JATLite 기반 멀티 에이전트 시스템의 설계 및 구현

이 해 수† · 장 덕 성††

요 약

기존의 웹 기반 의료 서비스는 환자 자신이 직접 해당 사이트를 찾아 읽어보고 의사들과 e-mail로 상담하는 수준이었다. 환자는 자신의 증상과 유사한 정보가 어디에 있는지 알아야 하고 의사들은 e-mail을 일일이 읽고 답해 주어야만 한다. 그러나 메일 에이전트, 진단 에이전트 등이 존재하는 멀티 에이전트 시스템이 개발된다면 기존의 web 기반 의료 서비스의 단점을 보완할 수 있다.

본 논문에서는 JATLite 기반의 멀티 에이전트 시스템을 설계하여 자동으로 e-mail을 분석, 분류, 전달, 진단하는 시스템을 개발하였다. 제안된 멀티 에이전트는 메일 에이전트, 진단 에이전트, 중재 에이전트 등으로 구성된다. 메일 에이전트는 환자의 상담, 요청, 증상 등을 분석하여 해당하는 진단 에이전트에게 전달하는 일을 한다. 진단 에이전트는 지식베이스를 이용하여 환자의 요구사항에 가장 적절한 진단을 하게 된다. 중재 에이전트는 에이전트의 상태를 파악하여 작업을 지시하고 메일 에이전트와 진단 에이전트간의 통신을 관리한다.

Design and Implementation of the Multi-Agent System based on JATLite

Hae-Su Lee† · Duk-Sung Jang††

ABSTRACT

The existing web-based medical services were limited to searching the appropriate web sites and sending e-mails to doctors by patients. The patients had to know where the information related to their symptoms is and the doctors had to read and answer all of the e-mails for consulting.

Through this research, I want to develop the multi-agent system to analysis, classify, and deliver the e-mail automatically for the doctors and patients. In this paper, a multi-agent system is designed upon JATLite, which is composed a Mail-Agent, a Diagnose-Agent and a Coordinate-Agent. The Mail-Agent analyzes the symptom of a patient and delivers it to a proper Diagnose-Agent. Then the Diagnose-Agent examines from the knowledge base. The Coordinate-Agent administrates communication between the Mail-Agent and the Diagnose-Agents, and finds the proper Diagnose-Agent to examine disease.

1. 서 론

인터넷의 사용자가 급증하고 개인마다 각자의 홈페이지를 가지는 시대가 도래했다. 개인 병원이나 종합병원 등에서도 각 병원의 홈페이지를 만들어 환자의 진료를 돕거나 궁금증을 해결하도록 정보를 제공하고 있다[14]. 그러나, 홈페이지를 통해 제공되는 서비스는 정적인 정보만을 제공하는 데 그치게 되고, 의사들이 게시판의 글을 일일이 읽어보아야 환자에게 도움을 줄 수 있다. 환자 또한 자신의 질병에 대한 정보를 얻기 위해 병원 홈페이지를 이곳 저곳 찾아 다녀야 한다. 이런 불편을 해결하기 위해서는 환자와 의사의 개입 없이 번거로운 작업들을 대행해 주는 진단 시스템이

† 준 회 원 : 계명대학교 대학원 컴퓨터공학과
†† 정 회 원 : 계명대학교 컴퓨터공학과 교수
논문접수 : 2000년 6월 24일, 심사완료 : 2000년 7월 27일

필요하다.

환자의 증상을 진단하기 위해서는 여러 의료분야의 전문지식을 주고받는 과정이 필요하다. 멀티 에이전트 시스템은 분산된 네트워크에의 통신을 지원하고, 에이전트의 추가와 삭제가 용이하므로 이러한 병원 진단 시스템의 요구에 수렴할 수 있을 것이다.

멀티 에이전트 기반 진단 시스템은 새로운 지식베이스가 추가될 때에도 자유롭게 확장되어 사용될 수 있다. 새로운 에이전트와 수행문이 추가될 때, 다시 컴파일 하지 않고 중재 에이전트에 등록되기만 하면 되기 때문이다. 또한 지식 베이스나 수행문만을 변경시키기가 용이하므로 급변하는 정보에 대한 대처 능력이 뛰어나다.

멀티 에이전트로 진단 시스템을 구축할 때 환자들은 일일이 웹을 서핑(Surfing)하지 않고도 진단을 받을 수 있으며, 의사들은 메일을 하나하나 읽어 보지 않아도 된다. 환자가 임의의 진단 에이전트 시스템으로 메일을 보내면, 에이전트들은 통신을 통해 진단을 완료하고 결과를 메일로 전달하게 된다. 중재 에이전트는 자신의 시스템에서 해결할 수 없는 진단일 경우 네트워크를 통해 다른 시스템으로 메시지를 전달하게 되고, 진단 후 결과를 환자에게 메일로 통보하게 된다.

병원 진단 시스템은 접수를 대신하는 메일 에이전트가 환자의 증상을 분석하여 중재 에이전트로 보내고, 중재 에이전트는 분석된 결과에 따라 응용 에이전트(내과 진단 에이전트, 치과 진단 에이전트 등)로 진단을 의뢰하게 된다. 전문의 역할을 하는 각각의 진단 에이전트는 지식베이스를 통해 환자의 증상을 진단하게 된다. 하나의 에이전트 시스템에서 진단할 수 없는 증상의 경우는 다른 시스템의 진단 에이전트와 통신하면서 정보를 교환하며 진단하게 된다. 따라서 환자는 임의의 진단 에이전트와 접속하더라도 원하는 진료를 받을 수 있게 된다.

본 연구에서는 병원 진단 시스템을 멀티 에이전트를 이용하여 설계함으로써 병원 업무에의 에이전트 개념의 도입 가능성을 확인하고 멀티 에이전트 응용에 대한 기초 기술을 확립하고자 한다.

병원 진단 시스템은 KQML(Knowledge Query and Manipulation Language) 기반의 에이전트 플랫폼인 JATLite(Java Agent Template, Lite)와 ASP(Active Server Page) 그리고 Active X Server Component를 이용하여 멀티 에이전트 시스템으로 설계 및 구현하였고, 분산된 에이전트들 사이의 통신을 위해서 각 에이전트는 JATLite 기반의 라우터를 통해 KQML 메시지를 주고받게 된다[11, 13]. 메일의 전달은 SMTP(Simple Mail Transfer Protocol)와 POP3(Post Office Protocol Ver.3) 서비스를 이용하며, 메일 본문을 파싱(parsing)하고 분류하는 작업은 ASP와 Active X Server Component를 통해 이루어지게 된다.

2장에서는 관련 연구로 멀티 에이전트의 개요와 KQML, JATLite를 간단히 소개하고, 3장에서는 진단 시스템을 설계 및 구현하였다. 4장에서는 실행결과를 보이고, 5장에서 결론을 맺는다.

2. 관련 연구

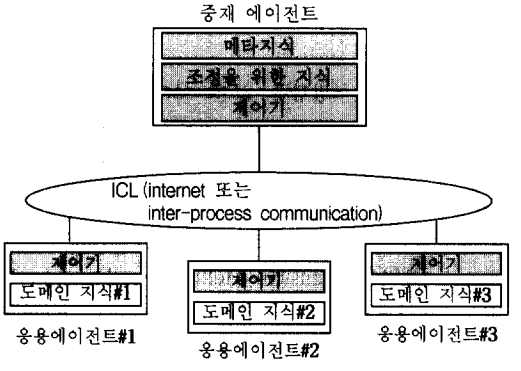
에이전트의 연구방향은 크게 에이전트 이론(agent theory), 에이전트 구조(agent architecture), 에이전트 언어(agent language), 에이전트 응용(agent application) 등으로 구분된다. 이들 연구를 집약하는 대표적인 에이전트 기반 기술로는 OAA(Open Agent Architecture), EMAF(Extensible Multi-Agent Framework), FIPA(Foundation for Intelligent Physical Agent)의 AP(Agent Platform), JATLite(Java Agent Template, Lite), ACACIA(Agency Collaboration Architecture for Cooperating Intelligent Agent), Challenger, MAP (Multi-Agent Planning), PGP(Partial Global Planning), PAST(Postech Agent Software Technology) 등을 들 수 있다[1, 6, 12, 13]. 이들 중 JATLite는 공개된 Java 패키지로서 KQML(Knowledge Query and Manipulation Language)을 기반으로 한다[9, 13]. 본 논문에서는 JATLite를 사용하여 멀티 에이전트를 설계하고 구현하였다.

