□ 기술해설 □

KQML에 기반한 멀티에이전트 통신 환경

포항공과대학교 노현철*·이근배**·이종혁**

● 목 차 ●

1. 서 론

에이전트를 한마디로 정의하긴 힘들지만 몇 가지 특성들을 통해 대략적이나마 기존의 지식 기반 시스템(knowledge based system)과의 구분이 가능하다. 이러한 구분을 가능케하는 특성들로는 다음과 같이 자율성(autonomy), 사회성(social ability), 반응성(reactivity), 선행위성(pro-activeness) 등을 대표적으로 들 수 있다. 하지만 이 중에서도 최근 가장 주목받고 있는 특성은 에이전트의 사회성이다[1].

에이전트는 실제 그 기능성에서 기존의 소프트웨어와 별반 다를 바 없다고도 볼 수 있다. 이는 기존의 소프트웨어들도 다양한 방법으로 지능형 에이전트로 변모할 수 있다는 말이다. 실제 많은 소프트웨어가 랩핑(wrapping) 혹은 재코딩 등의 다양한 방법으로 에이전트화되고 있으며 이러한 추세는 당연한 것이라 할 수 있다[2]. 여기서 우리는 기존의 소프트웨어를 에이전트답게 만드는 부분이 그 자신의 고유한 능력에 근거한 기능성(functionality)이 아닌, 같은 능력이라도 사용자에게 보다 자연스럽고 편리하게 제공해 주는 지능성(intelligence)임을 주목해야 한다. 하지만 하나의 시스템이 사용자에게 제공할 수 있는 기능성이 제한되어 있는 한, 이에 대한 지능성도 한계를 가지게 된다.

이러한 이유로 에이전트는 점차 사회성을 지향하게 된다. 즉, 여러 에이전트들이 서로 상호작용함으로서 서로의 기능을 이용하여 자신들의 지능성을 확대해 나가는 에이전트 사회(agent community)를 이루어 간다. 이를 통해 모든 소프트웨어는 자신의 고유한 기능성만으로도 에이전트 사회에 참여하여 하나의 에이전트로서의 역할을 하게 되며, 필요한 모든 기능성을 다 지니고 있지 않아도 에이전트간의 상호작용을 통해 보다 많은 서비스를 사용자에게 제공할 수 있게 된다.

이러한 에이전트 사회를 실현하기 위해 가장 먼저 해결해야 할 문제는 바로 각각의 독립된 에이전트들이 자신의 목적을 위해 외부 에이전트의 기능을 이용할 수 있고, 경우에 따라선 자신의 기능을 제공할 수 있도록 하는 메카니즘의 확보이다. 이에 대한 가장 보편적인 해결

*비 회 원
**종신회원

책은 에이전트 사회에서 통용될 수 있는 공통된 에이전트 통신 언어(Agent Communication Language, ACL)을 확보하는 방법이다. 물론 이 ACL은 에이전트끼리의 자율적인 문제 해결 방법으로 이용되는 것이므로 기존의 통신 프로토콜과는 상당한 차이가 있다. 충분한 표현력은 물론이고, 각 에이전트의 정보 교환에 필요한 적절한 의미구조를 갖추어야 하며, 무엇보다 에이전트의 가장 큰 특성 중에 하나인 자율성(autonomy)에 적합해야 한다. 이러한 에이전트 사회 구축과 에이전트 통신 언어 제정에 대한 노력이 "지식 공유" 문제로 집약되어 현재까지 많은 연구가 진행되었으며, 지금도 진행되고 있다. 본 논문에서는 이러한 연구 중에서도 대표적인 연구 성과인 KQML의 전반에 대해 알아 보고자 한다.

먼저 2장에서[1] KQML에 대한 소개와 그 구체적인 모습에 대해 알아 보고, 3장에선 실제 이 KQML을 통해 에이전트간 통신 메카니즘을 제공하고 있는 멀티에이전트 환경과 이에 대한 응용 시스템에 대해 알아본다. 이어 4장에선 현재 POSTECH 내에서 진행중인 KQML 멀티에이전트 환경 구축과 KQML을 이용한 응용 에이전트 등 몇몇 관련 연구를 소개한다.

2. KQML 통신표준

KQML은 에이전트 사회에서의 대표적인 에이전트 통신 언어로 [2]ARPA KSE(Knowledge Sharing Effort)의 한 연구 그룹인 External Interfaces Working Group에 의해 제시되었다.

2.1 ARPA의 Knowledge Sharing Effort

KSE는 지식 베이스(Knowledge Base)의 공유와 재사용에 대한 연구를 위해 ARPA, AFOSR(Air Force Office of Scientific Research), NRI(Corporation for National Research Initiative), NSF(National Science Foundation) 등의 지원으로 1990년에 구성된 연구 그룹이다. 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.

이러한 연구는 기본적으로 지식 표현의 문제에 직면한다. KSE는 이 지식 표현과 공유에 대해 계층적 접근 방법(layered approach)을 취하고 다음과 같이 Interlingua, KRSS, External Interfaces, SRKB 등의 네 개의 소그룹으로 구성되며 이 지식 표현 문제의 해당 레벨별 연구를 수행하고 있다[3, 4, 5].

Interlingua working group은 각기 다른 두 표현 언어를 통하여 구축된 지식 베이스 사이의 번역 메카니즘에 관련된 연구를 목적으로 하고 있으며, [3]KRSS working group은 같은 계통의 지식 표현 언어에 기반한 동일 계열 지식 표현 시스템(Knowledge Representation System)군 사이의 표준에 대한 연구를 목적으로, External Interfaces working group은 지식 베이스와 데이타 베이스 사이의 하이레벨 통신 언어에 대한 연구를, [4]SRKB working group은 지식 공유를 위한 공통된 개념 체계(Ontology)에 대한 연구를 목적으로 하고 있다.

이 중 KQML은 세번째 그룹인 External Interfaces 그룹의 대표적인 연구성과에 속한다.

2.2 KQML의 규명

KQML의 현실적인 실용성과 그의 상위레벨 통신언어로서의 특성을 살펴 보면 다음과 같다.

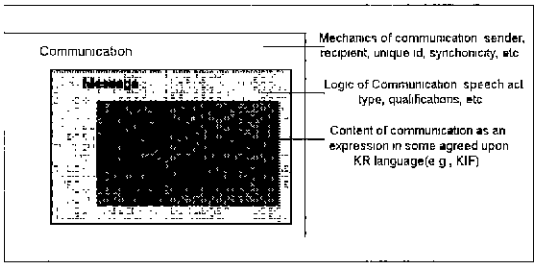

그림 1 KQML 3 Layers

1) Knowledge Query and Manipulation Language
2) Advanced Research Projects Agency
3) Knowledge Representation System Specification
4) Shared, Reusable Knowledge Bases

2.2.1 실용성

KQML은 임의의 지식 베이스(혹은 기능성)들에 대한 응용 시스템을 설계할 때, 개발자들이 다음과 같은 점들로부터 자유로울 수 있다는 데 그 실용적 가치를 찾을 수 있다.

