통합 에이전트 구축 언어를 지원하는 지능형 에이전트 쉘의 개발

장 혜 진†

요 약

여러 종류의 기존의 다중 에이전트 프레임웍들(multi-agent frameworks)이 에이전트의 지적인 능력의 표현을 위하여 고수준의 지식 표현 언어를 지원한다. 하지만 그들의 에이전트 프로그래밍 인터페이스는 지식 표현 언어 뿐 아니라 어떤 다른 범용의 프로그래밍 언어들의 사용을 요구한다. 일반적으로 고수준 지식 표현 언어와 범용의 프로그래밍 언어간에는 언어의 수준 및 자료 표현 모델에 있어서 상당한 차이가 있으며, 그런 차이는 지능형 에이전트의 개발에 필요한 요소들의 결합에 관련된 문제점들을 발생시킨다. 본 논문은 그런 문제점들의 극복을 위해 개발한 새로운 유형의 지능형 에이전트 쉘 INAS(INtelligent Agent Shell) 버전 2에 대한 것이다. 지능형 에이전트의 개발을 위하여 고수준의 지식 표현 언어와 범용 프로그래밍 언어를 결합하여 사용해야 하는 기존의 에이전트 프레임웍들과 달리 INAS 버전 2는 그 자체만으로 에이전트들을 구축할 수 있는 고수준의 통합 에이전트 구축 언어를 지원한다. 따라서 INAS 버전 2를 사용한 지능형 에이전트의 개발은 지능형 에이전트의 개발에 필요한 요소들의 결합에 관련된 기존의 에이전트 프레임웍들의 문제점들을 겪지 않는다. 몇 종류의 지능형 에이전트들의 개발을 통하여 INAS 버전 2의 통합 에이전트 구축 언어가 지능형 에이전트들의 개발에 효과적임을 경험할 수 있었다.

Development of An Intelligent Agent Shell Supporting An Integrated Agent Building Language

Hai Jin Chang†

ABSTRACT

There are many kinds of multi-agent frameworks which support the high-level knowledge representation languages for providing intelligence to their agents. But, the agent programming interfaces of the frameworks require to use some general-purpose programming languages as well as the knowledge representation languages. In general, knowledge representation languages and general-purpose programming languages are different in their levels and data representation models. The differences can make the problems about the coupling of the elements which are necessary for developing intelligent agents. This paper describes a new type of intelligent agent shell INAS(INtelligent Agent Shell) version 2 which has developed to cope with the problems. Unlike the previous agent frameworks, INAS supports a high-level integrated agent building language for building intelligent agents by itself. Therefore, the development of intelligent agents by using INAS version 2 does not suffer from the problems of the previous agent frameworks. Through the development of several intelligent agents, we experienced that the agent building language of INAS version 2 could reduce the difficulties of developing intelligent agents.

※ 본 논문은 상명대학교 '99학년도 교내학술연구비 지원에 의하여 연구되었음.

† 정 회 원 : 상명대학교 컴퓨터정보통신학부 교수

논문접수 : 1999년 5월 13일, 심사완료 : 1999년 11월 8일

1. 서 론

에이전트 시스템은 인공 지능, 데이터 통신, 병렬 시스템 연구, 로봇 연구 그리고 사용자 인터페이스 설계 등의 다양한 분야의 연구 개발자들에게 점차 중요한 주제가 되고 있으며, 많은 에이전트 연구 개발자들에게 지적 능력은 에이전트들이 가져야 하는 중요한 특성으로 인식되고 있다[1]. 에이전트 시스템이 지적 능력을 갖는다면 바람직할 것이다. 에이전트 시스템이 지적 능력을 갖게 되면, 에이전트 시스템을 구성하는 자원들을 보다 효과적으로 이용할 수 있으며, 에이전트 시스템에게 주어진 문제들에 대하여 보다 융통성 있는 문제 해결 능력을 제공할 수 있을 것이다. 또한 사용자에게 보다 편리한 사용자 인터페이스를 제공할 수도 있을 것이다.

지능형 에이전트 프레임웍(intelligent agent framework) 또는 지능형 에이전트 쉘(intelligent agent shell)은 지능형 에이전트 시스템들의 구축에 필요한 여러 가지 요소들을 잘 규정된 어떤 형태로 제공하여 고품질의 지능형 에이전트 시스템의 구축에 필요한 노력과 시간을 감소시키기 위한 시스템이다. 지능형 에이전트 프레임웍 또는 지능형 에이전트 쉘의 도움을 받지 않고 지능형 에이전트 시스템을 구축한다면 에이전트 통신 언어, 사용자 인터페이스, 추론 엔진 등을 설계하고 구현하기 위하여 많은 시간과 노력이 필요할 것이다. 또한 결과적으로 구축된 지능형 에이전트 시스템의 품질이 만족스럽지 않을 수도 있을 것이다.

지능형 에이전트 시스템들에게는 에이전트간 통신 기능이나 에이전트 생명 주기 관리 기능과 같은 다중 에이전트 시스템 관련 고유 기능들 뿐 아니라 지적인 행위를 할 수 있는 능력이 요구된다. 지적인 능력을 제공하기 위한 전통적이며 자주 사용되고 있는 방법들 중의 하나는 광의의 지식 표현 언어를 사용하는 것이라고 할 수 있다. 예를 들어, 규칙 기반 시스템들은 경험적 지식을 표현하기 위하여 규칙들을 사용하며, 계획 수립기들은 계획 수립을 위한 다양한 종류의 고수준의 연산자들을 사용한다. 여러 종류의 기존의 에이전트 프레임웍들이 지적인 능력의 제공을 위하여 논리나 생성 규칙과 같은 고수준의 지식 표현 언어를 사용하고 있다.

지식 표현 언어와 범용 프로그래밍 언어간에는 언어의 수준 및 자료 표현 모델에 있어서 상당한 차이가 존재한다. 지식 표현 언어는 심벌(symbol), 사실, 중첩 리스트(nested list)와 같은 고수준의 자료 구조들의 직접적인 표현을 지원하지만 범용 언어들은 문자열, 구조체, 포인터와 같은 상대적으로 저수준의 자료 표현 방식을 사용한다. 또한 지식 표현 언어는 일반적으로 인터프리터의 방식이지만 범용 언어들은 대개 컴파일러 방식을 사용한다. 따라서, 고수준의 지식 표현 언어 뿐 아니라 C/C++ 언어나 Java 언어와 같은 범용의 프로그래밍 언어의 사용을 함께 요구하는 에이전트 개발 인터페이스를 제공하는 기존의 지능형 에이전트 프레임웍들은 다음과 같은 문제점들을 갖는다.