2.1 멀티 에이전트

하나의 에이전트로 해결하지 못하는 복잡한 문제의 해결을 위하여 다른 에이전트들과의 협동이 필요하게 되었고, 이를 효과적으로 수행하기 위해 멀티 에이전트 시스템이 제안되었다[4, 5]. 멀티 에이전트 시스템은 여러 응용 에이전트 외에 중재 에이전트(coordinating agent, facilitator)라는 조정자를 통해 메시지의 전달과 각 에이전트의 제어를 수행한다. 이 경우 모든 응용 에이전트의 통신 메시지는 중재 에이전트를 통해 상대 에이전트로 전달된다. 각 에이전트가 수행할 수 있는

처리 능력에 대한 정보는 응용 에이전트 생성 시에 중재 에이전트로 등록되며, 중재 에이전트는 이를 바탕으로 통신 메시지를 해당 응용 에이전트에게 전달한다. 응용 에이전트는 다른 에이전트가 어떤 일을 할 수 있는 지에 대하여 알 필요가 없고, 단지 필요시 통신 메시지를 중재 에이전트에게 보내어 맡기면 된다[2, 5].

멀티 에이전트는 일반적으로 연합 에이전트 구조(federated agent architecture)를 가진다[4, 5]. 연합 에이전트 구조는 하나의 중재 에이전트와 여러 에이전트가 계층구조를 이루고, 중재 에이전트는 다른 멀티 에이전트 시스템의 중재 에이전트와 메시지를 주고받는다.



(그림 1) 에이전트 시스템 구조

(그림 1)은 일반적인 에이전트 시스템의 구조를 보여준다. 중재 에이전트는 각 응용 에이전트의 요구를 수용하면서 응용 에이전트들 사이의 정보전달을 담당한다. 응용 에이전트의 '도메인 지식'은 각 응용 에이전트의 목적을 수행하기 위한 지식이며, 중재 에이전트의 '조정을 위한 지식'은 에이전트들을 관리하기 위한 지식이다. '메타 지식'은 새로운 에이전트의 등록이나 상이한 시스템과의 통신에 사용된다. '제어기'는 각 에이전트의 동작을 제어하며 목적을 수행하도록 한다. ICL(Internet or Inter-process Communication Language)은 에이전트 통신에 이용되는 언어로써 본 연구에서는 KQML을 이용한다.

에이전트 통신의 주된 목적이 다른 에이전트와의 협동을 통한 문제해결이라고 할 때 몇 가지 문제점이 발생한다. 즉 다른 에이전트가 존재하는 지, 또는 다른 에이전트가 해결할 수 있는 능력이 무엇인지를 알고 있어야 한다는 것이다. 그렇지 않고는 어떤 에이전트에게 도움을 청할 지 판단할 수 없기 때문이다. 또 다른 문제점은 모든 에이전트들끼리 서로 연결되어 있어야 하기 때문에 발생하는 네트워크 사용의 비용과 성능저하를 들 수 있다[15, 16].

중재 에이전트는 이러한 문제점들을 조정할 수 있는 능력을 가져야 한다. 응용 에이전트가 어떤 능력을 가지는 지에 대해서는 OTS(Ontology Type Server)를 통해서, 에이전트의 주소와 위치에 대해서는 ANS(Agent Name Server)를 통해 파악하게 된다. 또 네트워크 사용 비용을 절감하기 위해 필요한 정보만을 전달하여 각각의 응용 에이전트 자체 처리에 맡기고, 그 결과만을 돌려 받음으로 네트워크 사용 부하를 줄일 수 있게 된다[1].

효과적인 정보전달을 위해서는 서로 상이한 에이전트 시스템간의 정보 전달이 필요하다. 이를 위해서는 각각의 시스템들이 어떤 형태로 작성되어 있는지 알 필요가 있다. 각각의 시스템은 개개의 목적에 맞는 특정 환경으로 구성되어 있기 때문에 모든 시스템을 만족하는 에이전트 통신환경을 구축하는 것은 상당히 어렵다. 따라서 에이전트가 정보를 교류하기 위해서는 각 에이전트를 중개하는 기능이 있어야 한다. 각각의 에이전트는 서로 다른 표현법에 따라 정의되고 통신하는데, KQML(Knowledge Query and Manipulation Language)에서는 이 상이한 표현법을 Ontology라 명하고 있다.

Ontology란, 개념적인 의미에서 객체들을 정의한 것으로, 에이전트가 특정 도메인에 관한 질의를 하거나 통신하고자 할 때 먼저 요구되어 지는, 그 도메인에 대한 개념화를 말한다. 즉 Ontology는 도메인에 대한 공통적 어휘 기반을 제공하는 이 기종 시스템의 운용과 지적 시스템 개발에 필수적인 요소이다[19, 20].

2.2 KQML

KQML은 정보와 지식을 교환하기 위한 언어이자 프로토콜이다. 또한 에이전트들이 실시간으로 지식을 공유하는 것을 지원하는 메시지 포맷과 메시지 관리 프로토콜이다[22, 23]. 이것은 ARPA KSE(Advanced Research Projects Agency Knowledge Sharing Effort) 연구의 일환으로서, 공유가능하고 재사용 가능한 대규모 지식 베이스를 위한 기술과 방법들을 개발하는데 중점을 두고 있다. 인텔리전트 시스템과 상호작용하기 위한 응용 프로그램을 위한 언어로서 사용될 수도 있고, 또는 상호 협력적인 문제 해결을 지원하기 위해

지식을 공유하는 두 개 이상의 인텔리전트 시스템들 사이에서 사용될 수도 있다[17].

KQML은 에이전트들 사이에서 각 상대방 에이전트의 지식과 목적을 얻기 위해 사용할 수 있는 허용 가능한 오퍼레이션들을 정의하는 수행문(perfomative)의 확장된 집합에 초점을 두고 있다[7, 9]. KQML은 에이전트들의 상호작용을 조정하는 특별한 에이전트인 통신 중재자(중재 에이전트)를 통해 지식을 공유하기 위한 기본적인 구조를 제공한다. KQML은 에이전트 사회에서의 대표적인 에이전트 통신 언어로 ARPA KSE의 한 연구 그룹인 EIWG(External Interfaces Working Group)에 의해 제시되었다[8, 10]. 이들 연구의 궁극적인 목적은 단독 시스템으로 제공하는 별개의 기능성보다 여러 시스템들이 같이 참여하여 보다 풍부하고 다양한 기능성을 제공할 수 있게 하는 기반구조(infrastructure)를 정의하고 개발하는 것이다[8-10].

KQML은 다음과 같은 특성을 가진 에이전트에 적합한 통신언어이다[7, 22].

- 정보 교환과 관련한 통신에 중점을 둔 통신언어이다.
- 기존의 프로그램에 쉽게 포장(wrapping)될 수 있다.
- 정보의 내용과 형식에 독립적이다.
- 임의의 지식 베이스들에 대한 응용 시스템을 설계할 때 지식 베이스의 위치와 이를 필요로 하는 곳의 위치가 제한적이지 않다.
- 지식 표현 방법 및 요구 방법을 자유롭게 정의해서 사용할 수 있으며 지식 전달 메커니즘에 있어서 많은 융통성을 제공한다.

실제 두 에이전트들 사이에 오고가는 KQML 스트림은 내용 표현(content expression)이 메시지 포장기(wrapper)에 의해 메시지 표현(message expression)으로 싸여서, 이것이 다시 통신 포장기에 의해 포장된 것으로 이해할 수 있다. 즉 기존의 통신 프로토콜의 계층 구조처럼, KQML도 내용 계층(content layer), 메시지 계층(message layer), 통신 계층(communication)의 3단계 계층으로 이루어진다[22].