- 지식 베이스(기능성)의 위치와 이를 필요로 하는 곳의 위치
- 지식 표현 방법, 요구 방법 및 전달 방법
- 직접적인 지식 전달 메카니즘

또한 이런 점들을 통해 기존 지식 베이스의 응용 시스템이나 새로이 만들어지는 지식 베이스는, 쉽게 기존의 시스템에 연결될 수 있고 그 기능성을 이용할 수 있다[6, 7].

2.2.2 상위레벨 통신 언어로서의 특징

KQML은 지식 기반 시스템에서의 지식 공유를 위한 상위레벨 통신 언어로서 다음과 같은 특징을 지니고 있다.

- 정보 교환에 관련된 통신 상황에 중점을 둔 통신 언어이다.
- 통신의 내용이 되는 정보의 형태나 의미에 독립적이다.
- 자율적(autonomous)이고 비동기적(asynchronous)이다.
- 쉽게 기존의 소프트웨어들에 랩핑(wrapping)되어 에이전트화될 수 있다.
- 단순한 구문위주의 프로토콜 이상이다.

이 중에서 마지막 특징의 의미는 중요하다. 이는 KQML이 단순한 통신 프로토콜의 구문정의 정도가 아닌, 자신을 이용할 통신 상황에 대한 이해와 이에 기반한 에이전트 시스템의 구조에 대한 이해도 포함하고 있음을 의미한다. 뒤에서 볼 KQML 사양에서는 KQML이 단순한 통신 프로토콜 정도를 넘어, 지식 통신 전반에 대한 의미구조와 이를 위한 에이전트 구조까지도 포함하고 있음을 확인할 수 있다[7, 9, 11].

2.3 KQML의 구조 : 계층 언어(Layered Language)

실제 두 에이전트 간에 오고가는 KQML 메시지 스트림은 내용 표현(content expression)이 메세지 wrapper에 의해 메시지 표현(message expression)으로 싸여서, 이것이 다시 통신 wrapper에 의해 포장된 것으로 이해할 수 있다. 즉, 기존의 통신 프로토콜의 계층 구조처럼, KQML도 그림 1에서 보는 바와 같이 내용 계층(content layer), 메시지 계층(message layer), 통신 계층(communication layer)의 3단계 계층으로 이루어진다[6].

2.3.1 Content Layer

KQML을 사용할 때 소프트웨어 에이전트는 content 메세지를 자신의 언어로 구성한 후, 이를 KQML 메시지 안에 싸서 보내게 된다. content 메세지는 어떤 표현 언어(representation language)로 쓰여지든지, ASCII string이든지, 혹은 binary notation들 중의 하나이던지 상관이 없다. KQML 구현은 content가 언제 시작되고 언제 끝나는가 외에는 message의 content 부분에 관심을 가지지 않는다. 실제 이용되는 KQML 응용 시스템의 경우, KIF, LOOM, Prolog, KRSL 등 다양한 content language를 이용할 수 있다. 심지어 KQML 자신을 content language로 이용할 수도 있다.

2.3.2 Message Layer

내용 계층에 기술된 내용에 부가적인 정보를 포장하여 메세지를 생성해 내는 계층이다. 흔히 화행 계층(speech act layer)이라 불리며 추가되는 부가적인 정보는 주로 content가 어떠한 종류의 화행(assertion, query, response, error 메세지 등)를 띄고 전달되는가에 대한 것이다. 여기서 생성되는 메시지는 그 성격에 의해 크게 두 가지로 나뉘는데, query, assertion 등과 같은 직접적인 지식 전달에 관련된 content 메세지와, 직접적인 지식 전달은 아니나 이러한 활동에 관련된 meta 정보에 대한 것인 declaration 메시지로 나뉘어진다.

특히 이 declaration 메시지는 에이전트 사회에서 자신의 존재를 그 기능성과 함께 선언하거나 자신이 받고자 하는 어떠한 서비스에 대해 등록을 하거나 하는 등의 중요한 부분에 이용된다.

2.3.3 Communication Layer

이는 KQML에서는 가장 바깥 계층으로 위에서 언급된 content layer, 메세지 layer를 거친 메세지를 통신에 관련된 부가 정보를 추가하여 package의 형태로 만든다. 여기에서 추가되는 정보는 sender, receiver와 메세지 ID 등이다.

2.4 KQML 사양

그럼 위에서와 같은 구조를 가지는 KQML의 사양에 대해서 알아 보자. 본 KQML 사양은 위에서 본 KQML의 3계층 중에 가장 중요한 부분인 메시지 계층을 중심으로 이루어진다.

단, 이곳에서는 KQML 사양에 있어서 KQML의 특성을 설명할 수 있는 부분에 대한 것으로 한하고, 구체적인 모습은 참고 문헌에 기재된 KQML 사양 보고서를 참고하기 바란다[7].

외부의 메시지에 자율적으로 반응하는 에이전트의 인터페이스를 기술하는데는 일반적으로 다음과 같은 몇 레벨에 걸친 이해가 필요하다.

Transport : 어떻게 에이전트가 메세지를 보내고 받는가.

Language : 각각의 메세지가 어떤 의미를 띄고 있는가.

Policy : 에이전트들끼리의 대화가 어떤 형태로 이루어지는가.

Architecture : 어떻게 시스템이 구성되는가.

이 중에서 KQML 사양은 주로 transport와 language 레벨의 것이다.

KQML 메세지는 수행문(performative)이란 형태로 정의되어 있다. 이는 각각의 메세지들이 어떤 action을 내재하고 있음을 의미한다. 이 용어는 speech act 이론에서 기인한다. KQML에 기반하여 구축되는 에이전트들은 반드시 정의된 모든 수행문을 지원해야 할 필요는 없다. 또한 본 스펙에 정의되지 않은 수행문도 필요에 따라 정의하여 사용할 수 있다. 하지만 어떤 경우에서든 그 사용과 정의에 있어서 본 사양이 제시하는 스타일에 맞도록 해야 한다.

2.4.1 KQML Transport에 대한 가정

이 부분은 앞서 살펴 본 바 있는 에이전트 인터페이스의 기술을 위한 네 가지 레벨 중 transport에 관련된 것으로 본 스펙의 주된 내용은 아니다. 이는 실제 구현되는 시스템의 환경에 상당히 의존하는 요소로서 많은 다양한 선택이 가능하다. 하지만 KQML은 에이전트 간의 통신에 관련된 언어이기에 어느 정도 메세지 transport에 대한 모델을 필요로 한다. 그래서 다음과 같이 transport 레벨에 대한 추상화된 모델을 제시한다.

