한국 CAD/CAM 학회 논문집
제4권 제4호 1999년 12월 pp. 360-370

학술논문

ACL과 CORBA를 이용한 선박 초기설계 에이전트 시스템에 관한 연구

김동현*, 이규열**, 이상욱*

A Study on Ship Initial Design Agent System Based on ACL and CORBA

Dong-Hyun Kim*, Kyu-yeul Lee** and Sang-Uk Lee*

ABSTRACT

The paper proposed a basic architecture of an agent system to support exchange and sharing of design informations by means of ACL(Agent Communication Language) which can represent design informations and knowledges. Based on the architecture of the agent system a ship initial design agent system was implemented in order to show the effectiveness of the agent-based system. The basic architecture of the agent consists of an ACL handlerand CORBA(Common Object Request Broker Architecture) objects for the exchange of ACL messages in the heterogeneous and distributed environment. The ACL handler can process expressions of knowledge and manage communication messages among the agents. The paper mainly focuses on the implementation of the ACL handler. The ACL handler consists of a KQML(Knowledge Query and Manipulation Language) handler that manages KQML messages, a conversation module, and a content handler that handles message contents. The conversation module implements conversation policies and checks all messages if they are allowable and meaningful messages based on the conversation policies. The implemented agent-based system was applied to the ship initial design to show the handling procedure of the agent system.

Key words : Agent, CORBA, ACL, KQML

1. 서 론

본 연구에서는 지식표현 능력을 가지는 에이전트 통신언어(Agent Communication Language; ACL)를 이용하여 분산된 시스템에서의 정보의 전달과 공유를 지원할 수 있는 선박 초기 설계 에이전트 시스템을 구현하였다.

에이전트 기본 구조는 ACL처리기와 메시지를 전달하는 CORBA객체로구성되어 있으며, 본 논문에서는 ACL 처리기를 중점적으로 구현하였다. ACL 처리기는 외부언어인 KQML을 처리하는 KQML처리기, Conversation Module, 그리고 내부언어 처리기로 구성된다.

*학생회원, 서울대학교 조선해양공학과
**정회원, 서울대학교 조선해양공학과 및 해양시스템공학연구소

2. 에이전트 시스템

2.1 광의의 에이전트 시스템

현재 수많은 프로그램들이 에이전트라는 이름을 가지고 개발되고, 발표되고 있다. 이러한 에이전트 시스템으로는 인터넷을 돌아다니며 사용자가 원하는 자료를 모아주는 모빌 에이전트와 사용자가 원하는 작업을 자동으로 감지하여 수행하는 것 등이 있다. 이렇게 사용자의 요구 조건들을 입력받아, 현재 상황을 감지하여 일정한 조건이 맞추어 지면 사용자를 대신하여 요구받은 일을 수행하는 것을 에이전트라 부르고 있다(Fig. 1).

예를 들어보면 사용자가 저녁 12시 이후 한시간 이상 컴퓨터를 사용하지 않을 경우, 하드 디스크를 대신 정리해 주는 프로그램도 일종의 혼자 일을 수행하는 Stand-alone 타입의 에이전트 프로그램으로

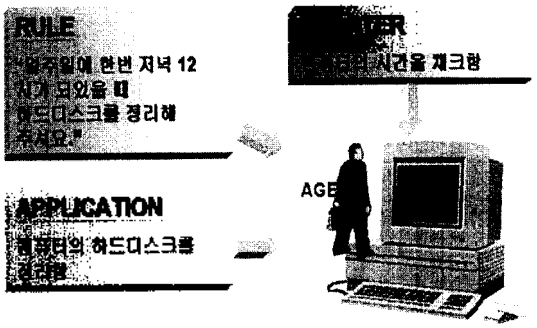

Fig. 1. Example of stand-alone type agent.

볼 수 있을 것이다.

