa.org/specs/fipa00023/SC00023K.pdf.

[14] FIPA-OS [Internet], http://www.nortelnetworks.com/products/announcements/ fipa/index.html.

[15] The Grasshopper [Internet], http://www.ikv.de/products/grasshopper.

[16] SoonCheol Baeg, Joong Min Choi, Myeong Wuk Jang, Sang Kyu Park, Young Whan Lim, "A Framework for Multi-agent Systems Supporting Cooperation between Heterogeneous Agents", *The Journal of The Korean Institute of Information Scientists and Engineers*, Vol.2,1, pp.24-37, March, 1996.

[17] FIPA Agent Message Transport Protocol for IIOP Specification [Internet], http://www.fipa.org/specs /fipa00075/SC00075G.pdf

[18] FIPA Agent Message Transport Protocol for HTTP Specification [Internet], http://www.fipa.org/specs/fipa00084/SC00084F.pdf

[19] Fabio Bellifemine, Agostino Poggi, Giovanni Rimassa, "JADE - A FIPA-compliant agent framework", in *Proceedings of PAAM'99*, London, April, 1999, pp.97-108.

[20] Java Remote Method Invocation Home [Internet], http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html.

[21] Microfocus ORBACUS Object Request Broker [Internet], http://www.microfocus.com/products/corba/orbacus/index.aspx

[22] Edward Curry, Desmond Chambers, Gerald Lyons, "A JMS Message Transport Protocol for the JADE Platform", in *Proceedings of IEEE/WIC International Conference on Intelligent Agent Technology*, Oct., 2003, pp.596-600.

[23] Finin T, Labrou Y. "KQML as an agent communication language", in *Software Agents*, Bradshaw JM (eds.). MIT Press: Cambridge, MA, 1997, pp.291-316.

[24] Jadex BDI Agent System [Internet], http://sourceforge.net/projects/jadex/.

[25] E. Cortese, F. Quarta, G. Vitaglione, "Scalability and Performance of JADE Message Transport System", in *Proceedings of AAMAS Workshop on AgentCities*, Bologna, 2002.

장 혜 진

e-mail : hjchang@smu.ac.kr

1994년 서울대학교 계산통계학과(전산학 전공, 박사)

1994년~현 재 상명대학교 소프트웨어공학과 교수

관심분야 : 분산 제어 시스템, 로봇 제어, 다중 에이전트 시스템