- 에이전트들은 단방향 통신(unidirectional communication link)로 연결된다.
- 통신 링크는 메시지 지연을 가질 수 있다.
- 에이전트가 임의의 메세지를 수신했을 때, 그 메세지가 어디에서 온 것인지를 알 수 있다.
- 에이전트가 임의의 메세지를 보낼 때, 그 메시지가 어디로 갈 것인지를 명시해야 한다.
- 임의의 한 통신 링크에 있어서 전달되는 메세지의 순서는 반드시 보내어진 순서여야 한다.
- 메세지 전달은 반드시 안정적이어야 한다.

이러한 모델은 실제 많은 다양한 방법으로 구현이 가능하다. 예를 들어 Internet상의 TCP/IP 연결일 수도 있고, Mailing system의 email path일 수도 있으며, Ethernet상에 구축된 LAN의 UNIX IPC일 수도 있다. 어떤 경우든 위의 추상화된 모델만 만족한다면 KQML은 제대로 동작할 수 있다.

2.4.2 KQML String Syntax

KQML performatives는 기본적으로 ASCII string으로 표현된다. 이에 대한 syntax는 Common Lisp Polish-prefix notation을 따르고 있다. Syntax의 대체적인 모습은 performative name과 parameter list로 이루어진다. 이 중에서 paremeter list는 parameter name과 parameter value의 쌍들로 이루어진 list이다. 여기서 주목할 만한 것은 performative parameter는 그 parameter name으로 구

분될 뿐, 그 순서나 위치와는 무관하다는 점이다. 즉 같은 parameter들의 내용만 담고 있으면 그 list안에서의 순서는 상관이 없다. 구체적인 KQML string syntax는 여기선 생략한다.

2.4.3 KQML 의미론

KQML Semantics는 각 에이전트들의 기능성과 통신 상황에 대한 정형화된 문맥에 바탕을 두고 있다. 각각의 에이전트들은 외부에서 볼 때, 하나의 지식 베이스(Knowledge Base, KB)의 관리자처럼 모델링된다. 이것은 한 에이전트와의 통신은 곧 그에 딸린 KB와의 통신임을 의미하며, 이러한 환경에서 일어날 수 있는 통신 상황은 다음과 같은 것들이다.

- KB가 지니고 있는 내용에 대한 질의
- KB가 지니고 있는 내용에 대한 기술
- KB 안 내용을 추가하거나 삭제하라는 요구
- 지식을 다른 에이전트에 전달하라는 요구

하지만 실제 구현상에 있어서 모든 에이전트가 자신의 KB를 소유하는 것은 아니다. 실제 각 에이전트가 관리하고 있는 것은 경우에 따라 작은 데이타베이스일 수도 있고 단순히 프로그램 내에 있는 하나의 데이타구조일 수도 있다. 이처럼 에이전트의 KB에 해당하는 부분이 다양한 형태가 존재하므로 KB보다는 VKB (Virtual Knowledge Base)라 부른다. 이러한 VKB의 내용은 크게 belief와 goal로 모델링된다. Belief는 에이전트 자신이 혹은 외부의 다른 에이전트가 지니고 있는 정보 그 자체이며, goal은 에이전트 자신이 외부의 환경에 대해 얻고자 하는 상태를 말한다. KQML performatives는 VKB의 이러한 belief와 goal과 관련하여 있을 수 있는 상황에 기반한다[7, 8].

2.4.4 Reserved Performatives

KQML 스펙은 그 구현에 있어서 개발자에게 많은 융통성을 제공하고는 있지만 본 사양에서 제시하는 performative에 한에서는 반드시 사양을 따를 것을 요구하고 있다. "Reserved" performative는 이러한 의미를 함축하고 있다[7, 9].

이 performative는 performative name과 parameter list로 실체화된다. 여기서 이용되는 parameter들은 키워드라 불리우는 parameter들로 구성된다. 각 parameter들의 종류와 그 의미는 참고문헌을 보기 바란다.

여기서 주목할 만한 것은 모든 parameter를 늘 다 필요로 하는 것은 아니며, 각 performative마다 필요로 하는 parameter가 다를 수 있고, 심지어 같은 performative에도 상황에 따라 이용되는 parameter가 다를 수 있다는 점이다. 실제 이용되는 형식은 다음과 같은 형태이다.

performative—name
: parameter—name parameter—value
: parameter—name parameter—value
.....

현재 KQML 스펙에는 30여 개의 다양한 performative들이 정의되어 있다. 이러한 Reserved Performatives는 다음과 같이 그 semantics에 따라 9개의 범주로 나뉘어진다. 이 각각의 의미에 대해선 참고문헌을 보기 바란다.

basic informative-tell, untell
basic query-evaluate, reply, ask-if, ask-about, ask-one, ask-all, sorry
multi-response query-stream-about, stream-all
basic effector-achieve, unachieve
generator-standby, ready, next, rest, discard, generator
capability-definition-advertise
notification-subscribe, monitor
networking-register, unregister, forward, pipe, break
facilitation-broker-one, broker-all, recommand-one, recommand-all, recruit-one, recruit-all

2.5 KQML에 기반한 에이전트의 구조

KQML을 따르는 에이전트의 일반적인 구조는 그림 2에 보는 바와 같다. 크게 application, handler functions/interface module, [5]con-

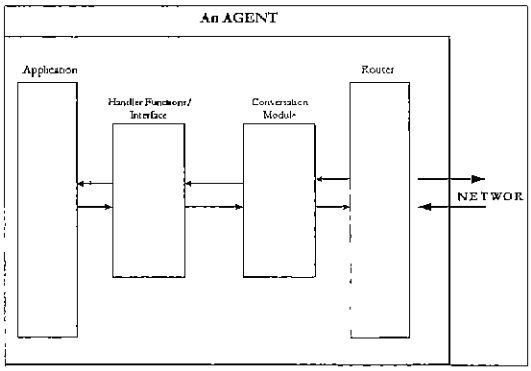

그림 2 KQML에 기반한 에이전트의 일반적인 아키텍쳐

versation module, router의 네개의 부분으로 나눌수 있다[9, 10].

2.5.1 Application

Application의 경우 시스템 내의 어느 지점에서 외부의 정보를 필요로 하며 외부와의 정보 교환이 이루어지느냐 하는 점이 주된 관심사이다. 이는 시스템의 설계시 설계자가 정확히 이해하고 있어야 하는 부분이다. 이와 관련하여 외부와의 정보 교환시 누구에게 무엇을 요구해야 하는가에 대한 문제가 발생하게 된다. 즉, 어떤 정보를 어떤 에이전트로부터 얻느냐 하는 문제이다. 이러한 문제는 아래의 몇가지 방법으로 구현할 수 있다. KQML은 아래 언급된 모든 approach의 구현을 지원한다.