2.2 본 연구에서의 에이전트 시스템의 개념

본 연구에 적용한 에이전트 시스템의 개념은 위에서 본 광의의 에이전트에 대한 개념이 아니라 인텔리전트 에이전트의 개념과 Agent-based software Engineering의 개념을 적용하였다. 인텔리전트 에이전트란 통신을 통하여 정보를 보내고 받을 수 있고, 복수의 objective와 goal을 수행할 수 있어야 한다. 그러기 위해서 자신과 다른 에이전트에 대한 plan을 만들어 낼 수 있고, 받은 정보를 가지고 추론 등을 통하여 자동으로 자신이 맡은 역할을 수행하며, 또한 자신의 명확한 belief model을 유지하며, 상태에 따라 동적으로 바뀔 수 있는 능력을 가지고 있어야 한다.

Agent-based software Engineering의 관점에서 보면 에이전트 시스템이란 다양한 분야, 다양한 정보와 서비스를 제공하는 여러 프로그램들 사이에 나타나는 정보 및 서비스의 교환을 해결하기 위한 방법이라고 볼 수 있을 것이다. 이런 이질적인 환경 하에 쉽게 서로 협동적으로 작동하는 프로그램을 구현하기 위한 방법으로 기존의 단순한 메시지나 정해진 서비스만을 교환하는 클라이언트/서버의 환경이 아닌 표현력이 풍부한 에이전트 통신 언어(Agent Com-munication Language; ACL)를 사용하여 지식을 교환하는 방법이 최근에 연구되고 있다. ACL은 메시지 통신을 위한 외부언어와 전달하고자 하는 정보/지식을 담는 내부언어로 구성된다. 이러한 개념의 에이전트에 관한 연구는 주로 추론을 지원하는 에이전트의 개발, 인터넷, 또는 분산환경에서의 에이전트의 메시지의 전달, 에이전트 메시지 각각에 관해 이루어져 왔다.

본 연구에서는 이러한 개념을 토대로 선박의 초기설계 과정의 정보를 ACL을 사용하여 교환/공유하는 선박초기설계 에이전트 시스템을 구현하였다. 외부언어로는KQML(Knowledge Query and Manipulation Language)[1,2]을, 내부언어로는 Prolog를 사용하였고, 또한 분산된 환경에서 정보를 교환할 수 있는 기술로서 표준적인 분산 객체 환경인 CORBA (Common Object Request Broker Architecture)[3,4]를 사용하였다.

3. 타 에이전트 시스템 구현 예

현재 ILOG, IBM 등 많은 곳에서 에이전트 시스템을 개발하는 중이다. 그중 IBM은 다양한 에이전트 시스템을 구현하였다. 먼저 광의의 에이전트 시스템의 예로써 들 수 있는 Web Browser Intelligence(WBI)는 사용자의 인터넷 경험들을 받아들여, Web Server가 웹의 내용을 더하거나 바꿀 수 있는 능력을 부여하는 컴포넌트 개념의 개발 환경이다. Aglet은 인터넷을 돌아다니며 사용자가 요구한 작업을 수행, 나중에 사용자가 그 결과를 요구하거나 인터넷에 다시 접속하면 작업의 결과를 돌려주는 모빌 에이전트를 지원하는 JAVA 기반의 개발환경이다. Agent Building Environment(ABE)는 인텔리전트 에이전트 시스템을 작성하는 개발 환경이며, JKQML(Java-based KQML)은 에이전트들이 KQML이란 언어를 통하여 통신하는 것을 지원하는 환경이다.

3.1 IBM Agent Building Environment(ABE)

ABE는 룰(rule)을 기반으로 한 추론을 사용하여 단독으로 작업을 수행하는 인텔리전트 에이전트 개발환경이다[5]. Fig. 2는 ABE의 내부 구조를 나타낸 것이다. 각 부분의 기능을 살펴보면 다음과 같다.