KQML을 사용할 때 소프트웨어 에이전트는 내용(content) 메시지를 자신의 언어로 구성한 후, 이를 KQML 메시지 안에 싸서 보내게 된다. 내용 메시지는 어떤 표현 언어(representation language)로 쓰여지든지, ASCII 문자열이든지 혹은 이진 표현법(binary notation)들 중의 하나이든지 상관이 없다. KQML은 내용이 언제 시작되고 언제 끝나는가 외에는 메시지의 내용 부분에 관심을 가지지 않는다.

메시지 층은 내용 계층에 부가적인 정보를 포함시킨 것이다. 메시지는 그 성격에 의해 크게 내용 메시지와 선언 메시지로 나뉜다. 내용 메시지는 질의(query)와 같은 직접적인 지식 전달에 관련된 것이고, 선언(declaration) 메시지는 직접적인 지식전달은 아니나 이러한 활동에 관련된 메타 정보에 대한 것이다[7]. 특히 이 선언 메시지는 에이전트 사회에서 자신의 존재를 그 기능성과 함께 선언하거나 자신이 받고자 하는 어떠한 서비스에 대해 등록하거나 하는 등의 중요한 부분에 이용된다.

통신층은 KQML에서는 가장 바깥 계층으로 위에서 언급된 내용 계층, 메시지 계층을 거친 메시지를 통신에 관련된 부가 정보를 추가하여 패키지의 형태로 만든다. 여기에서 추가되는 정보는 sender, receiver 등이다.

KQML 메시지는 수행문(performative)이란 형태로 정의되어 있다. 이는 각각의 메시지들이 어떤 동작(action)을 내재하고 있음을 의미한다. KQML에 기반하여 구축되는 에이전트들은 반드시 정의된 모든 수행문을 지원해야 할 필요는 없다. 또한 본 사양(specification)에 정의되지 않은 수행문도 필요에 따라 정의하여 사용할 수 있다. 하지만 어떤 경우에서든 그 사용과 정의에 있어서 본 사양이 제시하는 형식에 맞도록 해야 한다. KQML 수행문은 기본적으로 ASCII 문자열로 표현된다. 구문의 대체적인 모습은 수행문 이름과 인수 리스트(parameter list)로 이루어진다. 이 중에서 인수 리스트는 인수 명과 인수 값의 쌍들로 이루어진 리스트이다. 여기서 주목할 만한 것은 수행문 인수들은 그 인수 명으로 구분될 뿐, 그 순서나 위치와는 무관하다는 점이다. 즉 같은 인수들의 내용만 담고 있으면 그 리스트 안에서의 순서는 상관이 없다.

KQML 의미론은 각 에이전트들의 기능성과 통신 상황에 대한 정형화된 문맥에 바탕을 두고 있다. 각각의 에이전트들은 외부에서 볼 때, 하나의 지식베이스(Knowledge Base, KB)의 관리자처럼 모델링 된다. 이것은 한 에이전트와의 통신은 곧 그에 딸린 지식베이스와의 통신임을 의미한다. 하지만 실제 구현상에 있어서 모든 에이전트가 자신의 지식베이스를 소유하는 것은 아니다. 실제 각 에이전트가 관리하고 있는 것은 경우에 따라 작은 데이터베이스일 수도 있고 단순히

프로그램 내에 있는 하나의 데이터 구조일 수도 있다. 이처럼 에이전트의 지식베이스에 해당하는 부분은 다양한 형태가 존재하므로 지식베이스보다는 가상지식베이스(Virtual Knowledge Base, VKB)라 부른다. 이러한 가상지식베이스의 내용은 크게 belief와 goal로 모델링된다. belief는 에이전트 자신 혹은 외부의 다른 에이전트가 지니고 있는 정보 그 자체이며, goal은 에이전트 자신이 외부의 환경에 대해 도달하고자 하는 상태를 말한다. KQML 수행문은 가상지식베이스의 이러한 belief와 goal과 관련하여 있을 수 있는 상황에 기반 한다[7, 21].

KQML 사양은 그 구현에 있어서 개발자에게 많은 융통성을 제공하고는 있지만 본 사양에서 지시하는 수행문에 한해서는 반드시 기준 사양을 따를 것을 요구하고 있다[7, 23]. 여기서 인수 전부를 필요로 하는 것은 아니며, 각 수행문마다 필요로 하는 인수가 다를 수 있고, 심지어 같은 수행문에서도 상황에 따라 이용되는 인수가 다를 수 있다. <표 1>은 KQML 사양에 미리 정의된 인수들이다.

〈표 1〉 미리 정의되어 있는 인수들(parameters)

인수명	의 미
: sender	메시지의 송신자를 표시.
: receiver	메시지의 수신자를 표시.
: content	메시지의 내용 부분.
: reply-with	원본 메시지에 대한 응답으로 사용될 변수를 알림.
: in-reply-to	원본 메시지에 대한 응답을 보냄(reply-with에서 사용된 변수를 사용한다).
: envelope	메시지 전송 서비스에 의해 보여진 유용한 정보에 대한 표현.
: language	action의 해석에 사용되는 언어.
: ontology	내용 표현에 쓰이는 기호들의 의미를 나타냄.
: reply-by	송신 에이전트가 받기 원하는 제한 시간 혹은 날짜.
: protocol	송신 에이전트가 의존하는 프로토콜을 알림.
: conversation-id	통신 과정에 사용되는 식별자를 알림.

2.3 JATLite

JATLite(Java Agent Template, Lite)는 Standford 대학에서 제작한 에이전트 프레임워크(Agent Framework) 개발을 위한 자바 패키지(Java Package)이다. 융통성을 위해 JATLite는 4가지 층(Abstract layer, Base Layer, KQML layer, Router layer)을 제공한다. 사용자는 제공되는 모든 층을 사용할 수 있으며 사용자 응용 프로그램과 그 층에 대한 가정에 따라 적절한 층을 사용할 수 있다. JATLite는 TCP/IP(Transmission Control Protocol/Internet Protocol)를 기본으로 하지만 다른 프로토콜(protocol)들로 확장이 가능하며 다중 서버 소켓(ServerSocket)과 다중 메시지 수신 소켓(message receiver socket)을 지원한다. JATLite는 통신 프로토콜에 대한 제한을 따르는 몇몇 층을 갖고 있다. 이 접근법에 의해서 개발자는 자신의 시스템 작성을 시작할 수 있는 적절한 층을 선택할 수 있다.

Java를 이용하면 웹 상에서 곧바로 수행될 수 있는 '애플릿'이라고 불리는 프로그램을 작성할 수 있다. 애플릿 프로그램은 별도의 자바 인터프리터 없이도 Netscape와 같은 웹 브라우저를 통해 수행시키고 웹 상에서 그 결과를 곧바로 볼 수 있다. 이 때 구동된 애플릿 프로그램은 그 애플릿을 전송한 웹 서버와만 소켓을 열 수 있다. 즉 서로 다른 호스트의 애플릿 프로그램간에는 직접적인 소켓 연결이 불가능하다. 애플릿 에이전트를 포함한 시스템에의 적용을 고려할 때 이는 애플릿을 전송한 웹 서버 상에서 작동하며 애플릿 에이전트로부터의 메시지를 중계하는 응용 프로그램이 필요함을 의미한다.

JATLite에서는 애플릿 에이전트에의 적용을 고려함과 동시에 보다 효율적으로 통신을 제어할 목적으로 '라우터'라는 메시지 중계 템플리트를 제공한다. JATLite를 이용한 멀티 에이전트는 라우터 클라이언트 애플릿, 라우터, 응용에이전트 등으로 구성된다. 라우터 클라이언트 애플릿(Router Client Applet)은 사용자와의 인터페이스(interface)라 할 수 있다.