만일 자신이 정보를 얻을 수 있는 외부 에이전트들의 질의 응답 능력에 대해 application 내에 저장해 놓을 수 있다면, query가 외부의 에이전트에 의해 응답되어져야만 할 때, application은 미리 누구에게 query를 던져야 할지 알게 된다. 이는 위의 그림 3에 해당된다.

만약 application이 open system들로 구성되는 환경에서라면, 필요로 하는 정보에 대한 query를 facilitator에게 적절히 운반하라고 요청하여 그 결과를 다시 facilitator로부터 전달받을 수 있다. 이는 위의 그림 4에 해당한다.

application은 facilitator 혹은 다른 에이전

5) Yannis Labrou가 그의 박사논문[10]에서 정의한 일반 구조에 새롭게 추가된 부분으로 아직 표준에는 반영되지 않은 부분이다.

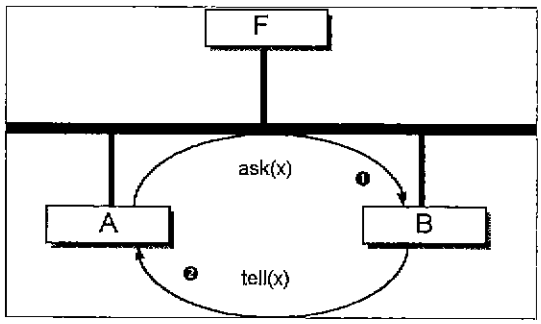

그림 3 A가 B를 알고 있어, B에게 직접 질의한다

트에게 query의 해결에 대해 도움을 청하거나, 같이 상호 협조하면서 이를 해결해 나갈 수 있다. 이러한 방법은 단순히 하나의 query를 처리하기 위해서 뿐만 아니라 기타 application에 필요로 하는 적절한 정보를 수집하는데 크게 도움이 될 수 있다. 이러한 방법이 open system architecture에 가장 적합한 방법이다. 이러한 시나리오는 그림 5를 통해 확인할 수 있다.

2.5.2 Handler Functions and Interface Module

Application 개발자는 다양한 performatives를 처리할 함수(handler function)들을 제공해야만 한다. 예를 들어서, ask-if performative에 대해, handler function은 application에 접근하여 performative의 내용으로 전달된 query에 대해 application내에서의 truth status를 체크하여 그 결과에 대해서 query를 만든 에이전트에게 응답해야 한다. Application

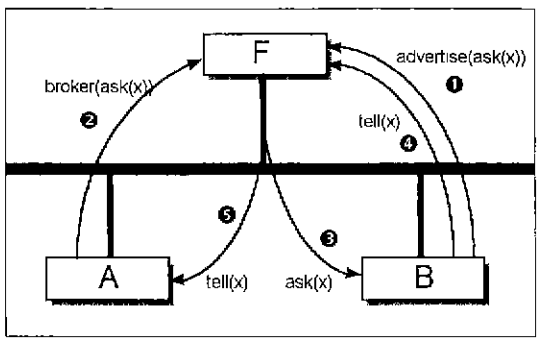

그림 4 A는 F에게 의뢰한다. F는 대신 B에게 질의하여 결과를 A에게 돌려준다

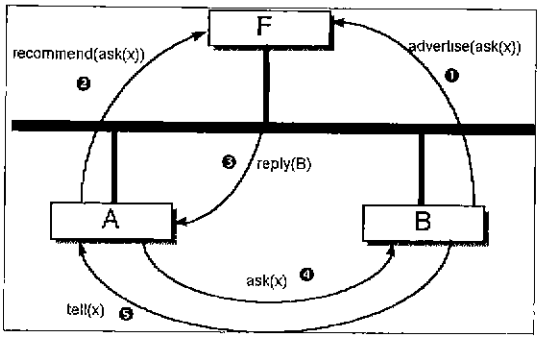

그림 5 A는 F에게 물어 보아 B의 위치를 알아낸 후 B에게 직접 질의한다

개발자는 handler function을 제공하기 위해 다양한 performative들의 정확한 의미를 파악해야 한다. 또한 이 작업은 conversation policy를 수립하는 것과 병행되어야 하는데, 이는 아래의 conversation module에서 담당한다.

2.5.3 Conversation Module

Conversation module은 router와 handler functions/interface module 사이에 위치한다. 모든 메세지는 반드시 conversation module을 통해 나가야 한다. 이 module에는 conversation policy가 구현되어, 다른 에이전트와 통신 상황에서 적절한 conversation context를 제어하게 된다.

즉 수신되는 KQML 메시시가 현재의 통신 상황에서 적절한가를 판단하고 이에 대해 적절한 conversation context를 형성해 나가기 위한 다음 KQML 메시지를 결정해 나간다.

2.5.4 Router

Router는 자신과 관계된 application으로부터의 모든 KQML 메세지를 처리한다. 각각의 KQML speaking 에이전트는 자신만의 Router process를 가지지만 공유도 가능하다. Router는 위의 다른 부분과는 달리 메세지 내의 내용에는 무관하며, 다만 에이전트에게 네트웍내에서의 단일 접속 공간을 제공할 뿐이다. 이 부분은 다른 에이전트와의 다중 동시 연결(multiple simultaneous connection)을 제어한다.

3. KQML의 응용

앞서 살펴 본 바와 같은 모습을 지닌 KQML이 실제 어떻게 이용되고 있는지 알아보자. 먼저 KQML 사양에 따라 개발된 KQML 소프트웨어(혹은 개발환경, API)를 알아 보고 이러한 소프트웨어를 이용하여 실제 에이전트를 구축한 응용 예에 대해 알아 보자.

3.1 KQML 소프트웨어 개발환경

위에서 소개한 KQML 사양에 따라 개발된 KQML API나 프레임웍은 현재 웹상에서 발견할 수 있는 것만도 대략 10개 정도이다. 그 중에서 대표적인 것으로 [6]KATS, JavaAgent, Magenta, LogicWare, KAPI 등을 들 수 있다.

이 중 KATS는 UMBC의 Loral Government Systems Group에 의해 개발된 대표적인 KQML 응용 환경이다. 하위레벨 통신은 TCP/IP API를 이용해 구현되어 있으며, Sun OS 상의 ANSI C와 Lucid Common Lisp 환경에서 각각 구현되어 있다. 둘 다 Unix socket을 통한 네트웍을 이용한다. 가장 큰 특징은 앞에서 잠시 언급한 바 있는 facilitator라 불리는 에이전트의 존재를 가정하고 이를 중심으로 에이전트 사회가 형성된다는 점이다. 이 facilitator를 통해 써비스 요구자와 써비스 제공자들이 각각의 위치를 파악할 수 있게 된다. 전체 구조는 크게 Router, KRIL(Knowledge Router Interface Library), Facilitator 등으로 구성되어 있다[13].

