大 韓 造 船 學 會 論 文 集
第 36 卷 第 3 號 1999年 8月
Journal of the Society of
Naval Architects of Korea
Vol. 36, No. 3, August 1999

에이전트 기반의 시스템 통합을 위한 에이전트 기본 아키텍쳐에 관한 연구

이상욱*, 이규열**

A Study on the Basic Architecture of an Agent System for Agent-based System Integration

by

Sang-Uk Lee* and Kyu-Yeul Lee**

요 약

본 연구에서는 에이전트 기본 아키텍쳐를 설계하였고 다른 에이전트에게 관심있는 정보를 등록하는 메시지, 처리할 수 있는 업무를 표시하는 메시지, 정보를 알려주는 메시지, 정보에 대해 질문하는 메시지를 처리하는 KQML 처리기의 기본적인 알고리즘을 구현하였다.

에이전트 기본 아키텍쳐는 KQML 처리기와 KIF 번역기, 제어코드(Flow control code)로 구성되어 있다. 에이전트는 전달하고자 하는 지식을 KIF(Knowledge Interchange Format)로 표현하여 메시지 통신을 위한 외부 언어인 KQML(Knowledge Query and Manipulation Language)을 통해 실어나르는데 KQML 처리기가 메시지 통신을 관리한다. KQML 처리기를 통해 전달된 메시지의 내용은 KIF 번역기에 의해 저장되고 해석된다. 제어 코드는 프로그램의 수행 순서 및 정보의 흐름을 제어하여 실제 목표하는 일을 수행하는 프로그램 코드이며 KIF 번역기의 지식 베이스(Knowledge Base)에서 필요한 정보를 얻어서 내부 실행 코드와 외부 프로그램을 관리한다.

Abstract

In this paper, the basic architecture of an agent system was designed and a KQML(Knowledge Query and Manipulation Language) handler was implemented to handle 'tell', 'ask', 'handles' and 'interested' KQML performatives.

The basic architecture of an agent system consists of a KQML handler, a KIF interpreter and a Flow control code. Agents use KIF(Knowledge Interchange Format) to represent the actual knowledges that are transmitted. They communicate others via an external language called KQML, which contains

접수일자 : 1998년 9월 16일, 재접수일자 : 1999년 4월 9일

*학생회원, 서울대학교 조선해양공학과 대학원

**정회원, 서울대학교 조선해양공학과

contents of messages written in KIF. The KQML handler controls communication. Contents of messages through it are stored and interpreted by the KIF interpreter. The flow control code controls the flow of program and information and performs engineering tasks. It gets knowledges from the knowledge base of the KIF interpreter and the other agents.

1. 서론

조선 산업은 수주 산업으로 선주들의 요구에 대해 매번 새로이 설계, 건조를 해야 하기 때문에 설계, 생산계획, 생산 업무를 비교적 단기간에 수행하여야 한다. 이런 배경하에서 현재의 설계업무는 단계별, 분야별로 수행되고 있으며 각각의 설계분야들은 복잡한 상호 관계를 맺고 있기 때문에 효과적이고 빠른 설계를 위해서는 그에 맞는 다양한 툴과 방법을 사용하여야 한다.

그러나 상이한 시스템간의 정보교환수단이 미비하기 때문에 현재의 정보의 일관화/공유화를 위한 수단으로는 설계영역 전반에 걸친 시스템의 통합에 어려움이 있다.

에이전트 기반 시스템은 이런 문제를 효율적으로 해결할 수 있는 방안의 하나가 될 수 있다. 에이전트 기반 시스템은 시스템간의 상이한 입출력 규약에 따른 정보 교환의 문제점을 매우 유연하고 강력한 기능을 갖는 에이전트 통신 언어(Agent Communication Language; ACL)를 사용하여 완화시킬 수 있으며 적절한 시기에 분산된 시스템의 각 컴포넌트에게 정보를 전달할 수 있다.

2. 에이전트 시스템

2.1 에이전트의 정의

에이전트라는 용어는 여러 분야에서 널리 사용되고 있지만 각각에 따라 조금씩 다른 정의와 개념을 사용하고 있다[1]. 본 연구에서는 Stanford 대학교 Computer Science Department의 Logic Group에서 연구하고 있는 Agent-based software Engineering(ABE)의 관점에 따라 에이전트를 살펴보기로 한다[2].

ABE의 관점에서 에이전트는 ACL로 다른 에이전트와 통신하는 소프트웨어 컴포넌트이다. 즉, 에이전트는 상황에 따라 적당한 ACL을 주고받으며 다른 에이전트와 정보 및 서비스를 교환하고 협동적으로 일을 수행한다. 이때 ACL의 표현력은 에이전트가 수행할 수 있는 일의 범위를 결정짓는 중요한 요소로, 여기서는 Knowledge Query And Manipulation Language(KQML)과 Knowledge Interchange Format(KIF)를 각각 외부언어와 내부언어로 사용한다.

2.2 에이전트 연방 시스템 (Federated System)

에이전트 연방 시스템은 퍼실리테이터(facilitator)라는 특별한 에이전트를 통해 연결되는 에이전트들의 집합을 말한다. 퍼실리테이터는 각 통신에 대한 부담을 덜어주고 조정과 중재의 역할을 맡기도 한다.

에이전트 연방 시스템에서 에이전트는 퍼실리테이터에게 ACL을 통해 자신에 대한 통신 명세(communication specification)를 제공한 후, 자신의 본래 업무를 수행한다. 에이전트는 업무 수행 도중 필요한 정보나 서비스를 퍼실리테이터에게 요청하고 퍼실리테이터는 다른 에이전트들에게 업무를 처리하도록 하여 결과를 돌려준다. 따라서 연방 시스템에서는 에이전트가 다른

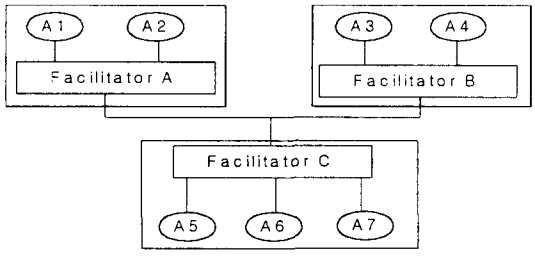


Fig.1 Federated system

에이전트들에 관한 정확한 정보없이도 퍼실리테이터를 중심으로 한 시스템을 구성할 수 있다[2].

Fig 1에 에이전트 연방 시스템의 구성도가 나타나 있다.

2.3 에이전트 통신 언어(ACL)

ACL은 에이전트간의 통신에 쓰이는 언어이다. 본 연구에서는 KQML과 KIF를 ACL로 사용한다.