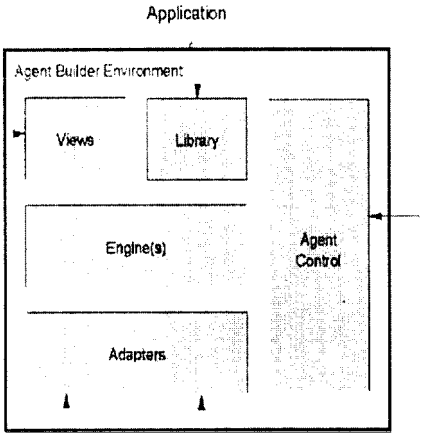

Fig. 2. System Structure of IBMAgent Building Environment(ABE) Toolkit.

Agent Control-사용자가 자신의 에이전트를 만들고 싶을 때, 인터페이스를 제공하는 부분이다. 또한 동적, 정적으로 에이전트 환경을 조정할 수 있는 기능을 제공한다. 또한 에이전트 실행 중에 각 부분들 간의 흐름을 제어한다.

Adapters-다른 프로그램이나 외부와의interaction을 제공한다. 사용자는 자신이 직접 Adapter를 작성할 수도 있으며, 기본적으로는 NNTP(News Group), HTTP, Time, File, Utility Adapter가 기본으로 제공된다.

Library-ABE는 현재 Rule에 기반 한 추론만을 지원한다. 따라서 그 룰과 Fact들을 관리하는 기능을 제공하는 부분이다. 현재는 File로 작성하는 것만을 지원하고 있지만, 앞으로는 관계형 데이터 베이스까지 지원하려고 한다. 또 rule은 KIF이란 언어를 사용하여 작성하도록 되어 있다.

Views-추론에 사용될 Rule을 작성하는 도구를 제공한다.

요약하면 ABE는 각 Adapter를 통하여 받아들인 환경을 룰을 통해 추론하여 작업을 수행하는 인텔리전트 에이전트 시스템 개발환경이다 이러한 룰은 KIF이란 표현력이 강력한 에이전트 내부언어를 통해 작성되고 파일의 형식으로 관리된다. 이를 예로 간단히 살펴보면 다음과 같다(Fig. 3).

ABE가 실행이 되면, 자신의 환경파일을 읽어 사용하는 Adapter인 Time, Mail, Stock Adapter를 초기화한다. 그리고 파일로부터 KIF으로 작성된 자신이 사용할 Rule을 읽어와 Library에 넣어 둔다. 나중에 Time Adapter에서 일정시간마다 발생되는 이벤트에 따라, Stock Adapter를 사용, 인터넷을 통하여 필요한 주식의 값을 읽어온다. 그 값과 룰을 이용하여 추론을 하며 사용자에게 알려야 할 상황이란 결론을 얻었을 경우, Mail Adapter를 사용, 사용자에게 알리게 된다.

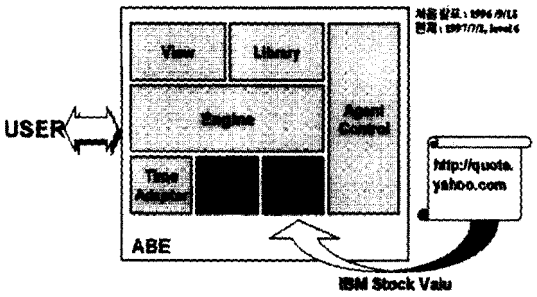

Fig. 3. Example of using IBMAgent Building Environment Toolkit.

3.2 IBM Java-based KQML(JKQML)

JKQML은 일본의 IBM Yamato lab에서 개발하였다. 앞의 ABE가 KIF를 통한 추론을 이용한 인텔리전트 에이전트 시스템을 지원하지만, 단일 에이전트에 대한 툴이다. 이와는 달리, JKQML은 KQML이란 에이전트 외부통신언어를 통하여 다수의 에이전트들이 자신의 지식이나 서비스를 교환하는 시스템을 쉽게 구축하도록 도와주는 개발환경이다[8]. 그 구조는 Fig. 4와 같으며 TCP/IP 기반의 KQML Transfer protocol과 Aglet을 지원하는 Agent Transfer protocol, Object Transfer protocol을 지원한다. 또 사용자가 protocol 처리기를 만들어 확장이 가능하도록 하였다.