Java Agent(Java Agent Template)는 Stanford 대학의 ABE(Agent-Based Engineering research group)에서 Java 언어를 통해 개발한 template으로, 에이전트 개발자에게 stand-alone 형태는 물론 Java Applet 형태의 에이전트를 쉽게 개발할 수 있는 구조적 환경(structured framework)을 제공한다[12, 14].

Magenta는 Stanford의 Logic Group에 의해 개발된 KIF와 KQML 등의 ACL API다. Magenta의 대표적인 특징으로 들 수 있는 것은 일대일 접속이나 하나의 노드에 여러 노드

6) KQML Agent Technoloty Software

가 연결될 수 있도록 하는 클라이언트/서버 등 다양한 통신 패러다임을 모두 지원할 수 있다는 점이다. 또 API에 [7]Frame-based Model과 같은 간단한 추상화 계층을 추가하여 API를 사용하는 에이전트 개발자들에게 특별한 뷰(view)를 제공하고 있다[15].

LogicWare는 Crystaliz Inc.에서 개발한 환경으로, WWW에서 응용 프로그램들이 상호 협동작업을 할 수 있는 환경을 제공하는 소프트웨어 팩키지의 형태이다. LogicWare는 LogicWare 프로토콜, LogicWare 언어 시스템, LogicWare 서버 등의 세 부분으로 구성되어 있는데, 이 중에서 LogicWare 프로토콜 부분이 KQML 프로토콜을 지원하는 부분이다[16].

KAPI는 [8]SHADE 프로젝트의 일환으로 개발된, 비교적 쉬운 구조를 가지는 KQML API이다. 가장 큰 특징으로는 인터넷 상에서 TCP/IP, MIME email, Mbus, HTTP 등과 같은 다양한 transport 메카니즘을 통한 KQML 메시지 패싱을 제공한다는 점이다. 특히 HTTP를 이용할 경우 에이전트는 WWW 브라우저와 WWW서버에게 메시지를 주고 받을 수 있다. 이러한 특성으로 인해 각 에이전트마다 TCP/IP, HTTP, Mbus 등 서로 다른 하위레벨 통신 프로토콜의 할당이 가능하다. 구조상의 간편함으로 인해 가장 이용하기 쉬운 KQML 환경이다[17].

3.2 KQML 응용

위 3.1에서 소개한 KQML 소프트웨어를 이용하여 구현된 응용 시스템으로 대표적인 PACT와 Next-Link, Agent-K를 살펴 보자.

[9]PACT는 Stanford 대학과 Lockheed Palo Alto Research Labs, Hewlett-Packard, Enterprise Integration Technology 등이 함께 모여 구성된 연구 그룹이다. 이 그룹은 궁극적으로 여러 다른 싸이트에 위치하는(multiple site), 여러 서브시스템들(multiple subsystem), 여러 전문분야(multidiscipline)를 아우르는 concurrent engineering의 기반구조의 확보를 목표로 하고 있다. 이 환경에서 각 시스템들은 지식 기반 통신 언어와 서비스를 통해 서로 상호작용하게 된다. 이러한 PACT 구조는 서로 자율적으로 상호작용하는 에이전트에 기초를 두고 있는데, 이 상호작용은 서로 다른 분야간에 지식 교환을 위한 세가지 레벨의 개념, 어휘 체계의 공유에 기반하고 있다. 이 때 첫번째 레벨의 공통된 ACL로 이 KQML이 이용된다.

그외에 두번째, 세번째 레벨에서는 KIF와 Ontology 등이 이용된다. 이 PACT 프로젝트는 시나리오를 시연하기 위해 앞서 소개한 SHADE의 KAPI를 사용하였다[18].

Next-Link는 Stanford의 Center for Design Research에서 개발한 프레임웍으로 분산된 디자인과 이에 대한 필요를 연결시켜 주는 에이전트 조정(coordination)에 대한 연구이다. 궁극적인 목적은 엔지니어들이 기존의 디자인들을 쉽게 이용 가능하게 하는 것이다. 그러나 실제 많은 디자인들은 다형질(heterogeneous)의 시스템과 툴들을 이용해서 이루어진 것이기 때문에 이를 위해선 다형질의 에이전트들과 이들 사이에 존재하는 디자인, 엔지니어링에 대한 공통된 Ontology를 기반으로 해야 한다. 그래서 이들은 공통된 ACL을 통해 확보되는 조정계층을 통해 Next-Link 구조를 구축하고 이를 조정에이전트로 구축하는 방법을 택했다. 이 Next-Link 구조에서 공통된 통신 인터페이스로 SHADE의 KAPI를 이용하고 있다. 현재는 항공기의 배선 케이블 제어에 관련해서 이 조정 에이전트를 개발 중에 있다[19, 20].

Agent-K는 AOP(Agent-Oriented Programming)와 KQML의 두 가지를 결합시킨 형태이다. AOP는 에이전트의 행위 양상을 기술하고 있고 KQML은 에이전트의 통신을 위한 통신 수단을 제공하게 된다. Agent-K에서 에이전트들을 AOP에 의해 프로그램되고 KQML을 통해 서로 통신할 수 있게 함으로, 구현에 있어서 효율성을 기할 수 있으며 작동가능한 소프트웨어 에이전트들 간의 기능성을 더해 줄 수 있다. SHADE의 KAPI를 이용하고 있다

7) Object creation, deletion, method call, attribute set/get, logical query 등의 수행에 대한 추상화된 모델이다.
8) SHAred Dependency Engineering
9) Palo Alto Collaborative Testbed

[21, 22].

4. POSTECH 멀티에이전트 환경

POSTECH에서 구축하고 있는 멀티에이전트 환경은 그 목적을 웹에 기반한 응용 에이전트 환경 구축에 두고 있다.

크게 두 방향으로 진행되고 있는데, 하나는 기존의 개발된 KQML 환경을 이용하여 현재 연구 중인 몇몇 시스템을 위한 멀티에이전트 환경을 구축하는 것이고, 다른 하나는 기존의 KQML 환경을 대체할 만한 새로운 KQML 프레임웍을 개발하는 것이다.

먼저 새로운 KQML 프레임웍으로 연구중인 PAST를 소개하고, 이어 KAPI를 이용한 두 가지의 응용 에이전트 시스템에 대해 알아 본다.