에이전트간의 정보 교환 및 공유를 위해서는 공통된 언어를 지식 표현에 사용하거나 적어도 서로 번역 가능한 언어를 사용하여야 하며 서로간의 통신이 가능하여야 한다. 지식 표현을 위한 공통 언어로 KIF이 개발되어 있고, KB system 간 혹은 KB system과 다른 프로그램간의 실시간 지식교환을 위해 KQML이 사용되고 있다[3].

2.3.1 Knowledge Query And Manipulation Language(KQML)

KQML은 에이전트 간의 지식 교환을 위한 메시지의 형식이자 상호간의 통신 규약이다. KQML은 지식을 표현하는 것이 아니라 표현된 지식을 통신으로 전달하기 위한 형식이다. KQML은 실제 실어나르는 지식 즉, 내용에 독립적이며 내용의 표현에는 여러가지 언어가 쓰일 수 있다. KQML은 다만 이들 내용에 대한 태도(질문, 요구, 명령, 단언)를 나타낸다. 태도를 나타내는 KQML의 기본 요소를 performative라고 부르는데 performative는 다른 여러 인자와 함께 완전한 KQML 문장을 이루며 또한 상호간의 통신 규약을 나타내는 역할을 한다.[4, 5] 따라서 에이전트는 performative의 통신 규약에 따라 대화를 수행해야 하며 performative가 나타내는 태도에 따라 전달된 지식을 해석하게 된다.

KQML의 간단한 예로서 다음의 KQML 메시지를 살펴보자.

```
(ask-one :sender agent1 :receiver facilitator
         :reply-with id1   :langauge KIF
         :ontology Paticular_Dim
         :content (length h30 ?l))
```

여기서 *ask-one*이 performative로서 *ask-one*은 :content에 대한 태도가 질문임을 나타냄과 동시에 상호간의 통신 규약에 따라 다음과 같은 형태의 KQML 메시지가 오기를 기다린다.

```
(tell    :sender facilitator :receiver agent1
         :in-reply-to id1  :langauge KIF
         :ontology Paticular_Dim
         :content (length h30 233))
```

tell performative는 :content의 사실이 참이라는 단언을 나타낸다. 따라서 이 대화의 내용은 h30번 배의 길이가 얼마냐고 물은 것에 대해 233m 라고 대답하는 것이다.

:sender는 메시지의 전송자, :receiver는 메시지의 수신자를 나타낸다. *ask-one*의 :reply-with와 *tell*의 :in-reply-to는 서로 짝을 이루어 질문에 대응하는 정확한 *tell* 메시지를 찾을 수 있게 한다. :language는 :content를 표현하는데 쓰인 언어가 KIF임을 밝힌다. 마지막으로 :ontology는 content에 쓰인 어휘의 정의를 담고 있는 것으로 여러 에이전트 간에 쓰이는 어휘(예로써, 앞 예제의 'length')를 공유하기 위한 것이다. 즉, 어떤 대상 영역에 대한 어휘를 KIF과 같은 언어를 써서 공식적으로 정의하고 이 정의에 따라 메시지를 해석함으로써 에이전트들이 서로 통신 할 수 있도록 하는 것이 목적이다.

온톨로지는 본 연구의 범위를 벗어나고, 또한 본 연구에서 시험적으로 구현한 에이전트 시스템의 규모가 작으므로 공식적으로 어휘를 정의하여 사용하지 않는다.

2.3.2 KQML performative의 해석

KQML performative는 상호간의 통신규약을 포함하고 있는데 하나의 KQML 메시지로는 완전한 대화가 이루어지지 못하는 경우가 많다. *tell* performative의 경우는 하나의 메시지로 대화가 완성되지만, *standby* performative로 시작되는 대화는 *ready*와 *next*, *tell* 등의 메시지가 현재의 상태에 맞추어 사용되고 *eos*로 대화를 끝맺는

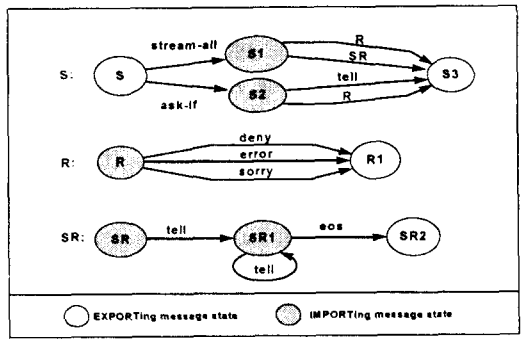


Fig.2 Example of KQML protocol

다. 따라서 KQML performative의 해석과 처리를 위해서는 하나의 대화가 시작되어 끝날 때까지의 performative를 분석하고 상태에 따른 적절한 반응을 제공할 필요가 있다.

Yannis Labrou와 Tim Finin은 이러한 performative의 의미를 형식화하여 나타내었다. Fig 2는 간단한 performative의 프로토콜을 보여준다[6].

Fig.2에서 S, R, SR 등은 state 이름이다. 여기서 *stream-all*을 예로 살펴보면, 에이전트가 상태 S에 있을 때 *stream-all* performative를 보내면 에이전트는 S1의 Importing message state가 되어 상대방의 메시지를 기다리게 되는데 정상적인 경우는 SR로 표시된 화살표의 경로를 따른다. Fig.2의 SR(그림의 하단)에서 알 수 있듯이 대답의 수 만큼 *tell* 메시지를 받은 후 *eos* 메시지를 받고는 하나의 대화가 끝이 난다.

또한 에이전트의 KQML 대화는 비동기적으로 이루어질 수 있다. 즉, 에이전트가 하나의 대화를 시작하면 에이전트는 이 대화가 끝날 때까지 다른 대화를 할 수 없는 것이 아니라 동시에 여러개의 대화를 진행할 수 있다는 것을 말한다.

실제로 KQML은 :reply-with와 :in-reply-to를 이용하여 이 KQML 메시지가 어떤 대화에 참여하고 있는지를 나타낸다. 예로써 아래의 *standby*는 대화를 시작하며 *ready*의 id4와 짝을 이룬다. *ready*는 *next*의 id6과 짝을 이루며 대답의 흐름을 조절하게 되고, *stream-all*은 *tell*, *eos*의 id5와 짝을 이룬다.

```
(standby :reply-with id4 :language KQML
        :content (stream-all :reply-with id5
                :content (father A ?son)))
(ready :reply-with id6 :in-reply-to id4)
(next :in-reply-to id6)
(tell :in-reply-to id5 :content (father A B))
(next :in-reply-to id6)
(eos :in-reply-to id5)
```

따라서 KQML의 해석을 위해서는 :reply-with와 :in-reply-to의 호응관계를 통하여 대화의 상태를 파악하고 비동기적 대화를 가능케 하여야 한다.