각 구성요소를 살펴보면 다음과 같다.

KQML Manager-agent application에 대하여 KQML message를 처리하기 위한 모든 interface를 제공한다. 각 agent application은 KQML manager의 interface를 통하여 모든 component에 접근할 수 있다.

Protocol Manager-KQML message를 실질적으로 전송하는 mechanism을 다룬다. 현재 3가지 transport protocol을 지원한다. KQML Transfer Protocol(KTP), Agent Transfer Protocol(ATP), Object Transfer Protocol(OTP).

Conversation Pool-현재 진행중인 conversation을

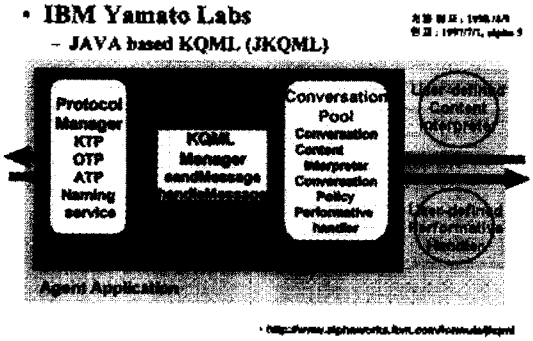

Fig. 4. Structure of IBM Java-based KQML.

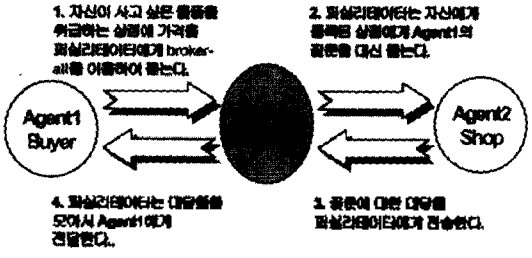

Fig. 5. Example of using Java-based KQML.

다루는 기능을 한다. conversation policies를 사용하여 message sequence를 조정한다.

JKQML을 이용하는 에이전트 시스템의 예를 들어보면(Fig. 5), 어떤 Buyer를 대신하는 에이전트 1이 있고, 어떤 Shop을 대신하는 에이전트 2가 있다고 가정하자. 이들 에이전트들은 facilitator라는 특별한 에이전트에게 KQML을 이용하여 자신을 등록하고 서비스를 요구하면 facilitator는 그것을 중계해주게 된다. 즉 JKQML은 KQML을 이용한 다중의 에이전트의 개발을 지원하는 개발환경이며 자신이 원하는 기능을 위해 확장이 가능한 구조를 가지고 있다.

3.4 CORBALT

CORBA를 이용한 에이전트 시스템 개발도 여러 곳에서 진행 중이며, D. BENECH 등은 CORBA를 사용하여 KQML의 구현(CORBALT)을 Fig. 6과 같이 제안하고 있다[9].

CORBALT는 IDL에서 제공하는 기본적인 기능만을 이용하여 구현하려 하고 있으며, 그 구현은 아직 완전히 이루어진 것이 아니고 현재는 CORBA구현에 사용하는 IDL만이 제시되어 있다.

인터넷이라는 분산환경에 모든 에이전트들이 있으며 이들간의 자유로운 KQML 메시지의 전달이 이루어 지려면 IDL의 표준화가 필요하며, CORBALT는 이러한 움직임중의 하나라고 볼 수 있다.

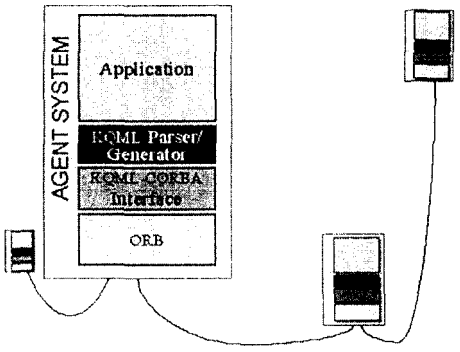

Fig. 6. The structure of CORBALT.