- 범용 프로그래밍 언어의 사용을 요구하는 기능들을 사용하려면, 새로운 에이전트 시스템을 개발할 때마다, 프로그램을 작성하여 컴파일하고 에이전트 통신 라이브러리나 추론 엔진 라이브러리와 결합(link)하는 과정을 반복해야 한다. 이런 과정은 간단한 에이전트를 개발하는 동안에도 여러 번 반복될 수 있다.
- 고수준 지식 표현 언어의 사용이 필요한 경우에는, 지능형 에이전트 시스템을 완성하려면 범용 프로그래밍 언어를 이용한 프로그래밍 뿐 아니라 추론 엔진이나 계획 수립기 모듈에 의해 사용될 지식 베이스의 프로그래밍이 추가적으로 필요하다.
- 고수준 지식 표현 언어와 범용 프로그래밍 언어와는 언어의 수준 및 자료 표현 모델에 상당한 차이가 있으므로 두 언어들의 결합에 관련된 문제가 발생할 수 있다. 이 문제는 데이터베이스 분야에서 SQL과 같은 고수준 질의 언어와 범용 호스트 언어간의 결합에 관련된 문제인 임피던스 불일치(impedance mismatch) 문제[2]에 비유될 수 있다.

INAS는 현재 버전 2까지 개발되었다. 본 논문의 저자들은 버전 1의 INAS의 개발 및 그것을 사용한 지능형 에이전트들의 개발 경험[3]을 통하여 위와 같은 문제점들을 경험하였으며 그런 경험을 반영하여 INAS 버전 2를 개발하였다. INAS 버전 1과 2의 가장 큰 개념적 차이점은 에이전트 개발 인터페이스에 있다. 버전 1의 개발 인터페이스는 기존의 에이전트 프레임웍들과 마찬가지로 범용 프로그래밍 언어와 지식 표현 언어를 결합하여 사용하는 것을 요구하지만, 버전 2는 고수준의 통합 에이전트 구축 언어만을 이용한 에이전트의 개발을 지원한다. 즉, 버전 2는 수준과 자료 표현 모델이 다른 두 언어의 결합에 관련된 문제점들을 겪

지 않는다. 그 문제점들은 버전 1의 INAS 만이 갖는 특수한 문제점들이 아니라 고수준의 지식 표현 언어와 범용 언어를 결합하여 사용해야 하는 기존의 지능형 에이전트 프레임웍들이 갖는 일반적인 문제점들이라고 할 수 있다.

본 논문은 위와 같은 문제들을 극복하기 위한 방안으로 설계되고 구현된 통합 에이전트 구축 언어를 지원하는 INAS 버전 2의 개발에 대한 것이다. INAS 버전 2의 에이전트 구축 언어를 사용하면 지식 표현 기능 뿐 아니라 에이전트 관련 고유 기능들 및 기타 기능들을 통합적으로 표현하는 것이 가능하므로 새로운 지능형 에이전트를 효과적으로 구축할 수 있다. 즉, 에이전트 구축 언어로 작성된 확장된 의미의 지식 베이스들을 구축하는 것만으로 새로운 지능형 에이전트 시스템을 구축할 수 있다. INAS 버전 2의 에이전트 구축 언어는 고수준이며, 인터프리터 방식을 사용하며, 확장된 지식 표현 언어의 형태를 갖는다. 지능형 에이전트 시스템을 개발하기 위한 전용의 통합 에이전트 구축 언어의 설계와 구현에 대한 연구 결과들은 아직 충분하지 않다고 사료된다.

본 논문은 다음과 같이 구성된다. 제 2장에서는 관련 연구들에 대하여 다룬다. 제 3장에서는 INAS가 지원하는 지능형 에이전트 구축 언어에 관련된 구조와 알고리즘 등을 다룬다. 제 4장에서는 INAS의 활용에 대한 내용을 다룬다. 제 5장에서는 결론을 맺는다.

2. 관련 연구

다중 에이전트 시스템의 효과적인 구축의 지원 및 이론적 탐색을 위하여 다양한 에이전트 프레임웍들이 연구되고 있다[4]. 본 장에서는 다중 에이전트 프레임웍들에서의 고수준의 지식 표현 언어의 지원에 관련된 내용들을 중심으로 기존 연구들을 검토한다.

에이전트 프레임웍들은 고수준의 지식 표현 언어를 지원하는 것들과 그렇지 않은 것들로 분류될 수 있다. 지식 표현 언어를 지원하는 기존의 에이전트 프레임웍들에는 ABE(Agent Building Environment)[5], EMAF (Extensible Multi-Agent Framework)[6], OAA(Open Agent Architecture)[7], DYNACLIPS[8], AGENT_CLIPS [9], 그리고 KAPICLIPS[10] 등이 존재한다. ABE는 규칙 기반 추론 기능을 제공한다. EMAF와 OAA는 논리 기반의 선언적 형태의 에이전트간 통신 언어를 제공한다. DYNACLIPS, AGENT_CLIPS, 그리고 KAPICLIPS는 생성 규칙을 사용한다. 고수준 지식 표현 언어를 지원하는 기존의 에이전트 프레임웍들은 지식 표현 언어를 사용하여 지적인 능력을 표현할 수 있다는 장점을 갖지만 에이전트 통신이나 사용자 인터페이스 등의 개발을 위하여 지식 표현 언어 이외의 범용의 언어들을 함께 사용해야 하므로 고수준 지식 표현 언어와 범용 프로그래밍 언어의 결합에 관련된 문제점들을 갖고 있다.

INAS 버전 1과 2는 모두 CLIPS(C Language Integrated Production System)[11]의 추론 엔진을 활용하여 규칙 기반 추론 기능을 제공한다. CLIPS를 활용한 이유는 CLIPS가 공개되어 있으며, 높은 성능 및 확장성을 제공하기 때문이다. CLIPS를 활용하는 기존의 지능형 에이전트 프레임웍들에는 DYNACLIPS, AGENT_CLIPS, KAPICLIPS 그리고 JESS(Java Expert System Shell)[12]등이 있다. DYNACLIPS는 SunOS 플랫폼에서 운영되며, 블랙보드(blackboard) 구조를 통해 여러 에이전트들간의 협동을 제어한다. AGENT_CLIPS는 인터넷상에서 동작되는 지능형 에이전트를 구현하기 위한 시스템이다. AGENT_CLIPS는 기존의 CLIPS에 뉴스 서버와 웹 서버를 연결할 수 있는 기능을 추가한 것이다. AGENT_CLIPS를 이용하면 뉴스 그룹과 웹 페이지를 검색할 수 있는 지능형 에이전트를 쉽게 개발할 수 있다. KAPICLIPS는 CLIPS의 응용 프로그래밍 인터페이스를 확장하여 가장 대표적인 에이전트간 통신 언어인 KQML(Knowledge Query and Manipulation Language)[13]을 지원한다. JESS는 C 언어로 구현된 CLIPS의 핵심 모듈들을 Java 언어로 다시 구현한 것이다. JESS는 다중 에이전트 환경을 직접 지원하는 기능들을 제공하고 있지 않지만 애플릿(applet) 형태의 지능형 에이전트들을 개발하는데 사용될 수 있다는 특징을 갖는다.