2.3.3 Knowledge Interchange Format(KIF)

KIF는 LISP, Prolog언어와 유사하게 지식(knowledge)이나 규칙(rule) 등을 표현할 수 있는 first order predicate calculus의 prefix 타입의 언어이다[7]. KIF는 상이한 지식 표현 언어를 사용하는 시스템 간의 지식 교환을 위한 목

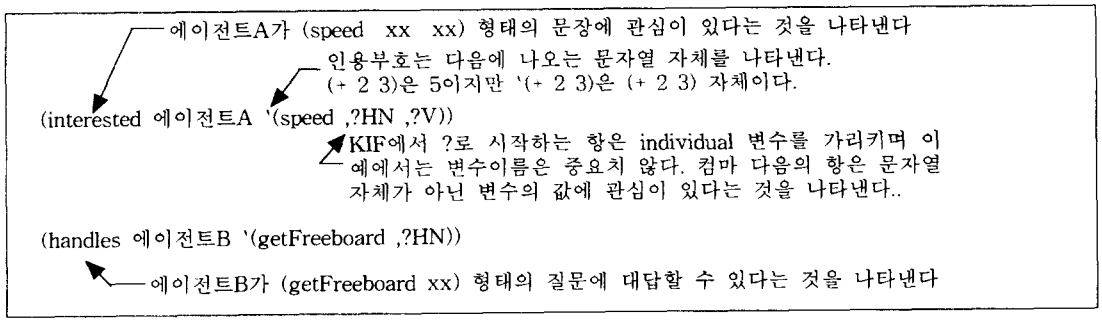


Fig 3. 'interested' and 'handles' performatives

적으로 개발되었으므로 풍부한 표현력을 가지고 있다.

2.4. 퍼실리테이터의 기능

2.4.1 Content-Based Routing(CBR)

CBR 기능은 에이전트가 보낸 ACL의 내용을 보고 그 메시지를 다룰 수 있는 인터페이스를 등록한 에이전트를 찾아서 그 에이전트에게 ACL을 routing 해주는 기능이다.

퍼실리테이터에 Fig.3의 두 개의 KIF 문장이 등록되어 있다고 하자. 본 연구에서는 *handles*와 *interested* performative를 정의해서 쓰는데 *handles*는 에이전트가 할 수 있는 일(능력)을 나타내고 *interested*는 에이전트가 제공받기를 원하는 정보의 형태를 표현한다. 이 두 개의 문장은 각기 에이전트A와 에이전트B에 정보를 제공하거나 요청할 때 사용할 인터페이스를 정의하는 역할을 한다. 어떤 에이전트가 퍼실리테이터에게 다음과 같은 ACL을 보내면

```
(tell :content (speed h30 15))
(ask-one :content (getFreeboard h30))
```

퍼실리테이터는 등록된 인터페이스와 ACL의 content를 비교하여 *tell*은 에이전트A에게, *ask-one*은 에이전트B에게 routing 해준다.

2.4.2 Translation

Translation 기능은 인터페이스를 맞추기 위해 들어온 ACL을 다른 형태로 변환하는 기능으로 이때 퍼실리테이터에 알려진 다른 정보를 이용하여 변환 작업을 수행한다.

퍼실리테이터에 다음과 같은 내용이 알려져 있고

```
(handles 에이전트A
        `(getLength ,?HN 'from ?L))
(length h30 233)
```

또한 퍼실리테이터에 변환에 필요한 지식이 있다고 하자.

```
((= (getSome ?HN)
    (getLength ?HN 'from (length ?HN ?L)))
```

그러면 Fig 4와 같은 ACL이 퍼실리테이터에게 왔을 때 퍼실리테이터는 KIF 번역기의 translation 기능을 이용하여 ACL을 변환한 후 CBR 기능을 이용하여 변환된 ACL을 에이전트A에게 보낸다.

2.4.3 Synthesis/Decomposition

인터페이스가 맞지 않는 또다른 경우는 인터페이스가 여러 지식으로 합성되어 있거나 또는 보내지는 지식이 분해되어야 하는 경우이다.

다음과 같은 여러개의 인터페이스가 등록되어 있을때

```
(interested 에이전트A `(length ,?HN ,?L))
(interested 에이전트B `(breadth ,?HN ,?B))
```

퍼실리테이터가 다음의 ACL을 받았다고 하자.

```
(tell :content (and (length h30 233)
                    (breadth h30 42)))
```

여기서 퍼실리테이터는 and 이하의 문장을 '참'으로 여기므로 결국 (length h30 233), (breadth h30 42)라는 두개의 문장이 참이 되고 각각의 문장을 CBR 기능을 이용하여 에이전트A와 에이전트B에게 중계한다.

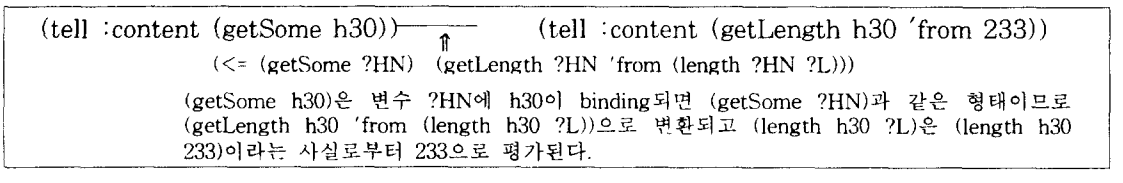


Fig 4. Translation

이상의 ACL에 대한 개념과 퍼실리테이터의 기능에 대해서는 '선박설계 에이전트 시스템 사양서'[8]에서 다루었다. 본 논문에서는 이러한 사양을 토대로 다음에 기술하는 에이전트 기본 아키텍처와 KQML 처리기에 관해 연구했다.

3. 에이전트 기본 아키텍처

에이전트 기반의 시스템을 구축하기 위해서는 이용 가능한 여러 자원의 에이전트화가 필요하다.

이를 위해서는 KQML performative와 content인 KIF를 처리하여야 하며, 이를 체계적이고 효과적으로 지원하기 위해서는 에이전트 기본 아키텍처가 확립되어야 한다.

본 연구에서는 에이전트 간의 통신을 위해서 현재 Stanford 대학의 ABE의 프로젝트에서 개발하고 있는 JATLite[9]의 통신 기능을 최대한 활용한다. JATLite는 에이전트 간의 통신을 지원하는 Java 템플리트 패키지이다.

퍼실리테이터 기능을 구현하기 위해서는 KIF 문장의 매칭(matching)을 통해 CBR 정보를 찾아내고, translation 기능을 지원하는 KIF 번역기가 필요하다. 본 연구에서는 이들 기능과 추론 기능, 함수 및 관계(relation)의 정의 기능을 제공하는 시험형 KIF 번역기를 사용하였다[10].