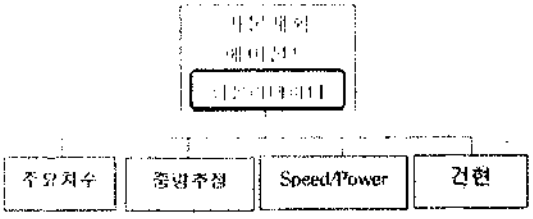

Fig. 7. Structure of Ship initial design agent system -previous version.

4. 선박 초기 설계 에이전트 시스템 구현

4.1 선박 초기설계 에이전트 시스템의 초기버전

Fig. 7은 선박 초기설계 에이전트 시스템 초기버전[8,9]의 구성도 이다. ACL의 운용에 있어서는 내부언어로 KIF[10]을, 외부언어로는 KQML을 사용하였으며, 통신은 JATLite[11]를 이용하여 이루어 졌다. 이 시스템의 문제점은 설계 과정의 흐름(Design Flow Control)의 구현에 있어서 고정된 구조를 가지고 있었으며, JATLite에 의한 통신에서는 메시지가 복잡한 Converting작업을 거쳤기 때문에 효율이 낮았다. 이러한 문제점을 해결하기 위해 선박 초기설계 에이전트 시스템을 다음과 같이 개선하였다.

4.2 개선된 선박 초기설계 에이전트 시스템의 구성

선박 초기설계 에이전트 시스템 초기 버전의개선된 점은 아래와 같다.

1) CORBA를 이용하여 효율적으로 에이전트간 통신을 할 수 있는 분산 객체 구조로 개선하였고, 이때 Visibroker의 Caffeine 기능을 이용하였다.

2) Prolog의 추론기능을 이용하여 Design Flow control을 유연하게 하였다.

3) KQML 메시지를 효율적으로 운영하고, 의미 있는 메시지의 교환을 위하여 대화 모듈(Conversation Module)을 추가하였다.

Fig. 8은 개선된 선박 초기 설계 에이전트 시스템의 구성도를 나타내고 있다.

이를 앞에서 언급한 IBM의 에이전트 시스템 개발 개념(Fig. 9a)과 비교하면 개선된 선박초기설계에이전트시스템의 내부구조는 Fig. 9b와 같다. 즉 IBM의 ABE가 KIF을 기반으로 한 Rule을 통하여 추론을 하는 데 비하여, 본 시스템은내부언어를 Prolog로 하

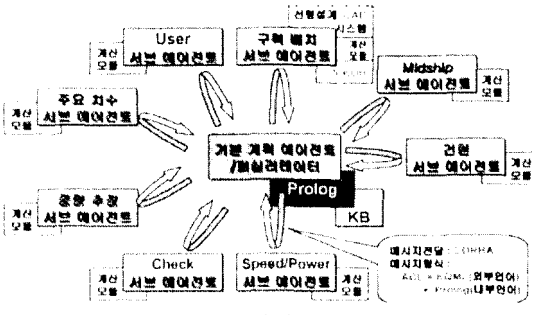

Fig. 8. Ship initial design agent system.

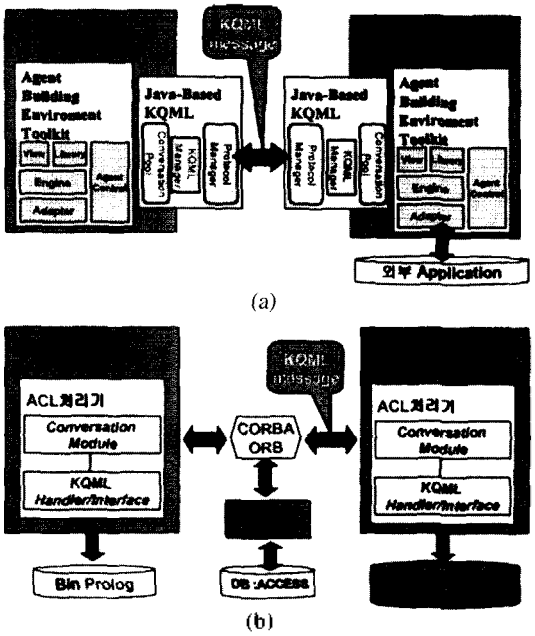

Fig. 9. (a) Concept for Building Agent System with ABE andJKQML. (b) Inner Structure of the Ship Initial Design Agent System