앞에서 언급한 CLIPS를 활용하는 기존의 지능형 에이전트 프레임웍들과 INAS 버전 2의 차이점들은 다음과 같다. 첫째, KAPICLIPS를 제외한 기존의 다른 시스템들은 고수준이며 계층화된 형태의 메시지를 사용하는 에이전트 통신 명령들을 제공하지 않는다. 둘째, 다른 시스템들은 INAS와 달리 표준화된 에이전트 참조 모델을 고려하여 설계되지 않았다. INAS는 버전 1부터 에이전트 기술의 표준화를 위해 활동하는 대표적인 기구의 하나인 FIPA(Foundation for Intelligent

Physical Agents)의 에이전트 참조 모델[14]을 사용하고 있다. 에이전트 참조 모델은 에이전트들이 존재하고 활동하는 환경 및 기준을 제공한다. 셋째, 다른 시스템들은 INAS 버전 2와 달리 통합 에이전트 구축 언어를 제공하지 않으므로 C/C++ 언어와 지식 표현 언어의 결합에 관련된 문제점들을 갖는다. 즉, CLIPS를 활용하는 기존의 다중 에이전트 프레임웍들에서는 새로운 에이전트를 작성할 때마다 새로운 지식 베이스를 구축하고 C 언어 등으로 작성된 코드들을 작성해야 할 뿐 아니라 지식 베이스와 C 코드들의 결합을 고려해야 한다. INAS 버전 2에서는 C 언어로 작성된 함수를 INAS 버전 2가 지원하는 에이전트 구축 언어의 내장 함수로 만들어야 하는 경우를 제외하고는 통합 에이전트 구축 언어만을 사용하여 새로운 에이전트 시스템을 구축할 수 있다.

지능형 에이전트 프레임웍이라는 용어는 지능형 에이전트 시스템을 구현하는 데 필요한 다양한 요소들을 갖추어 제공하는 틀이라는 의미로 사용된다. 에이전트 쉘이란 용어는 에이전트 프레임웍의 의미에 포함되는 의미를 갖는다. 하지만, 에이전트 쉘이란 용어는 에이전트 프레임웍의 의미 뿐 아니라 에이전트 응용 영역에 관련된 내용만을 채우면 새로운 응용 에이전트를 완성할 수 있도록 준비된 시스템이란 세부적인 의미를 반영한다고 할 수 있다.

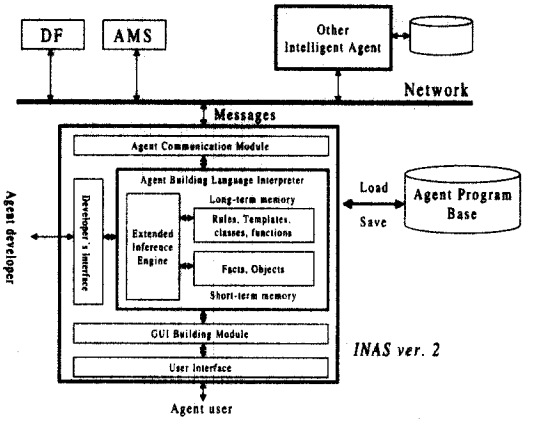
고수준의 지식 표현 언어를 지원하는 기존의 에이전트 프레임웍들은 그렇지 않은 에이전트 프레임웍들에 비하여 지적인 능력을 부여하는 데 유리하다. 하지만, 버전 1의 INAS 시스템을 비롯한 고수준 지식 표현 언어를 지원하는 기존의 다중 에이전트 프레임웍들은 서론에서 언급한 문제점들을 갖고 있다. INAS 버전 2는 그런 문제들을 극복하기 위한 대안으로 통합 에이전트 구축 언어를 지원하므로 새로운 유형의 지능형 에이전트 프레임웍이라고 할 수 있다.

3. INAS

지능형 에이전트란 센서(sensor)를 통하여 외부 환경을 인지하고 내부적인 지적 판단에 의해 적절한 이펙터(effector)를 선정하여 환경에 대한 합리적인 동작을 수행하는 시스템이라고 정의될 수 있다[15]. INAS 버전 2의 에이전트 구축 언어를 이용하면 그래픽 사용자 인터페이스를 통한 사용자로부터의 직접적인 명령이나 다른 에이전트로부터의 메시지 등으로부터 외부 환경을 인지하고, 수행 결과의 화면 출력이나 다른 에이전트의 질의 메시지에 대한 응답 메시지 송신 등을 통하여 외부 환경에 대한 동작을 수행하는 지능형 에이전트를 개발할 수 있다. 적절한 이펙터의 선정을 위한 지적 판단 과정은 생성 규칙 등으로 표현될 수 있다.

3.1 INAS의 시스템 구조 및 수행 환경

(그림 1)은 INAS 버전 2의 시스템 구조 및 그것을 이용하여 개발된 에이전트 시스템의 수행 환경을 보여준다. (그림 1)의 좌측 하단부의 커다란 박스가 INAS를 나타낸다.



(그림 1) INAS의 시스템 구조 및 지능형 에이전트 수행 환경

INAS를 구성하는 핵심 모듈은 에이전트 구축 언어 인터프리터(agent building language interpreter) 모듈과 에이전트 통신 모듈(agent communication module), 그래픽 사용자 인터페이스 구축 모듈(GUI building module)이다. INAS는 에이전트 시스템의 개발을 위한 시스템이므로, INAS는 에이전트 개발자를 위한 개발자 인터페이스(developer's interface)를 제공할 뿐 아니라, 에이전트 개발자가 개발한 에이전트를 사용할 에이전트 사용자를 위한 그래픽 방식의 사용자 인터페이스(user interface)를 제공할 수도 있다. INAS로 개발된 에이전트는 경우에 따라 그래픽 사용자 인터페이스를 갖지 않을 수도 있다.

INAS는 FIPA 에이전트 참조 모델[14]이 규정하는 다중 에이전트 환경에서 동작하며, 시스템 관리를 위

한 관리 에이전트들인 DF(Domain Facilitator), AMS (Agent Management System) 등을 사용한다. DF와 AMS는 네트웍 상의 위치와 기능이 잘 알려져 있다고 가정된다. AMS는 에이전트의 생성, 수행, 소멸과 같은 에이전트 생명 주기(life cycle)를 관리한다. DF는 응용 에이전트들의 서비스 기능들이 등록되는 곳이다. 모든 응용 에이전트들은 다른 에이전트들이 자신이 제공하는 서비스 기능들을 쉽게 이용할 수 있도록 자신의 서비스 기능들을 DF에게 등록해야 한다. 서비스 기능들의 등록은 에이전트간 통신 기능을 이용하여 동적으로 이루어진다. 에이전트들은 서로 다른 호스트 기계에 존재할 수 있다. 모든 에이전트들은 호스트 이름과 에이전트 이름에 의해 서로 구분되어야 한다.