4.1 [10] PAST

4.1.1 개발 동기

KQML을 이용한 응용프로그램을 작성하는데 있어서 프로그래머에게 가장 필요한 것중의 하나는 KQML 메세지의 통신을 충분히 지원해 줄 수 있는 API나 라이브러리이다. 쉽게 이용 가능한 이러한 KQML 프레임웍의 부재는, 결국 에이전트 시스템 개발자로 하여금 에이전트에 적합한 KQML의 여러 장점들에도 불구하고 KQML이 아닌 다른 통신 메카니즘을 선택하게 한다. 그러나, 현재로서는 충분한 기능성을 지닌 KQML 환경을 쉽게 얻을 수 있는 여건은 안된다. 기존에 개발된 이용 가능한 KQML API의 경우, 그 구현에 있어 어떤 일관된 형태를 띄지 못하거나 그 제공하는 환경이 프레임웍이라고 하기에는 많은 부족한 점이 있는 등 문제점이 있다. 예를 들어 SHADE KAPI[17]의 경우, 그 구조가 간단해 이용하기에는 용이한 면이 있으나 실제 KQML의 가장 중심이 되는 Discourse Semantics을 지원할만한 모듈은 존재하지 않고 있다. 이러한 문제점을 제기한 PAST팀은 사용자의 입장에서 가장 이해하기 쉬운 형태로 일반화된 체계를 구축하고, KQML의 효용성을 최대한 살릴 수 있는 KQML 소프트웨어 개발을 목표로 프로젝트를 시작하였다.

4.1.2 개발 환경

PAST의 구현 언어로 선택한 것은 자바이다. 자바 언어를 선택한 이유는 다음과 같다.

- 네트워크 라이브러리 등 쉽게 이용 가능한 풍부한 라이브러리로 개발시간을 단축시킨다.
- KQML의 장점인 비동기적인 처리(Asynchronous Processing)를 하는데 있어서 쓰레드(Thread)가 효과적으로 제공된다.
- 또한, 데이타 타입의 오버라이드가 손쉽게 이루어져 인터넷등에서 제공되는 자바 라이브러리들을 사용하는데 불편함이 없다.
- 자바로 만들어지는 소프트웨어는 자바 언어가 가지는 이식성등을 자연스럽게 가짐은 물론, 인터넷 환경에서 애플릿과 연동할 에이전트들의 가장 좋은 도구가 될 것이다.

실제 자바 언어를 사용함으로 이러한 언어적 특성들이 개발시간을 상당히 단축하여 줌을 확인하였다.

4.1.3 PAST의 구조

현재 PAST는 사용자에게 제공될 에이전트 라이브러리와, 서비스 에이전트들로 구성되어 있다. 여기서 서비스 에이전트들은 PAST에서 개발한 KQML 에이전트 라이브러리를 이용하여 개발한 utility 프로그램이다. 이는 앞서 살펴본 바에 있는 facilitator에 해당하는 기능성을 지닌 에이전트들이다.

우선 모든 라이브러리는 past라는 이름의 package로 구현되었다. 이 package는 다음과 같은 구조로 나뉘어져 있다.

User Interface Level—프로그래머가 사용해야 할 라이브러리 함수들은 모두 past.Agent에 구현되어 있다. KQML메세지를 파싱하는 부분과, 에이전트의 정보를 보관하는 부분이 포함되어 있다. 에이전트의 정보는 에이전트 이름, 물리적 주소, 통신 형태(TCP/

10) Postech Agent Software Technology

IP, HTTPD. SMTP)로 나누어져 있다. 이 정보는 ANS(Agent Name Server)에 등록되고, 이 정보에 따라 통신레벨에서 통신의 형태에 상관없이 메세지를 전달하도록 설계되어 있다. 현재는 TCP/IP만을 지원하고 있다.

Communication Level—이 레벨은 past. ACM(Agent Communication Mediator)에 구현되어 있다. 여기에는 KQML의 통신부분이 포함된다. 주된 기능은 앞서 살펴 본 바 있는 KQML Agent Architecture에서 Conversation Module과 Handler function부분에 해당한다. Conversation policy는 callback 함수의 형태로 구현되어 있다. 외부로 나가는 메세지는 응답을 필요로 하는 메세지와 응답을 필요로 하지 않는 메세지로 나눌 수 있는데, 만일 응답을 필요로 할 경우에는 응답이 들어올 경우 불려질 callback 함수를 설정함으로 conversation context를 형성해 나간다. 외부에서 들어오는 메세지는 질문에 대한 응답인 경우와 그렇지 않은 경우로 나눌 수 있는데, 이는 KQML 메세지의 "in-reply-to" 필드와 "reply-with" 필드와 함께 callback 함수를 저장해 놓은 Hash 테이블을 참조해 파악할 수 있다.

Knowledge Base Level—이 레벨은 past. KB에 구현되도록 디자인되었다. KQML 소프트웨어에는 에이전트들이 사용하게 될 많은 지식 기반 툴들이 포함되어야 할 것이다. 예를 들면 KIF(Knowledge Interchange Format) 파서나 LISP 해석기 등을 들 수 있을 것이다. 현재 이 레벨에는 테스트 프로그램으로 만든 간단한 프레임 언어가 포함되어 있다.

이러한 라이브러리를 통해 현재 구현되어 있는 써비스 에이전트들로는 ANS, Broker, Visualizer 등이다. 이들의 기능에 대해 살펴 보자.

ANS—에이전트는 다른 에이전트와의 통신을 물리적 주소가 아닌 에이전트의 이름을 사용하게 된다. 이 때 이름을 가지고 물리적인 주소와 통신 형태를 판단하여 주는 것이 ANS이다. past.ACM에서는 이러한 에이전트 정보들을 캐쉬시키는 기능을 포함하고 있고, 만일 캐쉬에 저장되어 있지 않을 때는 자동으로 ANS에 물어서 물리적 주소를 알아 메세지를 전달하도록 효율적으로 설계되어 있다.

Broker—Broker는 아직까지 완전한 구현은 되어 있지 않고 프레임 기반의 Ontology Server와의 연동을 연구하고 있다.

Visualizer—Visualizer는 전체 시스템의 모든 메세지를 관리하는 에이전트이다. 이 에이전트는 "subscribe" performative를 에이전트들에게 보내어 에이전트들이 메세지를 보낼 때 Visualizer에게도 알려주도록 하여 구현되었다[24].

4.2 AIR-Web의 멀티에이전트 환경

4.2.1 [11]AIR-Web

AIR-Web은 한국어 담화 분석을 이용한 세션(session) 기반의 웹 검색 시스템이다. 기존의 웹 검색모델과는 달리 세션 기반의 사용자 검색 모델을 채용한 웹 검색 시스템으로 현재 개발중인 시스템이다. 아직까지는 IR분야보다는 한국어 자연어 처리 기술, 담화 분석 기술 등을 이용한 세션 기반 사용자 검색 모델을 중심으로 하고 있어, 사용자 인터페이스 부분이 강화된 웹 검색 도구이다. 앞으로 한국어 자연어 처리를 IR쪽으로 확대하여 한국어의 특성에 적합한 문서 인덱싱을 추구해 나갈 계획이다[23].