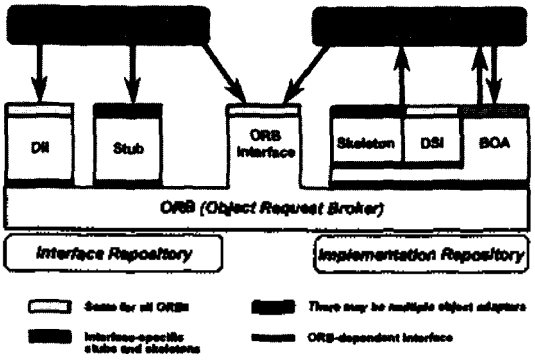

Fig. 10. System Architecture of CORBA.

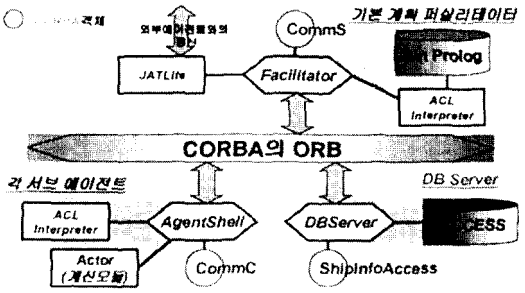

Fig. 11. CORBA objects of Ship Initial Design Agent System.

여 처리하였으며, IBM의 JKQML이 에이전트간의 통신과 KQML의 처리를 지원하는 것과 비교하여, 본 시스템은 Conversation Module, KQML Handler, 그리고 CORBA를 통하여 구현되었다.

4.3 CORBA를 이용한 에이전트간의 통신

CORBA(Common Object Request Broker Architecture)는 1989년 800개 이상의 회사로 이루어진 비영리 단체인 OMG(Object Management Group)에서 발표한 분산 환경 하에서 객체간의 정보 처리와 서로간의 운영 기능을 지원하기 위한 하나의 해결 방법이다.1991년 1.1 버전이 발표된 이후 1998년 3월 2.2 버전이 발표된 상태이다.

CORBA는 프로그래머에게 ORB(Object Request Broker)를 이용하여 프로그래밍을 하지만, 그 외에는 자신이 사용하는 분산 환경의 객체가 어디에서 누가 지원하는 지에 상관없이 전통적인 프로그래밍의 방법에 따라 작업을 하게 된다. 즉 분산 환경하의 모든 객체들이 마치 자신의 컴퓨터 아래에 위치해 있는 것처럼 보이게 해준다.즉 위치의 투명성을 제공한다는 것이다. 또한 CORBA는 IDL(Interface Definition Language)라는 별도의 인터페이스를 정의하는 언어를 사용하여 각 객체를 기술하게 된다. 이는 어떠한 특정언어만이 아니라, 다양한 프로그래밍 언어로 작성된 객체들간의 서비스의 교환이 가능하게 한다. 즉 언어에 대한 투명성을 제공한다. 이 외에도 여러 가지 분산 환경에서의 객체 또는 컴포넌트 기반의 시스템을 구현하기 위한 서비스를 제공한다. CORBA의 구조는 Fig. 10과 같다.

개선된 시스템의 구조는 Fig. 11과 같다. 기본 계획 에이전트/퍼실리테이터(이하 퍼실리테이터로 칭한다)와 각 서브 에이전트 사이를 CORBA를 이용하여 통신이 이루어지도록 하였다. 여기서 퍼실리테이터(Facilitator)는 에이전트간의 통신을 중계하고 업무 진행을 관리하고 또한 추론하는 기능을 가진일종의 슈퍼 에이전트라고 할 수 있다.