INAS를 이용하여 구축된 응용 에이전트는 사용자에 의해 직접 수행이 시작되거나 AMS에게 그 응용 에이전트의 수행을 요청하는 메시지를 보내는 방법에 의해 수행이 시작된다. 물론 응용 에이전트들이 수행되기 전에 관리 에이전트들인 DF와 AMS는 수행 상태에 있어야 한다. 응용 에이전트는 수행이 시작되면 자신을 초기화하고 기본 에이전트 프로그램들을 적재한다. 기본 에이전트 프로그램들이란 INAS 버전 2를 이용하여 구축한 지능형 에이전트의 수행이 시작될 때 자동으로 적재되도록 미리 정해져 있는 에이전트 프로그램들을 의미한다. INAS 수행 파일과 같은 디렉토리에 존재하는 agentutil.clp 프로그램과 inas.clp 프로그램이 기본 에이전트 프로그램들이다. 기본 에이전트 프로그램들은 응용 에이전트의 고유 이름이나 사용 포트에 대한 지정, 메시지 처리기 함수(message handler function), 그리고 응용 에이전트들에게 공통적으로 필요할 만한 유용한 함수들을 미리 제공하기 위한 것이다. 기본 에이전트 프로그램들의 적재가 끝난 응용 에이전트는 사용자나 다른 에이전트 시스템들의 요구의 종류에 따라 필요한 영역 에이전트 프로그램들을 동적으로 적재하고 수행하여 사용자나 다른 에이전트들에게 자신의 응용 영역에 대한 서비스들을 제공할 수 있다. 에이전트 프로그램 베이스(agent program base)란 기본 에이전트 프로그램들과 응용 영역에 대한 프로그램들이 저장된 보조 기억 장치 공간을 의미한다.

3.2 에이전트 구축 언어 인터프리터

에이전트 구축 언어 인터프리터는 사실, 규칙, 함수 등으로 구성된 에이전트 프로그램들을 주기억 장치로 적재하여 수행한다. INAS 버전 2가 지원하는 통합 에이전트 구축 언어는 Rete 알고리즘[16]을 사용하는 강력한 생성 규칙 시스템인 CLIPS의 기능과 문법을 확장하여 다중 에이전트 시스템의 구축에 필요한 기능들을 통합하는 방식으로 개발되었다. INAS 버전 2의 통합 에이전트 구축 언어는 반복, 조건 분기, 입출력 등의 다양한 제어 구조 및 사실, 객체, 생성 규칙과 같은 고수준 지식 표현 명령들 뿐 아니라 다중 에이전트간 통신 명령, 그래픽 사용자 인터페이스 구축 명령 등을 지식 표현 명령들과 같은 수준에서 지원하도록 설계되었다.

에이전트 구축 언어 인터프리터의 추론 엔진은 사용자로부터 직접 입력된 명령들이나 에이전트 프로그램 베이스로부터 적재된 에이전트 프로그램들을 수행한다. 추론 엔진은 에이전트의 현재 상태를 반영하는 사실들이나 객체들을 저장하는 임시 기억 공간(short-term memory)과 규칙, 사실들에 대한 템플릿(template), 클래스, 함수 정의들을 저장하는 장기 기억 공간(long-term memory)을 사용한다. 에이전트 프로그램 베이스는 보조 기억 장치 공간에 존재하지만 단기 기억 공간과 장기 기억 공간은 주기억장치 공간에 존재한다. 에이전트 프로그램 베이스의 에이전트 프로그램들은 사용자의 명령 또는 다른 에이전트의 요청에 의해서 뿐만 아니라, 에이전트 자체의 판단에 의해 수행 중에 동적으로 적재될 수 있다. 에이전트의 수행 중에 생성되는 사실들이나 객체들은 단기 기억 장소에 저장된다.

3.3 INAS의 에이전트 통신 기능

INAS의 통합 에이전트 구축 언어는 고수준이며, 계층화되었으며, 효율적인 통신 명령들을 제공한다. 에이전트 통신 모듈은 관리 에이전트들 및 응용 에이전트들과 지식, 명령 등을 메시지의 형태로 교환하기 위한 기반 기능을 제공한다. INAS 버전 2의 통신 명령들의 처리를 위한 자료 구조와 알고리즘들은 간결하지만 다중 쓰레드와 비동기 통신과 같은 효과적인 통신 기능들을 제공하고 있다.

INAS의 에이전트 통신 기능의 구조와 알고리즘은 다중 에이전트 프레임웍인 DCAF(Distributed Collaborative Agent Framework)[19]를 많은 면에서 참조하고 이용하였다. 하지만, INAS 버전 2의 통신 기능들은 DCAF의 여러 가지 문제점들을 개선하였으며, INAS 버전 2는 DCAF 자체보다는 DCAF가 참조한 FIPA 에이전트

참조 모델에 근거를 둔다고 할 수 있다. DCAF와 INAS의 통신 기능의 차이는 다음과 같다. 첫째, INAS의 통신 명령들은 DCAF가 제공하는 통신 명령들보다 고수준이며 단순하다. 둘째, INAS의 통신 명령들의 내용 계층 인자의 형태는 DCAF에서의 내용 계층 인자의 형태보다 제약이 적다. 셋째, DCAF와 달리 INAS 버전 2는 안정적인 비동기식 에이전트 통신 기능을 제공한다. DCAF는 시간 간격이 긴 비동기 통신을 잘 지원하지 못한다. 넷째, INAS에서는 수신자 지정의 오류 등의 이유로 잘못 보내진 메시지의 발견 및 처리가 수월하다.

INAS의 에이전트간 통신 명령들은 완전하지는 않지만 KQML 언어의 부분 집합을 구현하고 있다. 통신에 관련된 핵심적인 요소들은 sendMessage 명령, waiting-Message 명령, 그리고 메시지 처리기 함수 agentMsg-Handler이다.

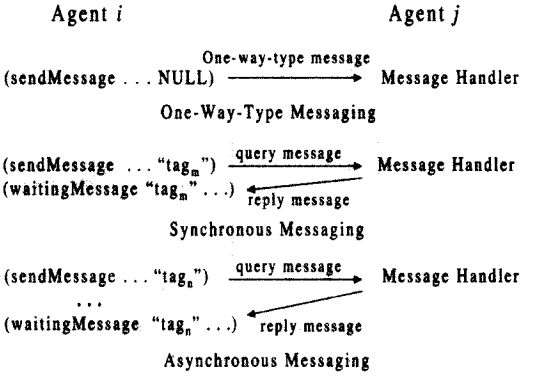
- (sendMessage <receiver> <performative> <content> <reply tag>)
- (waitingMessage <reply tag> <duration>)
- (agentMsgHandler (<performative> <content> <sender> <host> <port> <reply tag>) <handler body>)

sendMessage 명령은 다른 에이전트에게 메시지를 송신하는 명령이다. <receiver>는 수신 에이전트를 지정하는 인자이며, <performative>는 메시지의 역할(speech act)을 나타낸다. <performative>의 종류에는 TELL, REPLY, ASKONE 등이 존재한다. <content>는 실제로 전달하려는 내용을 담기 위한 인자이다. 다중 에이전트 환경에서는 여러 에이전트들이 동시에 여러 메시지들을 보낼 수도 있고 질의를 위한 메시지들을 보낸 순서와 그에 대한 응답 메시지들이 도착하는 순서가 틀려질 수도 있다. 따라서 도착한 응답 메시지가 어떤 질의 메시지에 대한 응답 메시지인가를 구분하는 방법이 필요하다. <reply tag>는 그 구분을 위한 인자이다.