사용자는 이 애플릿을 사용해서 어떤 일에 대해, 그 일을 수행할 에이전트의 정보와 그 일에 대한 요청 등을 포함한 메시지를 라우터에게 전달한다. 메시지를 전달받은 라우터는 이 메시지를 분석해서 지정된 에이전트에게 이 메시지를 전달한다. 메시지를 전달받은 지정된 에이전트는 요청 받은 일을 수행한 후 그 결과를 라우터에게 보낸다. 결과를 받은 라우터는 이를 라우터 클라이언트 애플릿에 보낸다. 라우터 클라이언트 애플릿은 받은 결과를 화면에 출력한다.

JATLite는 직접적인 소켓 연결을 통한 메시지 전달 방식으로 정보 교환을 구현하고 있다. 이러한 메시지 전달 방식으로는 메시지 폴링(polling)과 메시지 큐잉(Queuing)이 있는데, 전자는 메시지 전달이 필요할 때마다 일시적으로 소켓을 열고 즉각적으로 메시지를 전

달하는 방식이며, 후자는 메시지를 파일 시스템-메시지 큐에 저장해 두고 전달할 에이전트와 소켓 연결이 생기면 이를 전송하는 방식이다. 네트워크 시스템의 불안정성을 고려할 때 장기적인 프로젝트를 수행하는 에이전트 시스템이나 대형의 에이전트 시스템에서는 정보의 유실을 방지하는 등 보다 안정적인 정보 교환을 위해 메시지 큐잉 방식이 절대적으로 필요하다고 볼 수 있으며 JATLite는 라우터를 통하여 이러한 정보 교환 방식을 지원한다.

또한 라우터는 주소록(Address Table)을 관리함으로써 에이전트의 이름만으로 에이전트들 사이의 정보 교환이 가능하게 한다. 주소록에는 에이전트 시스템의 네트워크 정보 즉 각 에이전트의 이름과 그 IP Address, 포트, 에이전트 설명 등이 담겨 있다.

3. 병원 진단 시스템의 설계 및 구현

인터넷 이용자가 급증함에 따라 민간 요법이나 의료 정보를 웹을 통해 제공하고 메일을 통해 환자와 상담하여 진료를 돕는 가상 병원들이 늘어가고 있다. 이는 진료 전산화의 시도라고 할 수 있겠다. 하지만 환자가 진료를 받기 위해서는 개설된 웹 페이지를 일일이 찾아 다녀야만 하고, 웹 상의 각종 정보를 환자가 모두 이해한 후 스스로 판단하여 진단해야 한다. 의사 또한 일일이 메일을 읽어보아야 하는 번거로움이 있다.

병원 진단 시스템에 멀티 에이전트 개념을 도입함으로써 이러한 문제점을 보완할 수 있다. 환자는 자신의 증상을 전자우편을 통해 진단 시스템으로 보내기만 하면 진단 시스템은 에이전트들과의 통신을 통해 진단한 후, 결과를 메일로 환자에게 돌려주게 된다. 따라서 환자는 일일이 웹 페이지를 검색할 필요 없이 진단 시스템에 의한 진단 결과를 얻을 수 있고, 검색에 드는 시간과 노력을 절감할 것을 기대할 수 있다. 본 연구에서는 진단에서의 자동화가 가능한지 알아보기 위해 간단한 진단 시스템을 설계하고 실험한다.

3.1 전체 구조

(그림 2)는 진단 시스템의 전체 구성을 보여준다. 진단이 이루어지는 과정은 다음과 같다.

① 환자(사용자)는 자신의 증상들을 메일로 작성하여 진단 시스템에 보낸다.

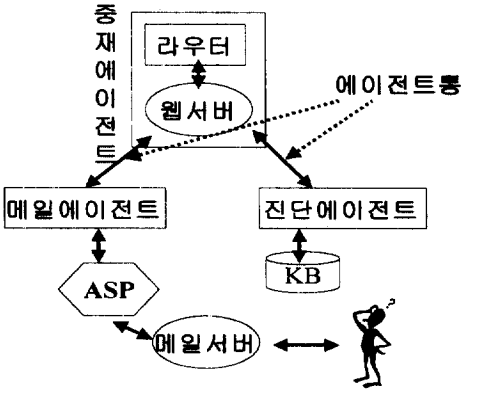
② 메일 에이전트는 메일을 읽어 키워드(Keyword)를 추출하여 간단한 분류과정을 거쳐 중재 에이전트에게 보낸다.

③ 중재 에이전트는 메일 에이전트로부터의 정보를 바탕으로 진단 에이전트에게 메시지를 중개한다.

④ 진단 에이전트는 전달된 키워드를 지식베이스를 통해 진단하고 진단 결과를 중재 에이전트에게 되돌려 준다.

⑤ 중재 에이전트는 진단 결과를 통보하기 위해 메일 에이전트에게 메시지를 전달한다.

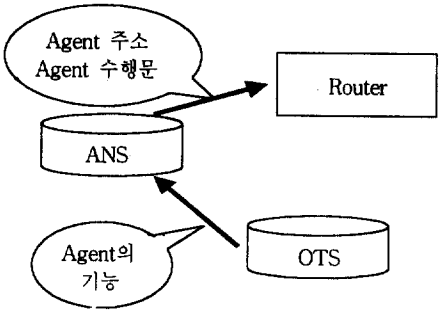
⑥ 메일 에이전트는 환자에게 메일로 진단 결과를 보냄으로 진단을 마치게 된다.



(그림 2) 전체 시스템 구성도

중재 에이전트는 에이전트의 정보를 중개하기 위해 ANS(Agent Name Server)와 OTS(Ontology Type Server)를 가진다. 이들은 상이한 시스템에서 서로 다른 형식언어를 정의했을 때에도 정보를 전달하게 하는 역할을 하며, 각 시스템에서 사용되어 지는 상이한 ontology 정보를 분석하여 자체 시스템에서 사용 가능한 형태로 바꾸어 주고, 필요한 에이전트의 위치와 수행문을 판단하는 일을 담당한다. 진단 에이전트는 환자의 증상들을 지식베이스를 통해 분석해서 진단하게 되고, 진단 결과와 민간 요법 또는 치료법 등의 정보를 반환하게 된다.

(그림 3)은 중재 에이전트의 구조를 나타낸다. 중재 에이전트는 OTS를 통해 어떤 에이전트를 수행해야 할지 판단하게 되며, ANS를 통해 응용 에이전트의 주소와 수행문을 알게 되어 등록된 에이전트에게 메시지를 전달하게 된다.



(그림 3) 중재 에이전트의 구조

본 연구에서는 진단을 위한 인수(parameter)들을 새롭게 정의하여 사용하고 있다. 진단 목적을 나타내는 'purpose', 진단된 질병을 나타내는 'disease', 진단된 질병에 대한 처방을 나타내는 'prescription', 진료과를 나타내는 'department', 에이전트의 동작 상태를 나타내는 'active', 에이전트의 수행 능력을 나타내는 'action' 등이다. <표 2>는 새롭게 정의한 인수들을 나타낸다.

〈표 2〉 새롭게 정의한 인수들(parameters)

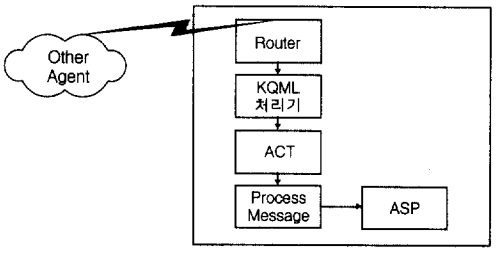
인수명	의 미
: purpose	메시지 전달 목적을 표시.
: department	진단하기 위한 목적 분과(내과, 외과, 안과 등)를 표시.
: disease	진단 에이전트의 수행 결과인 진단된 질병을 표시.
: prescription	특정 질병에 대한 처방을 표시.
: active	에이전트 상태를 표시.
: action	에이전트의 수행 능력을 표시.

중재 에이전트는 이 라우터의 정보를 관리하며 각 응용에이전트에게 메시지를 중개하고 할당 관리하게 된다. 본 연구에서 구축하는 에이전트 시스템은 TCP/IP 통신 규약을 따르고 에이전트들의 정보교환이 KQML을 통하여 이루어지므로 Router Layer의 클래스들을 이용하여 에이전트들을 구성할 수 있다.