3.1 관련 연구와의 비교

Stanford 대학교 Logic group의 Singh[11]은 2절에서 설명한 에이전트 시스템을 구현하기 위한 API를 Common LISP으로 작성하였다. 본 연구와 같은 개념을 기반으로 삼기 때문에 기능적으로 많이 유사하지만 LISP으로 구현함으로써 JAVA에 비해 플랫폼 독립성이 떨어진다. IBM의 Agent Building Environment[12]는 주로 stand-alone 형태의 에이전트를 개발하기 위한 환경으로 KIF으로 쓰여진 rule 기반 추론을 통해 일을 수행한다. 외부와의 통신을 위해서는 HTTP 등의 프로토콜을 지원하는 어댑터를 제공하지만 KQML을 지원하지는 않는다. 본 연구에 비해 추론 기능이 강조되고 있으며 다중 에이전트 간의 KQML 메시지 교환을 위한 통신 구조를 포함하지 않는다는 점에서 다르다.

국내에서는 백순철[13]에 의해 다중 에이전트 시스템의 구조가 연구된 바 있다. 여기서는 KQML과 KIF 대신 독자적인 언어가 사용되었다. 또한 WWW상에서의 에이전트 간의 통신구조에 관한 연구[14]가 있다.

3.2 기본 아키텍처

본 연구에서는 Fig.5와 같이 에이전트의 기본 아키텍처를 설계하였다. 에이전트는 JATLite의 틀에서 동작하며, ACL은 JATLite의 통신 인터페이스를 통해 KQML 처리기에 전달되고 다시 KIF 번역기에게 content가 전달된다. 이때 KIF 번역기에서는 *tell*의 content를 자신의 KB에 저장하고 *handles, interested*는 특별히 관리한다. 또한 *ask-one*에 대해서는 대답을 찾아주거나 만약 에이전트가 퍼실리테이터의 역할도 겸한다면 다른 에이전트에게 메시지를 보내기 위한 routing 정보를 찾아준다. KQML 처리기는 이와 관련된 메시지

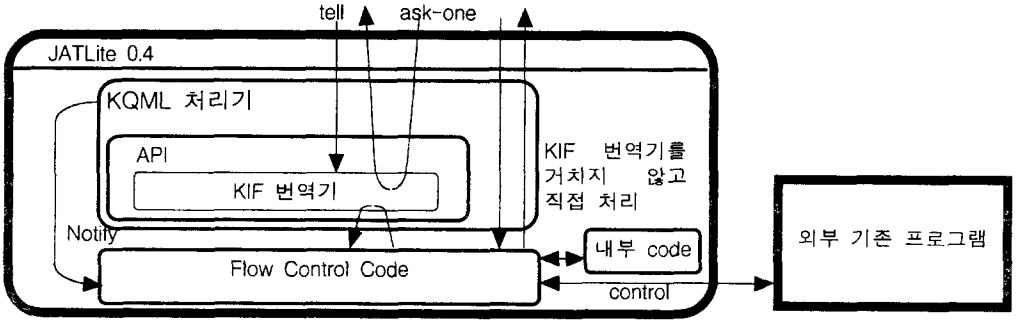


Fig 5. Basic Architecture of Agent System

관리의 일을 맡아서 하며 KIF 번역기의 결과를 바탕으로 메시지를 작성하여 발송한다.

지금까지 설명한 일련의 동작은 독립된 쓰레드(thread)가 자동적으로 수행하며 또 다른 쓰레드에 의해 Flow Control Code를 실행한다. Flow Control Code는 실제 목표하는 일을 수행하는 code이며 필요한 정보를 KIF 번역기의 API를 통해 KB에서 얻어서 내부 실행 code와 외부 기존 프로그램을 관리한다. 이 때 외부의 에이전트로부터 새로운 정보가 도착하면 KB의 내용이 변경될 것이고 따라서 KQML 메시지 처리기는 Notify 메시지를 통해 Flow control code에게 그 사실을 알린다. 또한 Flow Control Code는 KIF 번역기를 거칠 필요가 없는 *ask-one* performative를 직접 KQML 메시지 처리기에 전달할 수 있고 또한 KB에 저장될 필요가 없는 외부 정보도 다룰 수 있다.

에이전트 기본 아키텍처는 최대한 독립적인 모듈들과 Flow control code로 구성되며, 이들을 실행시키는 독립된 두개의 쓰레드에 의해 다양한 일을 수행할 수 있어 일관된 아키텍처를 제공한다. 따라서 최소한의 코드 작성으로 에이전트를 개발할 수 있어 생산성을 높일 수 있다.

3.3 KQML 처리기

3.3.1 KQML 처리기의 설계방향

KQML 처리기는 에이전트가 받은 KQML performative를 처리하는 독립적인 모듈로 크게 다음과 같은 두가지 사항을 고려할 필요가 있다.

첫째, 쉽게 확장할 수 있는 구조로 설계하여야 한다. KQML의 기본구조는 앞에서 보았듯이 performative와 performative의 파라메터들로 이루어져 있다. KQML 사양서에는 reserved performative와 파라메터의 의미에 대해 기술되어 있다. 하지만 reserved performative는 모든 에이전트의 요구를 충분히 수용하기에는 부족한 부분들이 있다. 따라서 KQML performative는 일정한 문법에 맞추어 확장하는 것이 허용되고 있다.

또 KQML은 ACL의 외부언어이기 때문에 내부언어에 대해서는 정해 놓은 바가 없다. 본 연구에서는 내부언어로서 KIF을 사용하나 KIF 이외의 다른 언어를 내부언어로 쓰는 것도 가능하며 심지어 다중의 내부언어를 지원하는 에이전트가 적합한 경우도 있다. 이와같이 KQML 처리기는 performative의 확장, 다양한 내부언어에 쉽게 대응하는 구조로 만드는 것이 중요하다.

둘째, 외부 세계와의 연결 및 KQML 처리기의 효과적인 사용을 위한 인터페이스를 설계하여야 한다. 에이전트 기본 아키텍처에서 KQML 처리기는 최대한 독립적인 모듈로 설계되었으며 flow control code에서 쉽게 사용할 수 있는 인터페이스를 제공한다. 그럼으로써 메시지 처리기의 재사용성을 높이고 flow control code의 작성에만 신경쓸 수 있게 해준다.

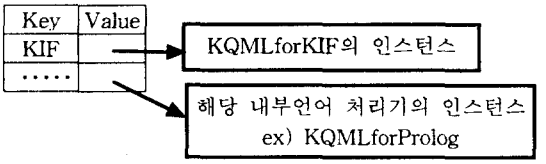
3.3.2 구성

KQML 처리기는 여러개의 클래스로 이루어졌으나 여기서는 다음과 같이 중요한 세 개의 클래스를 설명한다.

1) KQMLHandler

: KQML 처리기를 대표하는 클래스이다. JATLite를 통해 받은 KQML 메시지의 :language 파라메터를 보고 해당 언어의 처리기로 넘겨주는 역할과 외부 세계에 대한 KQML 처리기의 인터페이스 역할을 담당한다.