다른 에이전트로부터의 메시지의 수신은 두 경로를 통하여 이루어진다. 하나는 메시지 처리기 함수로 수신하는 방법이고, 다른 하나는 waitingMessage 명령으로 수신하는 방법이다. 메시지 처리기 함수는 다른 에이전트들로부터의 일방향 메시지나 질의 메시지들을 수신하여 처리하는 함수이다. 메시지 처리기 함수는 인자로 전달된 수신 메시지의 내용을 분석하여 <performative>의 종류 및 <content>의 내용에 따라 메시지를 처리한다. 메시지 처리기 함수로 수신된 메시지들의 처리 방식은 메시지 처리기 함수의 몸체인 <handler body>를 어떻게 프로그래밍 하느냐에 따라 달라진다.

3.3.1 지원되는 통신 방식들

에이전트 통신 모듈은 sendMessage 명령, waiting-Message 명령, 그리고 메시지 처리기 함수를 이용하여 일방향 통신(one-way-type messaging) 및 동기식 통신(synchronous messaging)과 비동기식 통신(asynchronous messaging)을 지원한다. (그림 2)는 에이전트간 통신 방식들에 대한 그림이다.



(그림 2) 에이전트간 통신 방식의 종류

일방향 통신이란 다른 에이전트에게 일방적으로 어떤 메시지를 보내는 것을 의미한다. 일방향 메시지를 송신한 에이전트는 그 메시지에 대한 응답 메시지를 기다리지 않는다. 일방향 통신 메시지의 송신은 <reply tag>의 값으로 NULL 심벌을 사용하는 sendMessage 명령으로 표현된다. 일방향 통신으로 보낸 메시지는 수신 에이전트의 메시지 처리기 함수의 인자들로 전달된다.

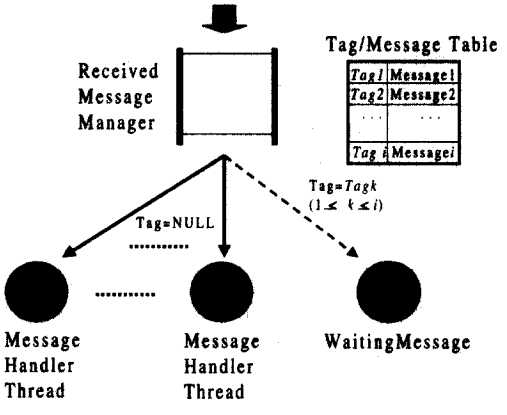
동기식 통신과 비동기식 통신은 양방향 통신이다. 양방향 통신은 다른 에이전트에게 어떤 질의를 보내고 그 응답을 받기 위한 통신의 형태이므로 양방향 통신에서 송수신되는 메시지들을 각각 질의 메시지(query message)와 응답 메시지(reply message)로 표기한다. 동기식 통신이란 다른 에이전트에게 질의 메시지를 보내고 응답 메시지가 올 때까지 기다리는 방식의 통신을 의미한다. 동기식 통신은 NULL 심벌이 아닌 동일한 <reply tag>를 사용하는 sendMessage 명령과 waiting-

Message 명령을 연속하여 사용하여 표현된다. 비동기식 통신이란 sendMessage 명령을 사용하여 다른 에이전트에게 질의 메시지를 보내고 어떤 다른 작업을 수행한 후 waitingMessage 명령을 수행하여 응답 메시지를 수신하는 방식의 통신을 의미한다. 비동기식 통신의 sendMessage 명령과 waitingMessage 명령도 NULL이 아닌 동일한 <reply tag>를 사용하여 표현된다.

네트웍 상에 분산된 에이전트들간의 통신은 통신 회선의 속도, 응답 에이전트의 계산 속도 등에 따라서 발생하는 지연을 고려하여야 한다. 다중 에이전트 시스템을 구성하는 에이전트들간의 통신이 순차적이며 동기적으로만 이루어진다면 통신의 효율이 나빠진다. 비동기식 통신은 에이전트간의 통신 효율에 있어서 중요하다[17].

3.3.2 통신 명령들의 처리 방식

(그림 3)은 INAS의 에이전트 통신 모듈이 통신 명령들을 처리하기 위한 구조를 보여준다. 다중 에이전트 환경에서는 여러 에이전트들이 동시에 메시지들을 보낼 수 있으므로 그런 경우에 대처하기 위하여 메시지 처리기 함수는 새로운 메시지를 받을 때마다 새로운 쓰레드(thread)의 형태로 수행되도록 설계되었다. 즉, 다른 에이전트들이 보낸 질의 메시지나 일방향 메시지 각각에 대하여 메시지 처리기 함수가 별개의 쓰레드로 수행된다.



(그림 3) 통신 명령들의 처리 구조

다른 에이전트에게 질의 메시지를 보내고 응답 메시지를 수신하는 과정에는 태그/메시지 테이블(tag/message table)과 수신 메시지 관리자(received message manager)가 사용된다. 태그/메시지 테이블은 태그 필드와 수신 메시지를 저장하기 위한 메시지 필드로 구성되는 태그/메시지 레코드들을 저장한다. 태그 필드가 레코드들의 검색을 위한 키(key) 필드의 역할을 한다. 수신 메시지 관리자는 태그/메시지 테이블을 이용하여 수신된 메시지들을 waitingMessage 명령이나 메시지 처리기 함수로 분배하는 역할을 수행한다.

NULL 심벌이 아닌 값을 <reply tag> 인자로 사용하는 sendMessage 명령은 메시지를 송신하기 전에 내부적으로 새로운 태그/메시지 레코드를 태그/메시지 테이블에 삽입한다. 그 레코드의 태그 필드와 메시지 필드에는 각각 <reply tag>와 빈 문자열(null string)이 기록된다. (그림 4)는 sendMessage 명령의 알고리즘이다.

```
Algorithm sendMessage
Input: receiver agent r, performative p, content c, tag t
Output: SUCCESS or ERROR
begin
    make a message string using r, p, c, and t;
    if (t != NULL)
      begin
        if (there already exists a record whose tag field contains t in the
            tag/message table)
              return ERROR; /* already used tag */
        insert the record <t, null> into the tag/message table;
      end
    send the message string to the agent r;
    return SUCCESS;
end
```

(그림 4) sendMessage 명령의 알고리즘

질의 메시지를 보낸 에이전트의 수신 메시지 관리자는 응답 메시지가 도착하면 태그/메시지 테이블을 참조하여 응답 메시지의 태그와 같은 태그 필드 값을 갖는 태그/메시지 레코드의 메시지 필드에 응답 메시지를 기록하고 대기 상태의 waitingMessage 명령을 깨우기 위한 이벤트를 발생시킨다. waitingMessage 명령은 수행이 시작되면 태그/메시지 테이블을 참조하여 자신의 인자로 사용된 <reply tag>와 같은 태그 필드 값을 갖는 태그/메시지 레코드의 메시지 필드의 값을 읽는다. 만일 그 값이 빈 문자열이 아니라면 응답 메시지가 도착한 것이므로 태그/메시지 테이블의 해당 레코드를 삭제한 후 응답 메시지의 문법 검사를 수행하고 문법에 오류가 없으면 응답 메시지를 반환한다. 만일 메시지 필드의 값이 빈 문자열이면 waitingMessage 명령은 대기 모드로 들어간다. 대기 모드로 들

어간 waitingMessage 명령은 인자로 주어진 <duration>에 해당하는 시간만큼 시간이 지나거나 수신 메시지 관리자에 의해 응답 메시지가 수신되었다는 이벤트를 받으면 활성화된다. 대기 모드의 waitingMessage 명령을 깨우기 위해서 내부적으로 이벤트 메커니즘이 이용된다. (그림 5)는 waitingMessage 명령의 알고리즘이다.