본 연구에서 진단 에이전트 시스템은 정보전달을 위해 JATLite의 라우터 기능을 이용하였고, 라우터 클라이언트 애플릿 대신 ASP를 사용하여 사용자의 요구를 수렴하게 된다. JATLite의 라우터 기능은 메시지를 저장해 두었다가 에이전트가 동작할 때 전달해 주는 역할을 한다. 이를 통해 신뢰성 있는 통신을 책임진다. 라우터 클라이언트는 사용자와의 인터페이스 부분으로 본 실험에서는 전자우편을 통해 질의가 전달되고, 진단 결과도 전자우편을 통해 환자에게 전달되기 때문에 ASP에 의해 수행되도록 하였다.

3.2 에이전트 통신

JATLite는 에이전트 사이의 네트워크 기능, KQML 처리 기능, 기본적인 그래픽 사용자 환경 등을 제공하지만 각 에이전트가 수행해야 하는 고유한 기능들은 에이전트 개발자가 직접 추가하여야 한다. 여기서 KQML 처리 기능이라 함은 KQML의 문법적인 처리 기능만을 말하며, KQML 형식으로 전달되는 내용에 따른 합당한 처리기능은 사용자 메쏘드들을 각 에이전트의 기능에 맞추어 추가하여야 한다. JATLite의 RouterLayer에는 사용자가 재정의 하는 두 가지의 핵심적인 기능을 하도록 설계된 추상 메쏘드가 있다. 에이전트 개발자는 'Act'와 'ProcessMessage'라는 두 가지 메쏘드를 적절히 정의해 줌으로써 기본적인 에이전트의 골격을 갖출 수 있다. 에이전트 통신의 구조는 (그림 4)와 같다. Router는 에이전트들 사이의 통신을 책임지며, KQML 처리기는 Router를 통해 전달된 KQML 메시지가 KQML 문법에 맞는지 검사하고 오류가 없으면 Act가 동작한다. Act에는 KQML 문에 따라 처리되어야 할 작업들이 정의된다. Act에 의한 분석이 끝나게 되면 ProcessMessage를 통해 ASP를 구동시켜 에이전트의 작업이 시작되게 된다.



(그림 4) 에이전트 통신 구조

라우터와 KQML 처리기는 다음 (그림 5)와 같이 작성되어 진다. 라우터는 JATLite가 제공하는 통신 서비스를 통해 메시지를 전달받으며, KQML 처리기는 전달된 KQML 메시지의 문법 오류를 체크한다. Coord-Agent() 함수가 초기화되면서 ANS와 OTS를 메모리로 읽어 들이고, 라우터 기능에서 사용하는 주소록을 관리하게 된다.

```
package RouterLayer.AgentClient.Example.CoordAgent;
                .....
import Abstract.*;
import KQMLLayer.*;
import RouterLayer.AgentClient.*;
// RouterLayer.AgentClient.RouterClientAction을
// 상속하는 CoordAgent 클래스를 작성한다.
public class CoordAgent extends RouterClientAction {
    String _returnValue;
            .....
    public CoordAgent() {
            super();
    }       // 새로운 생성자 작성
    public CoordAgent(Address myaddress,
                      Address routeraddress,
                      Address registraraddress,
                      .....
                      int durationtime) {
// use RouterLayer.AgentClient.RouterClientAction constructor
super(myaddress,routeraddress,registraraddress,durationtime);
    }       .....
```

(그림 5) 라우터의 KQML 처리 모듈

3.3 중재 에이전트

새로운 에이전트가 생성될 때 에이전트는 자신이 처리할 수 있는 작업들에 대해 중재 에이전트에게 알려주어야 한다. 중재 에이전트는 새로운 에이전트에 대한 정보를 관리하며 에이전트의 요청이 있을 때 등록된 에이전트에게 작업을 할당하게 된다. 중재 에이전트(Coordinate Agent)는 메일 에이전트가 보내온 KQML 문의 'purpose'를 읽게 되고, OTS를 통해 어떤 에이전트를 수행 시켜야 할지 결정한 후, ANS를 통해 에이전트의 위치를 파악하여 해당 에이전트에게 메시지를 전달한다. <표 3>은 OTS의 구조를 보여준다. OTS가 가지는 지식 구조는 KQML의 인수들이 처리되는 방식을 그대로 사용하여 인수와 값들의 나열로 구성되어 있다.

〈표 3〉 OTS의 지식 구조

:purpose
{ 진단 0 예약 0 정보 }
:Department
{ 치과 0 내과 0 안과 0 ... }
:Agent
{ DentDiagnoseAgent 0 InternalDiagnoseAgent 0 ... }
#
.....
.....

중재 에이전트는 전달된 KQML 메시지의 'purpose'와 'department'를 읽어 수행할 에이전트를 판단한다. 다음 (그림 6)은 중재 에이전트가 OTS의 정보를 검색 및 분석하여 알려주는 알고리즘이다. getValue()는 전달되어 오는 KQML문에서 인수를 찾아 값을 읽는 함수이다. 미리 정의되어 인수들과 새롭게 정의한 인수들이 읽혀지는 곳이다. takeOffList() 함수를 통해 OTS의 값들을 하나씩 읽어와 비교하여 수행문과 에이전트를 검색하게 된다.

```
try {
    String purp = kqml.getValue("purpose");
    String depart = kqml.getValue("department");
            .....
    Vector v = new takeOffList(OTS);
    // OTS에서 하나씩 읽어서 비교한다.
    while(!v.size()) {
                String token[i] = (String)v.elementAt(i++);}
                addToDeleteBuffer(0);
                if(purp.equals(token[i] return token[i];
    // 일치하면 수행할 에이전트와 수행할 동작을 돌려준다.
            ..... }
                else { sendErrorMessage(kqml);
                       return false;     }
            } catch (Exception e) { return false; }
    return true; }
```

(그림 6) OTS에 의한 에이전트 검색 모듈

새로운 에이전트인 MailAgent는 중재 에이전트인 CoordAgent에 등록되며, 이때 MailAgent의 능력은 'content'에 정의되어 있다. 메시지를 받은 중재 에이전트는 중복된 이름이 존재하는지 검색한 후 'content'에 담긴 내용을 번역하여 ANS에 등록하게 된다. ANS (Agent Name Server)는 현재 등록된 모든 에이전트의 통신유형과 물리적인 주소를 관리하며 새로 들어온 에이전트와 나가는 에이전트를 관리한다. '에이전트 A'가 '에이전트 B'에게 메시지를 보내고자 할 때, B의 물리적 주소에 대해 ANS에 묻게 되고, 이에 대한 대답을 전송 수행문(transport performative)을 통해 처리한다. 에이전트가 생성되면 각 에이전트는 등록 수행문(register performative)으로 ANS에 등록하고, 소멸시에 자신의 엔트리를 삭제한다.

(그림 7)은 중복된 에이전트가 있는지 검색하는 알고리즘이다. equals() 함수를 통해 ANS를 검색해 동일

```
public boolean Compare(Object ans) {
    String stampmsg = (String)ans;
    try {           .....
            if(ans.AgentName.equals(NewAgent) {
                    .....
            return true;            }
      } catch (Exception e) { return false; }
    return true; }
```

(그림 7) 중복된 에이전트의 유무 판단 모듈

한 이름의 에이전트가 존재하는 지 파악한다.

<표 4>는 ANS의 구조를 보여준다. ANS 또한 KQML 표현 형식대로 인수를 관리한다. '#AgentName'은 에이전트의 명칭을, ':Action'은 에이전트 수행문을, ':Address'는 에이전트의 주소를, ':Active'는 에이전트의 상태를 가지는 인수들이다.

〈표 4〉 ANS의 구조

#AgentName
{ MailAgent 0 DiagnoseAgent 0 ... }
:Action
{ SendMailTo 0 sendResult 0 sendQuery 0 ... }
:Description
{ "Send Mail To Parameter" 0 ... }
:Address
{ http://love.keimyung.ac.kr/MARS/sendmail.asp 0 ... }
:Active
{ True 0 False }
#AgentName
{ DiagnoseAgent 0 MailAgent }
:Action
.....