내부언어 이름을 key로 내부 언어 처리기의 인스턴스를 value로 가지고 있는 hashtable이 있어서 내부 언어 처리기와 연결된다.

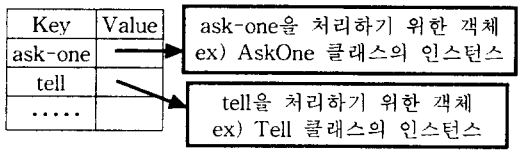


2) KQMLforKIF

: 내부언어가 KIF로 지정된 KQML을 처리하는 클래스이다. 외부언어와 내부언어로 구성되는 KQML 메시지의 구조와 내부언어 해석기의 확장 및 해당 내부언어용 performative의 확장을 고려

하여 만들어진 클래스이다. 이 클래스에서는 KQML performative 하나당 하나의 PerfHandler 구현 클래스를 등록받으며 KQMLHandler 클래스로부터 받은 KQML 메시지의 performative에 따라 해당하는 PerfHandler를 실행한다.

performative 이름을 key로 PerfHandler를 구현한 객체를 value로 가지고 있는 hashtable이 있다.



3) PerfHandler

: 이것은 performative를 실제로 처리하는 코드를 위한 것으로 클래스가 아니라 JAVA 언어의 인터페이스이다. 여기에는 간단히 receive (KQMLmessage msg)라는 인터페이스가 정의되어 있다. 이 인터페이스는 KQMLforKIF 클래스에 의해 호출된다.

3.3.3 KQML 메시지 처리 경로

Fig 6은 에이전트 기본 아키텍쳐에서의 메시지 처리 경로를 보여주고 있다. 에이전트 기본 아키텍쳐에서 메시지는 통신 기능을 담당하는 CommInterface(본 연구에서는 JATLite가 사용되었다.)와 KQML 처리기 그리고 flow control code 순으로 전달되거나 그 역순으로 보내진다. Fig.6의 검은 상자는 메시지 큐(queue)를 나타낸다. 에이전트 기본 아키텍쳐에는 두 개의 메시지 큐가 있는데 하나는 수신되는 메시지에 대한 큐로서 JATLite에 의해 관리되고 다른 하나(notifyQueue)는 KQML 처리기가 관리하는 것으로 flow control code에게 메시지를 전달하는 역할을 한다.

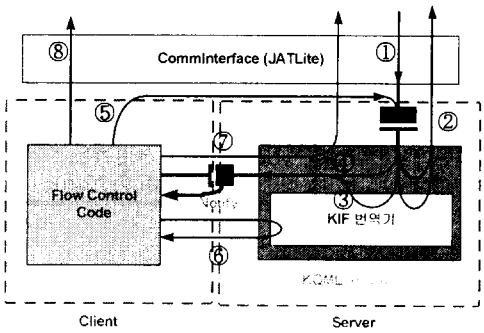


Fig.6 Processing route of KQML handler

notifyQueue는 KQML 처리기가 새로운 메시지를 받았을 때 메시지를 저장해 놓는 장소이자 flow control code에게 새로운 메시지가 왔음을 알려주는 객체이다.

에이전트 기본 아키텍쳐는 다양한 메시지 경로를 지원하는데 이러한 경로는 사용자의 필요에 의해 선택될 수 있다.

기본적으로, 들어오는 메시지는 ①-③을 거쳐 notifyQueue에 저장되거나, ①-②를 따라서 대답이 보내진다. KQML 처리기를 특별히 확장하여 ④의 경로로 메시지를 notifyQueue에 전달할 수도 있다. 보내는 메시지는 ⑦의 경로를 통해 KQML 처리기가 외부로 보내거나 ⑤와 같이 자기 자신에게 전달할 수 있다. 또한 ⑧을 통해 직접 외부로 메시지를 보낼 수도 있다. 마지막으로 ⑥은 flow control code에서 내부의 KQML 처리기를 직접 이용하는 경우를 나타낸다.

이 중 그림에서 ①, ⑤의 메시지 경로는 :sender가 다른 에이전트인가 자신인가에 따른 차이일 뿐 기본적으로는 Fig.7과 같은 처리 경로를 따라 KIF 번역기에게 :content가 전달된다.

JATLite의 통신기능을 이용해 외부로부터 들어오는 메시지는 JATLite의 메시지 큐에 저장되며 Fig.7에서 보듯이 RouterActionClient클래스의 Act 메쏘드를 처음으로 여러 메쏘드를 차례로 거친다.

3.3.4 performative 처리 알고리즘

다음에서 퍼실리테이터가 ask-one 메시지를 받았을 때 KIF 번역기를 통해 적당한 대답을 보내거나, CBR 기능을 이용해 다른 에이전트에게 routing 하는 알고리즘을 예로써 소개한다.

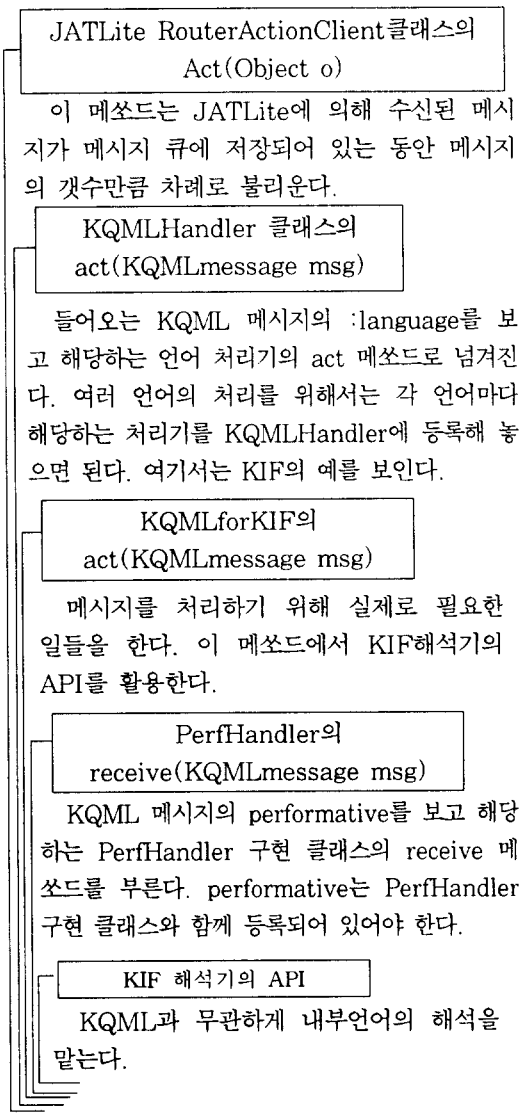


Fig 7. Route of received message

이들은 메시지 처리 경로(Fig.7)의 Perf-Handler 단계에 해당하는데 여기서는 *ask-one* performative의 처리에 관한 부분을 소개한다.