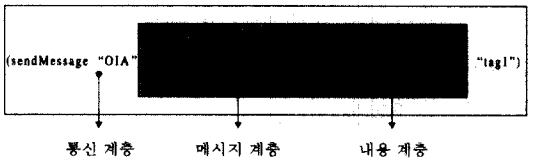
```
Algorithm waitingMessage
Input: tag t, duration d
Output: message string or ERROR
begin
    c := get the current time;
    r := get a record whose tag field contains t from the tag/message table;
    if (r does not exist) return ERROR; /* mismatched tag */
    if (r.message == null) /* the reply message does not arrive yet */
      begin
        wait for the event on the arrival of the message with tag t till
        time c + d;
        m := get the value of the message field of r;
        delete r from the tag/message table;
      end
    return m;
end
```

(그림 5) waitingMessage 명령의 알고리즘

다른 에이전트가 <reply tag>나 <receiver> 인자를 잘못 지정하여 sendMessage 명령을 사용하는 경우가 존재할 수 있다. 그런 경우에 대비하여, 메시지 처리기 함수는 태그 값이 NULL인 메시지 뿐 아니라 태그/메시지 테이블에 없는 태그 값을 갖는 메시지도 수신하도록 설계되었다. 잘못 전달된 메시지는 메시지 처리기 함수에서 메시지 반송 등의 방법으로 처리될 수 있다. 태그/메시지 테이블의 레코드의 수는 질의 메시지를 보냈으나 아직 응답 메시지가 수신되지 않은 경우들의 수와 같게 유지된다.

3.3.3 통신 명령의 구조

INAS가 제공하는 통신 명령들은 표준 에이전트 통신 언어인 KQML[13]에서와 마찬가지로 통신 계층(communication layer), 메시지 계층(message layer), 내용 계층(content layer)으로 구성된 계층화된 구조를 가진다. 통신 계층은 <receiver>나 <reply tag> 값과 같은 통신 파라미터들을 기술하는 층이다. 메시지 계층은 TELL, ASKONE, REPLY 등의 <performative> 인자의 값을 기술하는 층이다. 내용 계층은 실제로 전달하고자 하는 내용이 기술되는 계층이다. 통신 명령의 구조를 계층화하는 것은 구현의 편리 및 의미의 명확성을 확보하는 데 필요하다. (그림 6)은 OIA라는 에이전트에게 id가 6636인 사람의 이름이 무엇인가를 질의하는 예제 통신 명령을 이용하여 통신 명령의 계층 구조를 보여준다. 에이전트 OIA는 이 메시지에 대한 응답으로 <performative>가 "REPLY"이고 <reply tag>가 "tag1"인 응답 메시지를 보낼 것이다.



(그림 6) 계층화된 구조의 통신 명령

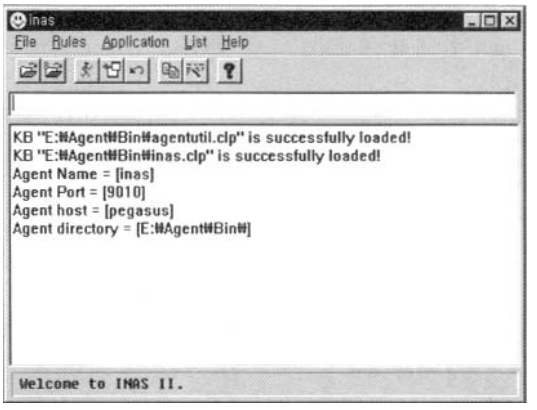
3.3.4 기타 사항

INAS 버전 2는 KIF(Knowledge Interchange Format)[18]와 같은 특정한 내용 계층 언어를 사용하지 않으며, 중첩이 가능한 일반적인 리스트(general list)라는 형태적 조건만을 요구한다. 하지만, 통신 명령들은 계층화된 구조를 가지므로 KIF와 같은 특정한 내용 계층 언어를 추후에 쉽게 도입할 수 있다.

sendMessage 명령과 waitingMessage 명령은 효율적이며 다양한 방식의 통신을 지원하지만 그대로 사용하기에는 번거로운 경우가 있다. INAS의 에이전트 구축 언어는 기존의 함수들을 조합하여 새로운 사용자 함수를 정의할 수 있는 기능을 제공하므로 sendMessage 명령과 waitingMessage 명령을 조합하여 보다 고수준의 통신 명령들을 정의할 수 있다. 예를 들어 DF나 AMS 같은 관리 에이전트들과 시스템 관리 차원의 관례적인 메시지들을 주고받는 경우, 그 절차를 하나의 고수준의 통신 명령으로 정의한다면 편리할 것이다. 또한 일방향 통신이나 동기식 통신의 전체 과정을 각각 하나의 고수준의 명령으로 정의할 수 있다. INAS의 기본 에이전트 프로그램인 agentutil.clp에는 일방향 통신을 위한 (tell <receiver> <content>) 명령이나 동기식으로 질의 메시지를 보내고 응답 메시지를 반환하는 (syncAsk <receiver> <content> <duration>) 명령 등이 사용자 정의 함수로 정의되어 있다.

INAS는 MS Windows 환경에서 개발되었으며, 에이전트 통신 모듈은 통신 프로토콜로 TCP/IP를 사용한다. INAS 버전 2를 수행하면 (그림 7)과 같은 창(window)이 나타난다. 그 창은 에이전트 구축 언어로 작성된 에이전트 프로그램들을 적재하여 수행하거나

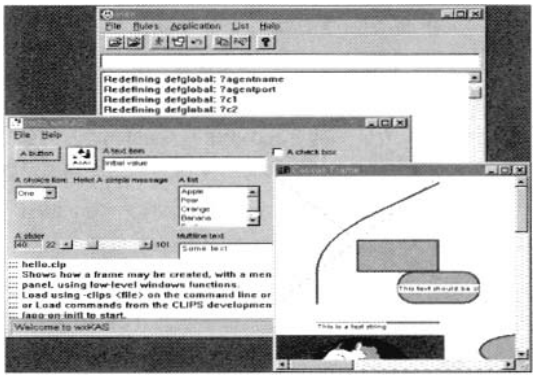
오류를 검사하기 위한 에이전트 개발자 인터페이스 기능을 제공한다.