CommC와 CommS는 CORBA Object로 통신을 위한 method를 제공한다. ShipInfoAccess도 CORBA Object로 실질적인 선박의 데이터를 데이터베이스에서 읽고 저장하는 method를 제공한다. 데이터베이스와의 연결은 JDBC를 이용하여 구현하였다. CORBA를 채택함으로써 보다 효율적인 통신과 표준적인 분산환경을 구축하였다.

4.4 Prolog를 이용한 Flow Control의 구현

소프트웨어 에이전트의 가장 큰 특징이라 할 수

있는 추론을 설계의 Flow Control에 적용해 보았다. 이를 위해 초기 설계 에이전트 시스템에 KIF이외에 Prolog를 내부언어로 사용하여 메시지를 처리하는 퍼실리테이터를 두었다. 이 퍼실리테이터는 주요치수 에이전트, 중량추정 에이전트, Speed/Power 에이전트, 건현 에이전트, User 에이전트, Check 에이전트들간의 Flow를 추론을 통하여 얻어낸다. 초기설계 에이전트 시스템의 퍼실리테이터는 아래의 여러 서브에이전트들의 메시지들 안의 내부언어를 번역하기 위하여 BinProlog를 프로그램 내부에 하나의 Process로 가지고 있다.

지금의 상황은 State and change의 방법에 따라 Pre-Condition과 Post-Condition, 이에 따른 Action으로 기술되어 진다. 여기서 일반적으로 일어나는 상황, 모두들 일어날 것을 알고 있는 상황들의 Condition들은 미리 기록되어 기본 계획 퍼실리테이터가 실행될 때 Prolog의 Database안에 들어가게 되고, 만일 새로운 상황에 대한 기술이 필요할 경우에는 메시지를 통해 실시간으로 Prolog의 Database안에 들어가게 된다. Action은 지금 현재 상황을 다음 상황으로 나아가게 하는 것으로, 이것은 Prolog안에 서술되지 않고 각각의 에이전트들이 일을 수행한 후 바뀌어진 상황을 퍼실리테이터에게 메시지로 알림으로써 상황이 진행되도록 하였다.

각각의 구조와 함께 흐름을 보면 다음과 같다. 각 서브 에이전트의 구축에 있어, 하나의 통일된 Shell인 AgentShell을 구현하였다. AgentShell은 통신에 관련된 CORBA를 이용한 서비스와 메시지를 처리하는 서비스를 제공한다. 실질적인 설계 작업을 수행하는 부분은 Actor라는 클래스를 구현하고 수행하는 일을 Actor.ini에 KQML와 Prolog를 이용하여 나타내도록 하였다. 처음 서브에이전트가 실행되면 Actor.ini를 읽어 퍼실리테이터의 CommS Object를 통하여 자신들의 존재와 자신이 할 수 있는 일을 알린다. 그러면 퍼실리테이터는 받은 메시지를 자신의 KQML 인터프리터에게 보내게 되고 추론에 필요한 내용은 Prolog의 Database에넣게 된다. 이것을 바탕으로 추론을 하여 일을 수행할 수 있는 서브에이전트들에게 CommC Object를 통해 메시지를 보내게 된다. 해당 일을 마친 서브에이전트는 CommS Object를 통해 바뀌어진 상태에 대한 메시지를 보내고 자신이 그 일을 마쳤음을 알림으로 다음 상태로 바뀌고 그것을 바탕으로 추론을 하는 반복작업을 하는 방식이다. Fig. 12는 기본설계 에이전트/퍼실리테이터의 구조이고 Fig. 13은 각 서브 에이전트들의 구조이다.