중재 에이전트는 Ontology를 중개하기 위해 OTS (Ontology Type Server)를 관리한다. OTS에는 각각의 Ontology들의 속성(Attribute)과 계층(hierarchy)관계가 정의되어 있다. 상이한 에이전트가 서로 정보를 교류할 때 중재 에이전트는 자신이 처리할 수 있는 영역인지 OTS에 등록되어 있는 Ontology를 살펴보고, 자신이 필요로 하는 것이 정의되어 있으면 처리 요청을 하고 그렇지 않을 때는 추가 요청을 하게 된다.

중재 에이전트는 이 OTS의 도움을 받아서 그 기능을 수행할 수가 있게 된다. 어떠한 질의가 들어 왔을 때, OTS에게 그 질의에 대한 Ontology 정보를 주면 OTS는 자신의 지식 베이스에서 해당 Ontology의 정보를 검색한다. 검색된 결과로 반환되는 수행문에 대한 에이전트의 위치를 ANS를 통해 파악하여, 정보를 해당 에이전트에게 전달한다.

중재 에이전트는 요청된 작업을 어떤 에이전트가 해줄 수 있는지 결정하기 위한 지식을 필요로 한다. 각 응용 에이전트는 생성시 중재 에이전트에게 자신의 이름과 자신이 할 수 있는 기능들을 보내어 중재 에이전트의 지식 부분인 OTS와 ANS에 추가하게 된다.

3.4 메일 에이전트

메일 에이전트(Mail Agent)는 병원 측으로 온 메일들을 검색해서 기본적인 분류 과정을 거치게 되며, 분류가 끝난 메일에 대한 정보를 중재 에이전트에게 넘겨 주게 된다. 메일 에이전트가 하는 일은 다음과 같다. 먼저 메일을 읽어서 본문의 내용을 파싱(parsing)하여 키워드(Keyword)를 추출한다. 메일 에이전트를 어떻게 설계하느냐에 따라 키워드의 내용도 달라질 것이다. 불용어와 조사 등을 제거하고 병인과 관련된 키워드가 추출된다. 키워드의 예로는 두통(headache), 복통(stomachache), 가려움(itch) 등이다. 키워드 추출시 조사를 제거하기 위해 증상 사전을 검색하여 존재하는 단어만을 키워드로 인식하는 방법을 사용하였다.

만약 자연어 검색을 지원하는 에이전트라면 시소러스를 구축하여야하고, 문장의 상관 관계를 통한 의미 분석도 이루어질 것이다. 시소러스 구축과 의미 분석은 본 연구 논문의 범위를 벗어나므로 언급하지 않기로 한다. 다음 (그림 8)은 메일을 읽어서 각 모듈에 작업을 할당하는 메일 에이전트의 일부 코드이다. '<%'와 '%>'는 ASP 코드의 시작과 끝을 나타낸다. Create-Object()를 통해 POP3 서비스 모듈, 분류 모듈을 메모리로 적재해, 메일 서버로부터 메일을 읽고 분류하는 작업을 하게 된다. 이 모듈들은 Active X Server Component로 작성되어 실행시 클라이언트 컴퓨터로 다운로드(download)되어 시스템 라이브러리에 등록되어 실행된다.

```
<%
// pop3 서비스 모듈을 읽어 온다.
set pop = Server.CreateObject("Web.POP3.1")
// 키워드 추출 모듈을 읽어 온다.
set pac1 = Server.CreateObject("packet.mail")
// 간단한 분류를 위한 모듈을 읽어 온다.
set coord = Server.CreateObject("agent.co")
pop.Server = "ielab.keimyung.ac.kr"
            .....
if ret = 0 then
    ret = pop.FindFirstMail
    do while (ret = 1)
    pac1.Subject = pop.Subject
    pac1.from = pop.FromAddr
            .....
    body = pac1.text
    coord.write_mail body
// 키워드가 정장된 파일을 인수로 넘겨준다.
    coord.FileMail = "mail_01.dct"
    coord.Compare_mail
// agent.co 모듈에 의해 분류된 결과를 받는다. (진단 분과)
    department = coord.dest
// agent.co 모듈에 의해 분석된 결과를 받는다. (예약/진단/정보)
    purpose = coord.purpose
    ret = pop.FindNextMail
  loop
  else
  Response.Write "Ret = " & ret & " : POP3 서버에 연결할 수 없습니다."
end If
%>
```

(그림 8) 메일 에이전트 동작 모듈

메일 에이전트는 ASP의 구동으로 초기화되며 POP3 서비스를 통해 메일이 수신되었는지 조사하게 된다. 일단 메일이 수신되면 메일의 본문을 읽어 Ative X Server Component를 통해 본문을 파싱하고 키워드를 추출한다. 추출된 키워드는 또 다른 Active X Server Component를 통해 간단한 분류 작업을 하게 된다. 진단의 경우 내과, 외과, 정신과 등 큰 범주로 분류되고 그 외 예약이나 병원 정보 의뢰 등의 부분으로 분류된다. 분류된 범주에 따라 'Purpose' 인수를 설정하여 중재 에이전트로 보내게 된다.

(그림 9)는 메시지를 KQML 양식으로 전송하는 알고리즘이다. sendDiagnose()는 진단 결과를 KQML 양식에 맞추어 전송하는 함수이다. 진단 에이전트에게 진단을 요청하는 'ask-if'문과 송신 에이전트(sender), 수신 에이전트(receiver), 진단 분과(department) 등의 정보를 실어 전송하게 된다.

```
/* 에이전트에게 결과를 의뢰한다.*/
protected void sendDiagnose(String receiver) {
      String sendmsg = "(ask-if :sender ";
      sendmsg = sendmsg + sender + " :receiver " + receiver;
      sendmsg = sendmsg + " :department " + department +
                " :purpose ";
      sendmsg = sendmsg + purpose;
      sendmsg = sendmsg + " :language KQML :content " +
                _returnValue + ")";
      try {     // 메시지 전송
                sendMessage(sendmsg);
      } catch (ConnectionException c) {
                System.out.println(c.toString());
      } catch (ParseException p) {
                System.out.println(p.toString());}
}
```

(그림 9) 메시지 전송 알고리즘

또한 진단 에이전트로부터 진단 결과가 메일 에이전트에게 전달될 때, 메일 에이전트는 메일의 송신자에게 답장을 보내게 된다. 답장을 보내게 되면 모든 목적이 수행되었으므로 유지하고 있던 메일 정보를 제거한다.

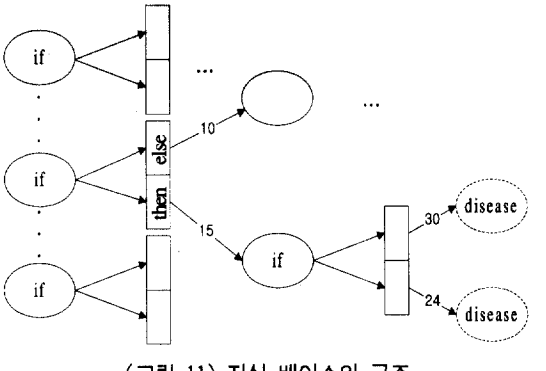
'Purpose'가 '의뢰'인 경우 담당 전문의에게 편지를 보내게 된다. 전문의의 답장이 와서 처리가 완결될 때까지 관련 정보를 유지한다. 답장이 왔을 때 환자에게 결과를 통보하고 진단을 마치게 된다. (그림 10)은 진단 결과를 답장으로 보내는 ASP 코드이다. 답장을 보낼 때는 ASP를 통해 SMTP 서비스 모듈을 실행시키게 된다.

```
<%
Set smtp = Server.CreateObject("Web.SMTP.1")
// 답장을 보내는 서버
smtp.Server = ielab.keimyung.ac.kr
smtp.FromAddr = MailAgent.Addr // 에이전트 email 주소
smtp.FromName = pop.FromName
smtp.To = pop.FromAddr          // 환자의 email 주소
// 환자가 질의한 증상에 대한 진단 결과
smtp.Subect = MA.Content.disease
// 환자의 질병에 대한 치료법(민간요법)
smtp.Message = MA.Content.remedy
// quoted-printable 로 인코딩
smtp.Encoding = "qp"
// 메일을 보낸다.
ret = smtp.SendMail(False)
if ret <> 0 then Response.Write "메일을 보내는데 실패하였습
니다.
%>
```