*ask-one*의 경우 다른 performative와 달리 메시지 관리에 크게 신경을 써야하는데 수신된 *ask-one*를 ReplyTo 메시지 테이블에 기록하여 관리한다. 이를 통해 퍼실리테이터는 현재 질문을 받았으나 아직 대답을 보내지 못한 것, 다른 에이전트에 routing 해주었으므로 결과를 받아 원질문자에게 보내주어야 하는 것들을 관리할 수 있다.

1) AskOne Handler의 receive 알고리즘

① ReplyTo 테이블에 ask-one 메시지를 기록한다 (ask-one 관리).

② KIF 번역기의 API function인 askOne을 부른다.

ex) askOne("getPrincipalDim h30", "bc")

"bc"는 theory 이름으로 KIF 번역기의 KB이름이다.

③ askOne은 return 값으로 다음 세가지 경우가 있다.

i. 상응하는 handles 메시지를 찾을 경우 (CBR 기능지원)

ex) "handles〉 Agt1 (getPrincipalDim h30 'from 234 42 19,5 13.6 0.8145 ....)"

ii. 대답을 return 하는 경우

ex) askOne("length h30 ?x", "bc")에 대해 "reply〉 (233)"로 return

iii. 해당하는 결과값이 없는 경우

ex) askOne("length h40 ?x", "bc")에 대해 "NIL〉 (length h40 ?x)"로 return

④ return된 결과의 헤더를 보고 각각에 대해 적절한 performative를 보낸다.

case i 일때

다음과 같이 sendAskOne 메쏘드를 부른다. 여기서 두번째 인자는 content이고 마지막 인자는 ReplyTo 테이블의 entry 번호로, 아래와 같이 ask-one을 Agt1에게 보내 얻어지는 대답을 지정한 entry 번호의 대답으로 routing 하라는 뜻이다. 이는 대답을 원질문자에게 보내기 위해서이다.

ex) sendAskOne("Agt1", "(getPrincipalDim h30 'from 234 42 19.5 13.6 0.8145 ....)", 3);

case ii 일때

다음과 같이 sendReply 메쏘드를 부른다. 여기서 "(length h30 233)"는 content이고

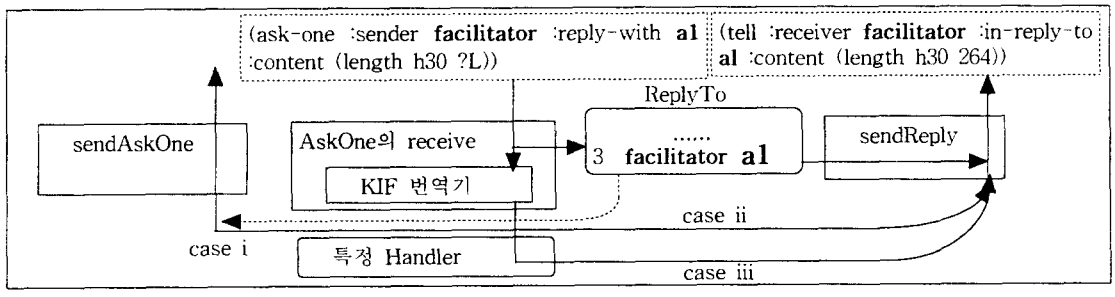


Fig 8. Operation Flow of KQML Message Handler

두번째 인자는 ReplyTo 테이블의 entry 번호로, 이 entry의 정보를 보고 sendReply 함수가 *tell* 메시지를 만들어 보낸다. 이때 in-reply-to를 *ask-one* 메시지의 reply-with 값으로 만들어 보낸다.

ex) sendReply("(length h30 233)", 3)

case iii 일때

"length h40 ?x"에 대한 Handler가 있으면 실행시키고 그 실행결과를 sendReply 메쏘드를 통해서 보낸다.

2) sendReply 메쏘드의 알고리즘

① 먼저 tell performative를 만들어 보낸다. 이때 ReplyTo 테이블의 entry 번호로부터 적절한 정보를 얻는다.

ex) sendMessage("tell :sender Agt2 :receiver facilitator :in-reply-to a1 :content (length h30 233)")

여기서는 ReplyTo 테이블의 정보로부터 이 메시지의 receiver가 facilitator이고 :in-reply-to 값이 a1임을 알게 된다.

② ReplyTo 테이블에서 방금 전송한 메시지에 해당하는 entry를 삭제한다.

3) sendAskOne 메쏘드의 알고리즘

① 보낼 *ask-one* 메시지를 생성하고 replyWith 테이블에 기록한다. 이때 메시지의 :reply-with 값은 유일한 값이 되도록 해야 한다. (ask-one 관리)

② 만약 routing을 위한 ReplyTo 테이블의 entry 번호가 지정되어 있으면 replyWith 메시지 테이블에 이 entry 번호를 기록한다. 이렇게 함으로써 지금 보내는 *ask-one* 메시지의 대답을 원질문자에게 보내줄 수 있다.

③ sendMessage를 통해 *ask-one* KQML을 발송한다.

Fig 8에 KQML 처리기의 개략적인 메시지 처리 전략을 나타내었다. 에이전트가 Fig 8의 *ask-one* 메시지를 받으면 Fig 7의 경로를 따라 AskOne Handler의 receive 메쏘드가 실행되는데 알고리즘의 case ii에 의해 sendReply 메쏘드가 실행된다. sendReply 메쏘드는 ReplyTo 테이블로부터 :receiver와 :in-reply-to를 각각 facilitator와 a1으로 설정하여 대답을 보낸다.

3.3.5 KQML 처리기 인터페이스

에이전트는 기본적으로 peer-to-peer 구조를 가지고 있다. 앞서 설명한 메시지 처리 경로와 KQML 처리 알고리즘은 서버(server)로 동작하는 에이전트의 역할을 보여준다. 반면, flow control code는 에이전트의 클라이언트 역할을 수행한다.

서버로서의 에이전트는 자신의 내부 또는 외부의 클라이언트가 질문을 했을 때 필요한 작업을 수행하며 클라이언트로서의 에이전트는 flow control code에 따라 실제 목적하는 일을 수행한다. 또한 이 과정에서 다른 에이전트에게 서비스를 요청한다. 본 연구의 KQML 처리기는 이런 작업에 필요한 메시지 전송 인터페이스를 제공한다.