4.2.2 전체 시스템 Architecture

AIR-Web은 시스템 전반이 멀티에이전트 구조를 취하고 있어 각각의 중요한 기능들이 하나의 독립된 에이전트의 형태로 제공된다. 여기에서 각 에이전트 상호간의 통신을 위해 KQML을 이용하고 있다. 그림 6에서 보는 바와 같이, 전체 시스템은 크게 HCI Agent, Web Search Agent, Web Indexing Agent 등으로 나뉘어지고, 이들 각각이 KQML로 서로 통신하며 독립된 에이전트로 동작한다.

AIR-Web은 이러한 멀티에이전트 프레임웍을 구축하는 과정에 앞서 살펴 본 KAPI를 이용하였다. KQML interface부분은 앞서 KQML 부분에서 제시한 바 있는 KQML

11) AIR-Web : Agent-based Natural language Information Retrieval on the Web

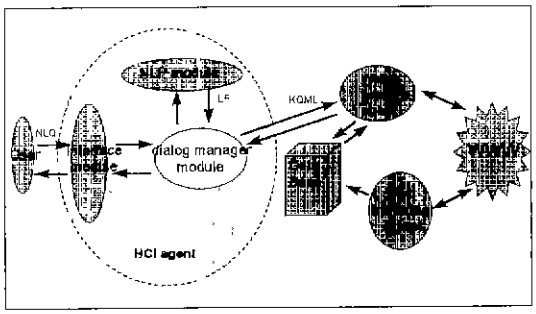

그림 6 AIR-Web System Architecture

speaking 에이전트의 일반적인 구조를 그대로 따르고 있다.

4.3 Web Resource Referential Integrity Maintenance

4.3.1 웹에서의 Broken-link 문제

WWW에서 구조적인 문제점으로 거론되고 있는 것 중에 하나가 바로 broken-link이다. 웹의 주요한 기능으로 이용되는 하이퍼 링크가 단방향 참조로 이루어짐으로 인해 생기는 이 문제는 기본적으로 웹 자원과 이 웹 자원을 참조하고 있는 참조자(참조하고 있는 싸이트) 간에 발생한다. 즉, 웹 자원에서의 변화(delete, update, migrate 등)가 참조자에게 직접적으로 전달될 수 있는 메카니즘이 없음에 기인한다.

4.3.2 Referential Integrity 확보를 위한 노력

이러한 broken-link 문제는 아주 일반적이면서 심각한 문제이므로, 이를 해결하여 하이퍼 링크의 참조 일관성(Referential Integrity)을 확보하기 위한 노력도 그동안 다양한 방법으로 있어 왔다. 그 노력의 결과로 몇몇 방법들이 제시되고 실제로 이러한 문제를 해결해 주는 소프트웨어도 개발되어 있으나, 이러한 대부분의 방법들은 참조하고 있는 웹 자원의 상태를 주기적으로 체크함으로 이를 해결하고자 하는 단방향 통신에 기반한 방법에 의존하고 있다. 이는 비록 손쉬운 해결 방안이기는 하나, 웹 자원의 변화가 이를 참조하고 있는 쪽에서 체크하기 전까지는 하이퍼 링크에 직접 반영될 수 없기 때문에 잠재적인 broken link가 존재한다.

4.3.3 KQML 대화 환경에서의 Broken-link 문제 해결

만일 서로 자연스럽게 대화할 수 있는 프로그램들이 있다면 이러한 문제는 쉽게 해결될 수 있는 문제다. 이러한 프로그램의 개발이 KQML speaking Agent라는 형태라면 충분히 가능한 일이다. 즉 KQML 환경에서라면 이러한 문제 해결에 대한 시나리오만 제공해 주면 나머지는 각각의 KQML speaking Agent끼리 자율적으로 해결이 가능하다는 말이다. 이러한 방법은 KQML이라는 ACL을 이용한 쌍방향 통신에 기반하므로 웹 자원 쪽의 변화가 이를 참조하는 쪽에게 즉각적으로 전달될 수 있다.

실제 이 문제를 해결하기 위해 개발 중인 WWW Referential Integrity Maintenace Agent의 구조는 그림 7과 같다[25]. 한 서버 내에 이러한 구조의 에이전트가 하나씩만 있으면 그 서버내의 모든 웹 하이퍼 링크를 관리할 수 있다. 이 중에서 KQML 통신을 위한 Interface부분은 앞서 AIR-Web에서 이용한 모듈을 그대로 재사용하고 여기에 Referential Resolution 시나리오 부분을 추가하여 보다 지능적인 대화 능력을 지니도록 보완하였다.

5. 결 론

기존의 에이전트는 독립적인 기능성만으로

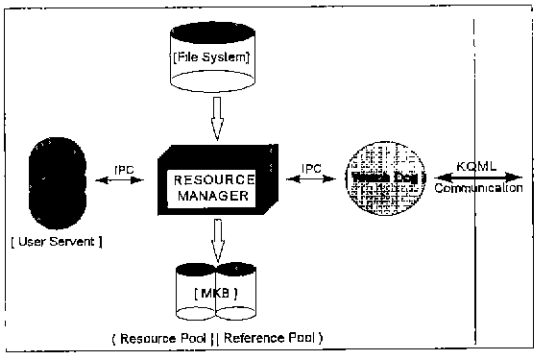

그림 7 WWW Referential Integrity Mainte nance Agent

평가받아 왔으므로 독립적으로 보다 많은 기능성을 보유해야 했다. 하지만 이렇게 구현된 에이전트의 기능은 단지 그 자체를 위한 것일 뿐 다른 환경이나 다른 장소에 있는 에이전트에 이용될 수 없을 뿐만 아니라, 비슷한 에이전트들의 기능성이 중복되어 개발되는 비효율성을 안고 있다. 하지만 최근 주목받고 있는 멀티에이전트 환경에서의 에이전트들은 자신의 고유한 기능만을 구현한 후, 다른 에이전트와의 통신을 위한 언어 능력만을 갖추게 된다면 에이전트 사회에서 하나의 에이전트로서 충분한 능력을 갖춘 셈이 된다.

이는 에이전트 시스템 개발자 입장에서도 큰 장점이 아닐 수 없다. 시스템의 확장성이나 가변성을 확보할 수 있을 뿐더러 무엇보다 시스템 개발의 효율성으로 인해 그만큼 자신의 고유 능력을 특화할 수 있는 여유를 가질 수 있다. 또한 이렇게 구축된 멀티에이전트 시스템은 얼마든지 새로운 환경에 적응하여 새로운 모습으로 변형될 수 있고, 이러한 몇몇 에이전트들을 조합하여 새로운 시스템을 쉽게 구성할 수도 있다.