(그림 10) 답장 전송 모듈

3.5 진단 에이전트

진단 에이전트는 각종 질병 정보의 지식베이스(의학백과 사전 Knowledge Base, 민간요법 Knowledge Base 등)를 구축하고 있다. 이 지식베이스들을 기반으로 중재 에이전트로부터 전달되어온 환자의 증상을 분석하게 된다. 중재 에이전트가 보내온 내용의 'purpose'를 읽어서 해야 할 일이 무엇인지 파악한다. 만약 '진단'이라면, 구축되어 있는 지식 베이스(Knowledge Base)를 통해서 진단하게 된다. 이때 Threshold가 70%이상이면 진단 결과가 정확하다고 판단하여 지식베이스에 의해 판단된 진단 결과(병명, 치료법 등)를 내용(content)에 실어 전송하게 된다.



(그림 11) 지식 베이스의 구조

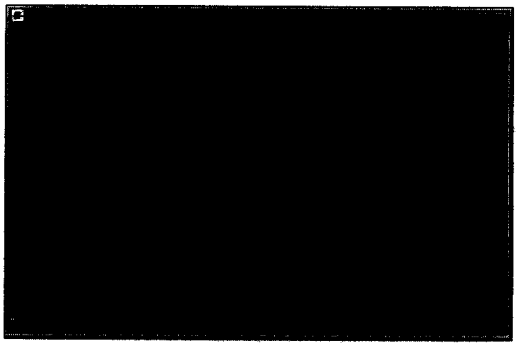
(그림 11)은 지식 베이스의 구조를 보여준다. 지식베이스는 "if ... then...." 구조로 구성되어 있으며, rule base를 따라 검색하는 가운데 Threshold를 구하게 된다. 진단 에이전트는 지식 베이스 구축의 시간적 어려

움으로 인해 간단하게 구성하였다. 지식베이스 구축은 본 논문의 범위를 벗어나므로 언급을 피하기로 한다.

4. 실험 결과

에이전트의 구동은 ASP의 동작으로 초기화되며 ASP의 동작을 위해 NT 4.0 Service Pack 3,IIS(Internet Information Server) 4.0, ASP 1.0을 설치하였다. ASP의 작성은 FrontPage 98 Server Extensions와 FrontPage 98을 이용하였으며, 메일 서버로는 Unix SunOS 5.5를 사용하였다. 키워드 추출과 분류 작업을 위해서 Visual Basic Active X Control을 이용하였으며, 에이전트들의 통신을 위해 JATLite 0.4를 이용하였다. 에이전트의 수행문은 JAVA 1.1.6 으로 프로그래밍 되었다.

새로운 메일이 도착하면 메일 에이전트는 메일의 본문을 키워드 추출 모듈을 이용해 분석하게 된다. 간단한 분석을 통해 내과 진료임을 알게 되고 진단을 목적으로 함을 알게된다. 이때 ASP는 자바 애플릿을 로드(load)하여 구동시킨다.

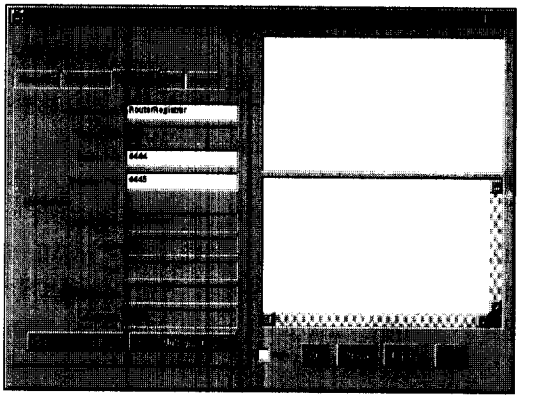


(그림 12) 라우터의 메시지 전달 화면

메일에서 키워드가 추출되면 ASP는 자바 애플릿을 호출하여 메시지를 중재 에이전트의 라우터에게 전달하게 된다. 라우터는 통신 상황을 모니터하며 메시지 전달을 조정한다. (그림 12)는 라우터가 메시지를 전달하는 화면이다. 라우터는 메시지를 전달할 대상이 연결되지 않았을 때 파일에 기록해 두었다가 에이전트가 연결되면 파일에 누적된 메시지를 차례로 전송한다.

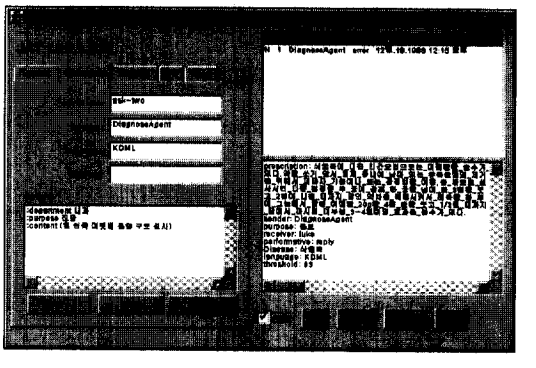
중재 에이전트는 라우터의 동작을 감시하며, 전달되어온 메시지를 분석하여 OTS와 ANS를 통해 실행할 수행문을 결정하고, 에이전트의 위치를 파악하여 메시지를 목적 에이전트로 전송하는 역할을 한다.

(그림 13)은 에이전트를 등록하는 과정을 보여주는 애플릿 실행 화면이다. ANS에 등록되지 않은 에이전트는 중재 에이전트에 등록되어야 하며, 각종 정보들이 ANS와 OTS에 기록되게 된다.



(그림 13) 에이전트의 등록 의뢰 화면

(그림 14)는 진단 결과가 전달되는 화면을 보여준다. 중재에이전트는 메일에이전트가 보낸 키워드를 진단에이전트에게 전달하고, 진단 에이전트의 지식베이스를 통해 진단한 결과를 메일 에이전트에게 전송하게 된다.



(그림 14) 진단 결과가 전달되는 화면

이상과 같이 간단한 메일을 병원 진단 시스템으로 보내어 메일 에이전트가 키워드를 추출하는 것과, 이 정보가 진단임을 분석하여 중재 에이전트에 의해 진단 에이전트로 전달하여 진단 결과가 답장 메일로 전송되는 과정을 확인하였다.

5. 결 론

인터넷의 확산으로 웹을 통해 정보를 제공하는 서비

스가 늘고 있다. 병원의 의사들도 홈페이지를 통해 각종 의료 정보를 제공하고 전자우편을 통해 환자의 질의에 응답해 주고 있다. 그러나 환자는 일일이 웹을 검색하면서 자신의 증상에 맞는 정보를 찾아 다녀야 하고, 의사들도 자신에게 오는 메일을 일일이 읽어보아야 한다. 환자의 질의를 자동으로 분석하고 지식베이스를 통해 진단하여 알려 주는 진단 시스템이 있다면 환자와 의사들에게 편의를 제공할 수 있을 것이다.

복잡하고 다양한 증상들을 진단하기 위해서는 하나의 시스템으로 해결하는 것보다 시스템간의 협동을 통해 해결하는 것이 효율적이다. 멀티 에이전트로 진단 시스템을 설계할 때 각 에이전트는 통신을 통해 정보를 교류하고 협력하여 문제를 해결하게 된다.