String sendMessage(String msg, boolean synch)
String postMessage(String msg, boolean synch)

첫 번째 인자에는 전송할 KQML 메시지가 주어진다. 이들 메쏘드는 주어진 메시지의 수신자를 보고 외부로 나가는 메시지인지 자신에게 보내지는 메시지인지를 구분한다. 외부로 나가는 메시지에 대해서는 두 메쏘드 간의 차이는 없다. 그러나 자신에게 보내지는 메시지에 대해서는 postMessage가 바로 처리하는데 반해, sendMessage는 메시지 큐에 저장하여 메시지가 큐에 들어오는 순서대로 처리한다. 두 번째 인자인 synch값은 메시지가 처리되어 return 값이 나올때까지 flow control code가 기다릴지 아니면 다른 일을 수행할 지를 결정한다. synch값이 false일 경우 메시지가 비동기적으로 처리되므로 return 값이 큐에 저장되고 notify 신호가 flow control code에 전달된다.

4. 구현예

본 연구에서의 아키텍쳐를 이용하여 Fig 9와 같은 시험적인 선박 기본계획에이전트 시스템을 구현하였다.

기본계획에이전트가 퍼실리테이터 역할을 하며 기본계획에이전트 아래의 에이전트들은 각기 간단한 기능을 수행하는 서브에이전트들이다.

이 시스템은 다음과 같이 작동한다.

① 서비스의 인터페이스 등록

기본계획에이전트는 각 서브에이전트로부터 다음과 같은 handles KQML 메시지를 등록받는다. 여기서는 중량추정서브에이전트와 건현서브에이전트의 메시지가 나타나 있다.

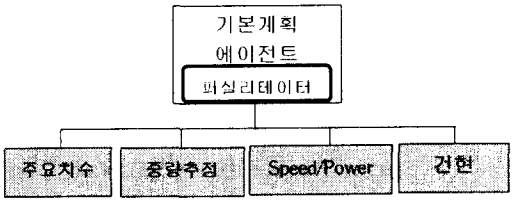


Fig 9. Diagram of implemented system

```
ex) (handles :receiver projagt :content
       (handles weightagt '(getWeights ,?HN
            'from ,?Wh ,?Wm ..... ,?NMCR)))

    (handles :receiver projagt :content
     (handles freeboardagt '(getFreeboard ,?HN
          'from ,?L 1,?D ,?B ,?Cb ..... )))
```

② 선주요구사항 입력

GUI를 통해 입력되는 정보를 KIF 형태로 바꾸어 KB에 저장한다.

```
ex) (hullnumber h30) (shipType h30 tanker)
    (DWT h30 100000) (speed h30 15)
    (draft h30 14.3) (CC h30 110000)
```

* Fig 10, Fig 11에서는 중량추정서브에이전트와 기본계획에이전트가 실행된 모습을 보이고 있는데 기본계획에이전트가 GUI를 통해 선주 요구 조건을 입력받고 있다.

③ 모선 데이터

모선DB로부터 모선주요치수를 가져와서 KB에 저장한다.

```
ex) (lengthb h30 234) (breadthb h30 42)
    (depthb h30 19.5) (draftb h30 13.6)
    (DWTb h30 94500) (speedb h30 15)
    (Cappb h30 1.02) (Cbar h30 0.615)
    (Cbb h30 0.8145) (DWT/dspb h30 0.887) ....
```

④ 질문

기본계획에이전트는 아래의 KQML 메시지를 sendMessage(msg, SYNC) 인터페이스를 사용하여 자기자신에게 보낸다.

```
(ask-one :sender projagt :receiver projagt
  :reply-with projagt3
  :theory theoryh30 :content (getweights h30))
```

⑤ Translation

앞에서 보내진 메시지는 기본계획에이전트의 KQML 처리기에 의해 처리되는데 KQML 처리기의 메시지 처리 경로를 따라 KIF 번역기에서

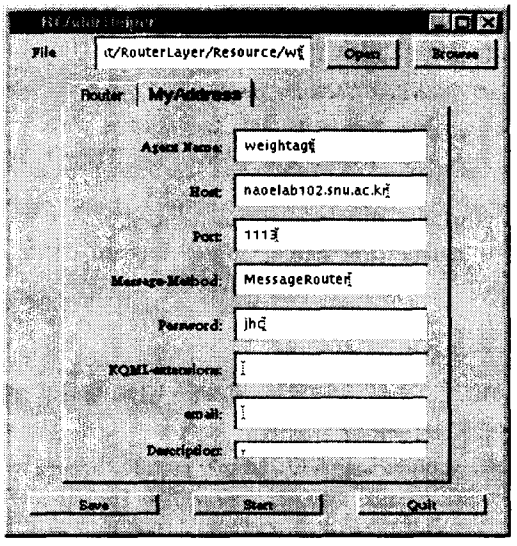


Fig 10. Execution of weight agent

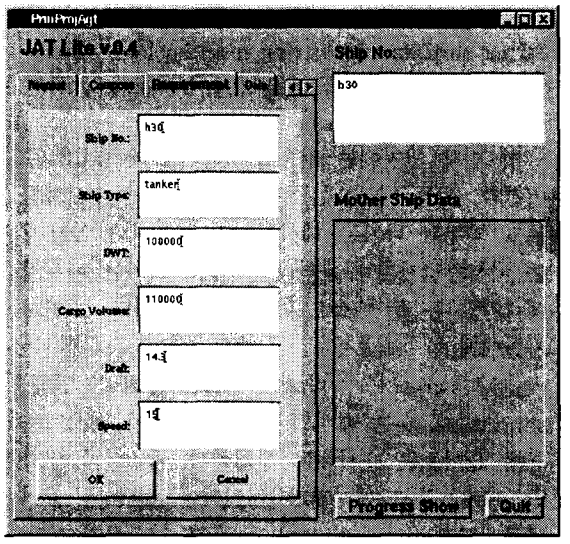


Fig 11. Input panel of requirement

다음과 같이 return 된다.

```
handles) weightagt (getweights h30 'from
                        14039 1681 1380 ..... )
```

이것은 미리 KIF 번역기에 들어있던 translation rule과 과정 ①에서 얻어진 handles 정보를 이용하여 얻어지는데 다음에 translation rule이 나타나 있다.

```
〈Translation rule〉
((= (getWeights ?HN)
    (getWeights ?HN 'from (wsb ?HN ?ws)
    (wob ?HN ?wo) ..... )
```

⑥ Content-Based Routing

과정 ⑤에서 얻어진 정보를 바탕으로 KQML 처리기는 다음과 같은 메시지를 중량추정에이전트에게 보내고

```
(ask-one :sender projagt :receiver weightagt
  :reply-with projagt4 :theory theoryh30
  :content (getweights h30
                'from 14039 1681 1380 ..... ))
```

그 대답을 얻는다.

```
(tell :sender weightagt :receiver projagt
  :in-reply-to projagt4
  :content (and (Ws h30 15141.6)
                (Wo h30 1824.1)  ..... ))
```

⑦ 원래의 질문자에게 Routing

과정 ⑥에서 얻어진 대답은 KQML 처리기의 askOne receiver handler의 알고리즘에 따라 원래의 질문자에게 보내진다.

```
(tell :sender projagt :receiver projagt
   :in-reply-to projagt3 :reply-with projagt5
   :content (and (Ws h30  15141.6) ..... ))
```

따라서 이 메시지는 KQMLHandler 클래스의 notifyQueue에 저장되어 flow control code가 대답이 도착했음을 알 수 있다.