(그림 7) INAS 버전 2의 초기 화면

3.4 그래픽 사용자 인터페이스 구축 기능

그래픽 사용자 인터페이스를 필요로 하는 에이전트 시스템이 존재한다. INAS 버전 2는 그래픽 방식의 사용자 인터페이스의 구축을 지원하기 위하여 창, 버튼, 체크박스(check box), 슬라이더(slider) 등의 그래픽 사용자 인터페이스 구성 요소들을 생성하는 명령들을 제공한다. INAS 버전 2의 그래픽 사용자 인터페이스 구축 지원 기능들은 wxWindows 시스템[20]을 이용하여 구현되었다. wxWindows 시스템은 X-Window, MS Windows 등의 다양한 윈도우 시스템들에 이식될 수 있는 그래픽 사용자 인터페이스 생성 도구이다. (그림 8)은 INAS 버전 2가 제공하는 그래픽 사용자 인터페이스 구축 명령들로 생성한 예제 창들이다.



(그림 8) INAS 버전 2의 그래픽 사용자 인터페이스 구축 기능의 예

3.5 에이전트 구축 언어의 통합성 및 확장성

INAS 버전 2는 지능형 에이전트들의 구축에 필요한 모든 기능들을 통합된 에이전트 구축 언어의 형태로 지원한다. 또한 INAS 버전 2가 제공하는 지능형 에이전트의 구축에 필요한 여러 기능들은 서로 밀접히 결합될 수 있다. 예를 들어, 생성 규칙의 행위 부분(action part)에서 에이전트간의 통신 명령들을 호출하여 지식을 다른 에이전트에게 전달할 수 있다. 사용자 정의 함수나 메시지 처리기 함수에서 수신된 질의 메시지에 응답하기 위해 에이전트 프로그램 베이스로부터 필요한 에이전트 프로그램들을 동적으로 적재하거나 추론을 시작시킬 수도 있다.

INAS 버전 2의 에이전트 구축 언어는 확장성을 갖는다. 에이전트 구축 언어가 제공하는 기존의 내장 함수들을 조합하여 새로운 사용자 정의 함수를 정의할 수 있으며, 기존의 함수들의 조합으로 불충분한 경우에는 C/C++ 언어로 정의된 함수를 에이전트 구축 언어의 새로운 내장 함수로 등록할 수 있다. 새로운 내장 함수를 등록하는 경우에는 컴파일 및 링크 작업이 필요하다. INAS 버전 2가 지원하는 에이전트 구축 언어는 문자열 처리 함수들 및 파일 입출력 함수들을 비롯한 다양한 함수들을 지원하므로 새로운 내장 함수가 필요한 경우는 그리 많지 않다.

4. INAS의 활용

실용적 분야에서의 INAS 버전 2의 활용 사례는 두 가지가 있다. 그것들은 에이전트 기반 워크플로우(workflow) 시스템에서의 자동 결재선 지정을 위한 MASAS (an intelligent Multi-Agent System for Approval Scheduling) 시스템[21]과 인터넷 정보 제어 및 분류를 위한 다중 에이전트 시스템[22]에서의 지능형 에이전트인 MA(Meta Agent)이다.

MASAS를 구성하는 에이전트의 하나인 ASA(Approval Scheduling Agent)는 결재 단계에 대한 스케줄을 작성하는 지능형 에이전트이다. 에이전트 ASA는 사용자 인터페이스 에이전트 등의 다른 에이전트들로부터 결재 스케줄의 작성을 요청하는 메시지를 받으면, 결재 조건들(예를 들어, 최종 결재가 5일 이내에 이루어져야 한다는 조건)을 확인하고 결재에 관련된 자신

의 지식베이스를 사용하여 결재 단계를 결정한다. 그 결정 과정에는 조직 정보 에이전트, 개인 일정 정보 에이전트와 교신하여 결재에 필요한 부수적 정보를 수집하는 작업이 포함된다. MASAS 시스템은 두가지 버전이 개발되었다. 초기 버전의 MASAS[3]는 버전 1의 INAS로 개발되었으며, INAS 버전 2가 개발이 마무리되기 직전에 INAS 버전 2의 검사를 겸하여 MASAS 버전 2가 개발되었다[21]. MASAS 버전 1과 2는 기능면에서는 유사하지만 버전 1은 C/C++ 언어와 지식 표현 언어를 결합하여 개발되었으며 버전 2는 통합 에이전트 구축 언어로 개발되었다는 점에서 다르다.

MA(Meta Agent)는 인터넷상의 HTML 문서들의 분야(예를 들어, 스포츠 분야, 의학 분야)와 위험성(예를 들어, 폭력성, 음란성) 수준을 자동적으로 분류하기 위한 지능형 에이전트이다. MA는 데이터베이스 에이전트, 사용자 인터페이스 에이전트 등의 다른 에이전트들과 교신하여 협동하도록 설계되었다. MA는 정보 검색 분야의 고전적인 알고리즘들을 표현하는 함수들과 규칙으로 표현된 경험적 지식들을 사용한다. 정보 검색에 관련된 함수들의 예로는 불용어(stoplist) 처리 함수, 스테밍(stemming) 함수들이 있다. 그런 함수들은 INAS 버전 2에 새로운 내장 함수 또는 사용자 정의 함수들을 정의함으로써 구현할 수 있었다. MA는 현재 시제품의 형태로 구현되었지만, 실용적인 문서 자동 분류에 대한 가능성을 보여주고 있다. MA는 INAS 버전 2가 정보 검색 분야와 같은 복잡한 응용 분야에도 효과적으로 사용될 수 있음을 보여준다.

본 논문의 저자들은 INAS 버전 2를 사용한 몇 가지 지능형 에이전트들의 개발을 통하여 통합 에이전트 구축 언어 방식의 에이전트 개발 인터페이스가 효과적임을 경험할 수 있었다. 그 근거는 다음과 같다. 첫째, 통합 에이전트 구축 언어를 이용하면 대부분 C/C++ 언어의 코딩 작업이 거의 필요하지 않았다. 둘째, 따라서 C/C++ 언어와 지식 표현 언어의 결합에 관련된 문제점들을 고려할 필요가 없었다. 셋째, INAS 버전 2의 에이전트 구축 언어는 지식 표현 언어 수준의 고수준 언어이므로 같은 내용을 C/C++를 사용하여 표현할 때보다 작성해야 하는 코드의 크기가 감소한다.

5. 결 론

지능형 에이전트 시스템들의 개발을 위하여 고수준의 지식 표현 언어 뿐 아니라 범용 프로그래밍 언어를 함께 사용해야 하는 기존의 에이전트 프레임웍들은 지식 표현 언어와 범용 프로그래밍 언어간의 수준 및 자료 표현 모델의 차이에서 유래되는 문제점들을 갖고 있다. 본 논문은 그런 문제점들을 극복하기 위한 대안으로 개발된 통합 에이전트 구축 언어를 지원하는 새로운 유형의 지능형 에이전트 쉘의 설계와 구현을 논하였다.