본 연구에서는 병원 진단 시스템을 위해 멀티 에이전트 개념을 도입하였고, 병원 진단 에이전트를 구현하기 위한 기본 구조를 설계하였다. 메일 분류와 검색, 진단 등의 작업은 ASP(Active Server Page)를 사용하였고, 통신의 대상이 되는 각 에이전트는 JATLite를 기반으로 한 Java로 작성되었다. 이들 에이전트들은 KQML 메시지를 전송하여 정보를 전달한다. 중재 에이전트는 에이전트들을 제어하여 원활한 통신이 이루어지도록 돕고 각 에이전트의 능력에 따라 작업을 분배하는 역할을 한다. 이때 한 에이전트에서 해결하지 못하는 문제를 다른 에이전트들과의 통신을 통해 해결하게 된다. 실험에서는 KQML을 해석하는 함수들의 역할과 인수들에 대해 중재 에이전트에 미리 정의해 준 다음 실행하였다. 그러나 KIF(Knowledge Interchange Format)나 그 외 에이전트 지식 표현 언어들이 표준화된다면, 에이전트들과의 지식교환을 통해 사용자의 개입 없이도 쌍방의 협조를 통해 문제 해결을 이루게 될 것이다[18].

실험을 통해 상이한 시스템간의 정보 교환이 이루어짐을 확인하였고, 사용자의 빈번한 작업 없이 에이전트에 의해 진단이 이루어짐을 알 수 있었다. 병원 진단 시스템을 멀티 에이전트 개념을 이용하여 구축할 때, 진단에 따르는 여러 가지 절차들을 에이전트가 수행함으로써 시간과 에너지를 절약할 수 있음을 기대할 수 있다.

실험에서는 메일 에이전트, 진단 에이전트, 중재 에이전트로 진단 시스템을 구성하여 에이전트들 사이의 통신의 가능성을 검증하여 보았다. 구현에 있어서 어려운 점은 지식베이스의 구축과 중재 에이전트의 지능성, 키워드 추출 문제 등을 들 수 있다. 지식베이스는 엄청난 양의 정보를 필요로 하고, 많은 정보를 신속하게 검색, 분석 및 분류하는 것은 상당한 시간과 기술을 필요로 한다. 에이전트의 수가 많아지고 지식베이스가 추가 될수록, 중재 에이전트의 역할에 있어서 지능화 연구가 추진되어야 할 것이다. 키워드를 추출할 때 한글에서는 조사의 구분과 유사어 분석 등이 문제점이 되며, 보다 개선된 진단 시스템을 위해서는 키워드에 의한 분석이 아닌 자연어 분석이 이루어져야 할 것이다.

차후 과제로 키워드 추출의 개선이나 자연어 검색, 메일 에이전트의 자연어 처리 문제, 진단 에이전트에서의 지식 베이스 구축 문제, 예약 에이전트와 상담 에이전트 구축과 같은 시스템 확장 문제 등이 연구할 것이다.

참 고 문 헌

[1] 노현철, 이근배, 이종혁, "KQML에 기반한 멀티에이전트 통신환경", 한국정보과학회 제15권 제3호, pp. 47-60, 1997.

[2] 민병의 외 5인, 분산 환경 기반 개방형 에이전트 구조, 정보과학회지, 제15권 제3호, pp.39-46, 1997.

[3] Bird S., "Toward a taxonomy of multi-agent systems," Int. J. of Man-Machine Studies, Vol.39, pp.689-704, 1993.

[4] Charles J. Petrie, Agent-Based Engineering, the Web and Intelligence, IEEE Expert, December 1996, Available as http://cdr.stanford.edu/NextLink/Expert.html.

[5] Cohen P., Cheyer A., Wang M., and Baeg S., "An open agent architecture," Working Notes of AAAI Spring Symposium on Software Agents, pp.1-8, 1994.

[6] Durfee E. H., Lesser V. R., "Partial Global Planning : A Coordination Framework for Distributed Hypothesis Formation," IEEE Tran. Syst. Man Cybernet, Vol.21, No.5, pp.1167-1183, 1991.

[7] Finin T. et al., "DRAFT Specification of the KQML Agent-Communication Language," The DARPA Knowledge Sharing Initiative External Interfaces Working Group, June 1993.

[8] Finin T., Fritzson R., McKay D. and McEntire R., KQML: an information nad knowledge exchange protocol, In International Conference on Building and Sharing of Very Large-Scale Knowledge

Bases, Available as http://www.cs.umbc.edu/kqml/papers/kbks.ps.

[9] Finin T., Labrou Y., and Mayfield J., KQML as an agent communication language. In Jeffery M. Bradshaw, editor, Software Agents, MIT Press, 1995, Available as http://www.cs.umbc.edu/kqml/papers/KQML-acl.ps.

[10] Finin T., Weber J., et. al, Specification of the KQML Agent-Communication Language, The DARPA Knowledge Sharing Initiative External Interfaces Working Group, Feb. 9, 1994.

[11] Fritzson R., Finin T., McKay D. and McEntire R., "KQML - A Language and Protocol for Knowledge and Information Exchange," 13th International Distributed Artificial Intelligence Workshop, Seattle WA, July 28-30, 1994.

[12] Georgff M., "Communication and Interation in Multi-Agent Planning," Proc. of AAAI-83, pp. 125-129, 1983.

[13] http://java.stanford.edu/java_agent/html/, JATLite Homepage

[14] Hyacinth S. Nwana, Software Agents: An Overview. Knowledge Engineering Review, Vol.II, No.3, pp.205-244, October/November 1996.

[15] Michael R. Genesereth, Singh N., "A Knowledge Sharing Approach to Software Interoperation," Technical Reports, Logic Group, Computer Science Dep., Stanford University, 1993.

[16] Michael R. Genesereth, Steven P. Ketchpel, Software Agents, Communication of the ACM, Vol.37, No.7, July 1994.

[17] Michael Wooldridge, Nicholas R. jennings, Intelligent Agents : Theory and Practice, Knowledge Engineering Review, October 1994. Revise January 1995.

[18] Patil R., Files R., Patel-Schneider P., McKay D., Finin T., Gruber T., and Neches R., The DARPA Knowledge Sharing Effort : Progress Report, In Principals of Knowledge Representation and Reasonning : Proceedings of the Third International Conference, November 1992, Available as http://www.cs.umbc.edu/kqml/papers/kr92.ps.

[19] Thomas R. Gruber, "A Translation Approach to Portable Ontology Specifications," Technical Reports, Knowledge Systems Laboratory, Stanford University, 1993.

[20] Thomas R. Gruber, "Toward Principles for the Design of Ontologies Used for Knowledge Sharing," Technical Reports, Knowledge Systems Laboratory, Stanford University, 1993.

[21] Finin T., Fritzson R. and McKay D., "A Language and Protocol to Support Intelligent Agent Interoperability," Proceedings of the CE & CALS Washington '92 Conference, June 1992.

[22] Labrou L., Finin T., A Proposal for a new KQML Specification, TR CS-97-03, Computer Science and Electrical Engineering Department, Uninversity of Maryland Baltimore County(UMBC), Feb. 3, 1997, Available as http://www.cs.umbc.edu/kqml/papers/trcs9703.ps.

[23] Labrou Y. and Finin T., A semantics approach for KQML-a general purpose communication language for software agents, In Third International Conference on Information and Knowledge Management, Available as http://www.cs.umbc.edu/kqml/papers/kqml-semantics.ps.

이 해 수

e-mail : mirr2@orgio.net

1997년 계명대학교 전자계산학과 졸업(학사)

1999년 계명대학교 컴퓨터공학과 (공학석사)

2000년~현재 계명대학교 컴퓨터공학과 박사과정 재학중 (정보공학전공)

관심분야 : 멀티 에이전트, 정보검색

장 덕 성

e-mail : dsjang@kmu.ac.kr

1979년 경북대학교 컴퓨터공학과 졸업(학사)

1981년 서울대학교 전산과학과 (이학석사)

1988년 서울대학교 컴퓨터공학과 (공학박사)

1982년~1985년 동아대학교 전산공학과 조교수

1985년~현재 계명대학교 컴퓨터공학과 교수

1992년~1993년 University of Colorado 방문연구교수

1998년~현재 계명대학교 기획정보처 전산원장

관심분야 : 컴파일러, 시각프로그래밍, 자연어처리, 정보검색, 에이전트 등