⑧ 평가

과정 ⑦의 결과로 얻어진 Ws, Wo, Wm을 이용하여 주요치수로부터 추정한 경하중량을 계산하고, 또한 주요치수에 의한 배수량과 요구조건인 재화중량의 차이로부터 경하중량을 계산한 후 서로 일치하지 않으면 주요치수를 조정한다.

Fig 12에 두 가지로 계산된 경하중량이 차이가 나서 주요치수를 조정해야하는 단계의 기본계획에이전트가 나타나 있다.

이 후 Speed/Power서브에이전트와 건현서브에이전트를 통해 보다 정확한 주요치수를 선정한

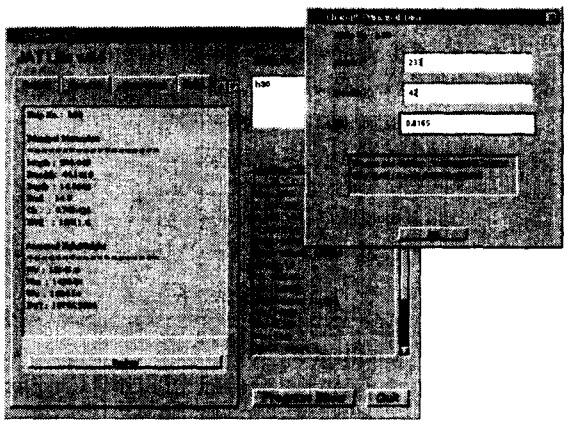


Fig 12. Input panel of principal dimension

다.

Fig 13에 실행예가 나타나 있다.

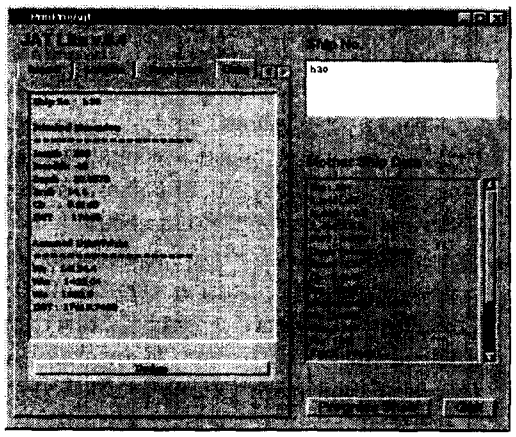


Fig 13. Result of estimated pricipal dimension

5. 결론

위의 예에서 보듯이 에이전트들은 퍼실리테이터에 자신의 인터페이스를 등록하고 이를 통하여 시스템에 서비스를 제공하거나 다른 에이전트의 서비스를 이용한다. 이때 상호간의 인터페이스의 불일치를 퍼실리테이터의 translation, synthesis/decomposition 기능을 통해 해소하고 CBR 기능을 통해 필요한 메시지를 분배함으로써 기존 시스템에 비해 유연한 시스템을 구성할 수 있다고 생각한다.

이에 따라 본 연구에서는 분산된 시스템의 통합을 위해 에이전트라는 개념을 도입하였고 에이전트 기본 아키텍쳐를 설계하였다. 또한 에이전트에게 관심있는 정보를 등록하는 메시지, 처리할 수 있는 업무를 표시하는 메시지, 정보를 알려주는 메시지, 정보에 대해 질문하는 메시지를 처리하는 시험적인 KQML 처리기를 구현하여서 선박 기본계획에이전트 시스템에 적용하였다.

본 연구에서 구현한 에이전트 시스템은 아직 시험적인 단계로 다음과 같은 사항이 향후 보완되어야 한다. 먼저, 본 연구에서 사용된 KIF 번역기와 KQML 처리기를 각 언어의 사양에 충실한 모듈로 개발하여서 훨씬 더 복잡한 설계 문제에 적용할 수 있도록 개선해야 할 것이며, 향후 시스템 컴포넌트간의 설계 충돌의 해결과 효과적인 지식의 공유를 위한 온톨로지의 도입과 개발 등이 연구되어야 할 것이다.

후 기

본 연구는 한국과학재단 특정기초연구과제 (과제번호 96-0200-01-01-3)로 수행되는 "CALS 지향 동시공학적 선박설계 에이전트 시스템 개발" 결과의 일부분임을 밝혀둔다.

참 고 문 헌

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

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

[3] R. Patil, R. Files, P. Patel-Schneider, D. McKay, T. Finin, T. Gruber, and R. Neches. The DARPA Knowledge Sharing Effort: Progress Report. In Principals of Knowledge Representation and

Reasonning: Proceedings of the Third International Conference, November 1992.

[4] T. Finin, Y. Labrou, and J. Mayfield. KQML as an agent communication language. In Jeffery M. Bradshaw, editor, Software Agents. MIT Press, 1995.

[5] Y. Labrou and T. Finin. 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.

[6] Y. Labrou and T. Finin. A semantics approach for KQML - a general purpose communication language for software agents. In Third International Conference on Information and Knowledge Management.

[7] M. Genesereth, R. Fikes, et. al. Knowledge interchange format, version 3.0 reference manual. Technical report, Computer Science Department, Stanford University, 1992

[8] 이규열, 연윤석, 김수영, 윤덕영. SIF를 토대로 한 선박설계 에이전트 시스템 사양개발 및 그 구현 예에 관한 연구. 조선학회 1997년도 춘계 학술대회 논문집, 1997. 4

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

[10] 최제민, 김봉재, 연윤석, 양영순. 에이전트 기반의 선박설계 시스템 구축을 위한 시험형 KIF 번역기 개발. 조선학회 1997년도 추계학술대회 논문집, 1997. 11

[11] Narinder Singh. A Common Lisp API and Facilitator for ABSI. Technical report Logic- 93-4, Logic Group, Computer Science Department, Stanford University, 1994. 3

[12] IBM Intelligent Agent Center of Competence, IBM Agent Building Environment Developer's Toolkit Level6 Overview, http::/www.alphaworks.ibm.com/tech/abe. 1997

[13] 백순철, 최중민, 장명욱, 박상규, 임영환. 이형 분산 환경에서 에이전트간원 이형성을 극복하기 위한 멀티에이전트 기반구조. 정보과학회논문지(C) 제 2권 제 1호, 1996. 3

[14] 안상준, 이수홍. WWW을 이용한 에이전트 기반 공동 설계 환경 개발. 한국 CAD/CAM 학회 논문집 제 3권 제 1호, 1998. 3