INAS 버전 2는 다양한 분야의 지능형 에이전트 시스템들을 효과적으로 구축할 수 있도록 설계되고 구현되었다. 그 근거는 INAS 버전 2의 에이전트 개발 인터페이스가 통합 에이전트 구축 언어를 제공하는 방식이라는 점에 있다. 첫째, 통합 에이전트 개발 언어는 고수준 지식 표현 언어와 범용 언어의 결합에 관련된 문제점들을 극복한다. 둘째, 통합 에이전트 개발 언어는 규칙 기반 전문가 시스템 쉘(rule-based expert system shell)의 기능을 포함한다. 따라서 전문가 시스템과 같은 전통적인 지식 기반 시스템들의 응용 분야에 활용될 수 있다. 셋째, 통합 에이전트 개발 언어는 다양하고 효율적인 에이전트 통신 명령들을 제공한다. 따라서 분산 환경에서 동작하는 다중 에이전트 시스템들의 구축에 활용될 수 있다. 넷째, INAS 버전 2의 통합 에이전트 구축 언어는 지능형 다중 에이전트 시스템의 구축에 필요한 다양한 기능들을 밀접히 통합된 방식으로 제공하며, 반복, 조건 분기, 입출력 등의 다양한 제어 구조 및 규칙, 절차, 객체와 같은 다양한 프로그래밍 패러다임들을 제공하므로 개발자에게 응용 분야의 필요에 따라 적합한 기능들과 패러다임을 선택하여 사용할 수 있도록 해준다. 다섯째, 통합 에이전트 구축 언어는 확장성을 제공하므로 새로운 응용 분야에서 필요로 하는 새로운 기능을 쉽게 추가할 수 있다.

INAS 버전 2의 문제점들로는 INAS가 참조하는 FIPA 에이전트 참조 모델에 관련된 기능들 중에 특히 에이전트 생명 주기에 관련된 기능들의 구현이 부족하다는 것과 다중 에이전트 환경에서의 에이전트간의 효과적 협력을 위한 작업 분배나 계획 수립 메커니즘이 미흡하다는 점을 들 수 있다. 향후 연구로는 그런 문제점들의 해결 및 전자 상거래를 위한 트랜잭션 처리 기능과 같은 특색있는 기능을 갖는 에이전트 쉘로 발전시키기 위한 작업등이 필요하다. 지능형 에이전트의 이론적 모델에 대한 탐색도 필요하다. 효과적으로 구현될 수 있는 지능형 에이전트 구축 언어 모델에 대한

연구가 부족하다고 사료된다.

참 고 문 헌

[1] Michael Wooldridge, Nicholas R. Jennings, "Agent Theories, Architectures, and Languages : A Survey," Lecture Notes in Artificial Intelligence #890, pp.1-39, Springer-Verlag, 1995.

[2] Thomas Connolly, Carolyn Begg and Anne Strachan, 'Database Systems A Practical Approach to Design, Implementation and Management,' 2nd Ed., Addison-Wesley, 1999.

[3] 장혜진, 에이전트 시스템을 위한 지식 기반 추론 모듈의 개발에 관한 연구 보고서, 한국전자통신연구소 위탁과제, 1997년 11월.

[4] Michael N. Huhns, Muninder P. Singh, "Agents and Multiagent Systems : Themes, Approaches, and Challenges," Readings in Agents, Morgan Kaufmann, pp.1-23, 1998.

[5] IBM, "IBM Agent Building Environments (ABE) Documentation," http://badaam.csc.ncsu.edu/research/Abe/index.htm.

[6] 백순철, 최중민, 장명욱, 박상규, 임영환, "이형 분산 환경에서의 에이전트들간의 이형성을 극복하기 위한 멀티 에이전트 기반 구조", 정보과학회논문지(c), 제2권 제1호, 1996년 3월.

[7] Philip R. Cohen, Adam Cheyer, Michelle Wang, and Soon Cheol Baeg, "An Open Agent Architecture," Readings in Agents, Morgan Kaufmann, pp.197-204, 1998.

[8] Yilmaz C., Soheil K., and Darrell L., "DYNACLIPS : A Dynamic Knowledge Exchange Tool For Intelligent Agents," in Proc. of the 3rd CLIPS Conference, Houston, TX, 1994.

[9] YilSoft, "AGENT_CLIPS 1.0," http://users.aimnet.com/~yilsoft/softwares/agentclips/ agentclips.html.

[10] Ernest Friedman-Hill, "KAPICLIPS 1.0," http://www.cs.umbc.edu/kqml/software/ kapiclips.shtml.

[11] Software Technology Branch, 'CLIPS Reference Manual Version 6.0' Vol. I, II, III, NASA Johnson Space Center.

[12] Ernest J. Friedman-Hill, "Jess, The Java Expert System Shell," http://herzberg1.ca.sandia.gov/jess/, 1998.

[13] Tim Finin, R. Frizson, D. McKay and R. McEntire, "KQML as an agent communication language," in Proc. of CIKM 94, pp.126-130, 1994.

[14] FIPA, "FIPA 97 Specification Version 2.0," http://fipa.comtec.co.jp/fipa/spec/ FIPA97.html, 1997.

[15] Stuart Russel and Peter Norvig, 'Artificial Intelligence A Modern Approach,' Prentice Hall, 1995.

[16] Charles Forgy, "Rete : A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," Artificial Intelligence, 19, pp.17-37, 1982.

[17] Mario Tokoro, "The Society of Objects," Readings in Agents, pp.421-429, Morgan Kaufman Pub. Co.

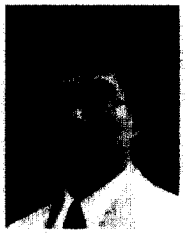
[18] M. Genesereth and R. Fikes, 'Knowledge Interchange Format Version 3.0 Reference Manual,' Technical Report Logic-92-1, Computer Science Department, Stanford Univ.

[19] 박상규, 민병의, 황승구, 박장석, "에이전트 기반 워크플로우 시스템", 정보처리학회지, 제4권, 제5호, 1997년 9월.

[20] Julian Smart, "Welcome to wxWindows,", http://dsn.astro.univie.ac.at/~sperl/wxwin/wx_contents.html, 1999.

[21] In Chol Kim, Hae Jin Chang, "An Intelligent Multi-Agent System Based on the FIPA Agent Reference Model," in Proc. of Pacific Rim International Workshop on Multi-Agents, pp.53-59, Nov. 1998.

[22] 하이콤리서치, 내용 기반 웹 정보 분석 에이전트 개발 보고서, 정보통신부 산업기술개발과제, 1999년 7월.



장 혜 진

e-mail : hjchang@smuc.sangmyung.ac.kr

1985년 서울대학교 사범대학 수학교육과(학사)

1987년 서울대학교 계산통계학과 전산과학전공(석사)

1994년 서울대학교 계산통계학과 전산과학전공(박사)

1987년~1989년 한국전자통신연구소 연구원

1994년 이후 상명대학교 컴퓨터정보통신학부 교수

관심분야 : 인공지능, 에이전트 시스템, 분산 소프트웨어, 객체지향 데이타베이스