우리는 이러한 일을 가능케 하는 대표적인 매카니즘, 즉 에이전트 통신 언어인 KQML의 사양과 응용에 대해 알아 보았다. 그리고 이러한 KQML 환경을 제공하고 있는 소프트웨어와 이러한 환경을 이용하고 있는 실례를 몇 가지 살펴 보았다. 이러한 멀티에이전트 환경, 즉 에이전트 사회에서의 사회성은 실용적 측면에서의 효율성은 물론, 인간 사회의 그것과 같이 자신의 목표를 남의 도움을 받으며 함께 이루어 가는 모습에서 미래 컴퓨터 프로그램의 진정한 모델을 제시해 주고 있다.

참고문헌

[1] Woodridge, M. et al., "Agent Theories, Architectures, and Languages : A Survey", In ECAI '94 Workshop on "Agent Theories, Architectures, and Languages, 11th European Conference on Artificial Intelligence, Amsterdam, The Netherlands, August 8-12, pp. 1-32, 1994.

[2] Genesereth et al., "Software Agents", Communications of the ACM 37 (7, July), pp.48-53. 1994.

[3] Robert Neches "Overview of Knowledge Sharing Effort", Robert Neches's executive summary of the ARPA KSE, Nov 1993.

[4] Robert Neches, et al, "Enabling Technology For Knowledge Sharing", AI Magazine, Volume 12, No. 3, Fall 1991.

[5] R.S. Patil et al., "The DARPA Knowledge Sharing Effort : Progress Report", In Proceedings of KR '92-The Annual International Conference on Knowledge Representation, Cambridge, MA, 1992.

[6] T.Finin, Don Mckay, Rich Fritzson, the KQML Advisory Group(Eds.), "An overview of KQML : A Knowledge Query and Manipulation Language", March 2, 1992.

[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] T. Finin et al., *"Specification of the KQML Agent-Communication Language plus example agent policies and architectures", unpublished draft, 1994.*(available at http : //www-ksl.stanford.edu/knowledge-sharing/papers/README.html)

[9] Yannis Labrou and Tim Finin, "A Semantics Approach for KQML-A General Purpose Communication Language for Software Agents", In the Third International Conference on Information and Knowledge Management (CIKM' 94), November 1994.

[10] Yannis Labrou, "Semantics for an Agent Communication Language" A Doctoral Dissertation, Unversity of

Maryland, Baltimore, Maryland, 1996.
[11] Tim Finin et al., "KQML as an Agent Communication Language", In The Proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94), ACM Press, November 1994.
[12] James Mayfield et al., "Evaluation of KQML as an Agent Communication Language", In Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages. M. Wooldridge, J. P. Muller and M. Tambe (eds). Lecture Notes in Artificial Intelligence, Springer-Verlag, 1996.
[13] Unisys Corporation, "Sotfware Design Document for KQML" (available at http : //www.cs.umbc.edu/kqml/software/kats)
[14] Rob Frost, "Documentation for the Java(TM)Agent Template, Version 2. 0" (available at http : //cdr.stanford. edu/ABE/JavaAgent.html)
[15] HREF http : //hpdce.stanford.edu/ magenta.html
[16] Sankar Wirdhagriswaren, "LogicWare Technology : Foundation for Next Generation, World Wide Web, Collaborative Applications", (available at http : //www.crystaliz.com/logicware/ white.html)
[17] Pointers for final Shade report (available at http : //www.eit.com/creations /research/shade/)
[18] M.R. Cutkosky et al., "PACT : An Experiment in Integrating Concurrent Engineering Systems", IEEECM, volume 26, No. 1, pp.28-37, jan 1993.
[19] Charles J. Petrie et al., "Using Pereto Optimality to Coordinate Distributed Agents", AIEDAM special issue on conflict management Vol. 9, pp. 269-281, 1995.
[20] HREF http : //cdr.stanford.edu/NextLink/NextLink.html
[21] W. Davies, P. Edwards, "Agent-K : An Integration of AOP and KQML", In the Third CIKM'94, National Institute of Standards and Technology Gaithersburg, Maryland, Friday, December 2, 1994.
[22] HREF http : //www.csd.abdn.ac.uk/~ pedwards/publs/agentk.html
[23] 노현철, 권혜진, 이근배, 이종혁, "한국어 담화 분석을 이용한 Session 기반의 웹 검색 시스템 : AIR-Web", In 한국정보과학회 25회 학술논문발표집, 1996.
[24] HREF http : //www.postech.ac.kr/~ zzuzzu/past.html
[25] 노현철, 이근배, 이종혁, "KQML을 이용한 멀티에이전트 환경하에서의 웹 자원 관리 에이전트 설계", In HCI "97 학술대회 발표 논문집, 1997.

노 현 철

1996 포항공대 전자계산학과 졸업(공학사)
1996~현재 포항공대 대학원 전자계산학과 석사 과정
관심분야 : 에이전트, 한국어처리, 정보검색 등

이 근 배

1984 서울대학교 컴퓨터공학과 졸업
1986 서울대학교 컴퓨터공학 석사학위 취득
1991 미국 UCLA 전자계산학과 박사
1984~1986 서울대학교 연구조교
1987~1991 UCLA 전자계산학과와 생명과학과에서 연구조교와 연구원으로 근무
1991~현재 포항공대 조교수
관심분야 : 자연어처리, 인공지능/신경망, 음성인식, 정보검색 등

이 종 혁

1980 서울대학교 수학교육과 (이학사)
1982 한국과학기술원 전자계산학과(이학석사)
1988 한국과학기술원 전자계산학과(공학박사)
1989~1991 일본 NEC C&C 정보연구소(초빙연구원)
1991~현재 포항공과대학교 전자계산학과 부교수

관심분야 : 자연언어처리, 한국어처리, 기계번역, 자동통역, 정보검색 등

● 통신정보합동 학술대회 ●

- 일 자 : 1997년 4월 17~19일
- 장 소 : 부산 해운대
- 주 최 : 정보통신연구회
- 문 의 처 : 한국전자통신연구원 최문기 박사
  T. 042-860-6100

● 제24회 임시총회 및 춘계학술발표회 ●

- 일 자 : 1997년 4월 25(금)~26일(토)
- 장 소 : 한림대학교
- 발표논문 접수마감 : 1997년 3월 8일(토)
- 문의처 및 접수처 : 한국정보과학회 사무국
  T. 02-588-9246, F. 02-521-1352
  서울시 서초구 방배3동 984-1(머리재빌딩) ㉮137-063