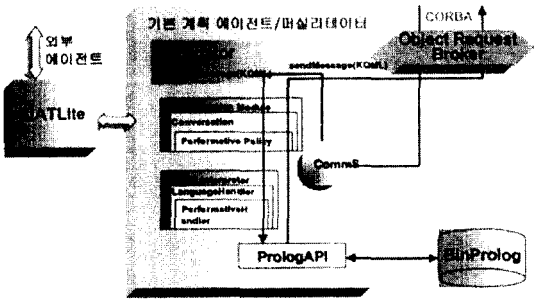

Fig. 12. System Structure of facilitator.

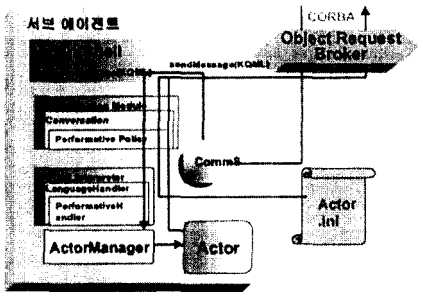

Fig. 13. System structure of Sub Agent.

4.5 Conversation Module

선박 초기설계에이전트 시스템의 초기버전에서는 KQML interpreter를 구현함에 있어서 performative의 운용에 대한 원칙이 명확하지 않았다.KQML은 에이전트들이 메시지에 담긴 내용에 대한태도와 내용을 전달하는 언어로서 이러한 태도를 나타내기 위하여 사용되는 것을 performative라고 한다. 따라서 KQML을 사용한 메시지가 의미 있는 통신이 되기 위해서는 각각 사용하는 performative들이 일정한 원칙에 따라 운영되어야만 한다. 개선된 시스템에서는 각 performative의 명확한 의미와 쓰임의 올바름을 검사하기 위해 Conversation Module을 추가하였다.

Conversation Module은 각 에이전트에 오는 모든 메시지를 기존의 일괄적인 관리가 아닌 대화단위로 관리하며 conversation policies에 따라 어떤 performative가 또 다른 어떠한 performative를 따라올 수 있게 하여 의미 있는 메시지를 통한 대화가 이루어지도록 한다. 각 메시지의 Policy는 Finin이 제안한 6가지 관점[8/12], 자연어 기술, logic에 따른 expression, agent가 performative를 보내고 receiver가 그것을 받고 처리하기 위한 state를 나타내는 preconditions, agent가 performative를 보낸 후 그리고 recei-

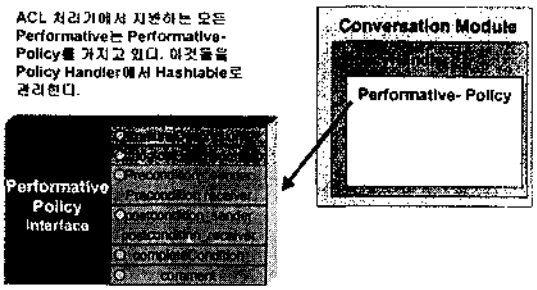

Fig. 14. Implementation of Conversation Module.

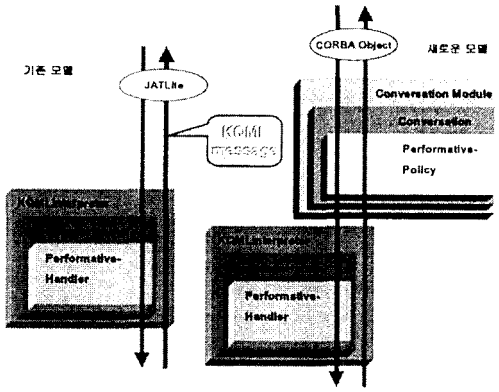

Fig. 15. Structure of Conversation Module and flow of KQML messages.

ver가 그것을 받고 처리한 다음의 state를 나타내는 postconditions, sender의 최종 상태를 나타내는 sender의 completion conditions, 자연어 comments에 따라 Java의 interface로 구현하였다(Fig. 14).

Fig. 15는 Conversation Module의 구조와 메시지의 흐름을 초기 버전과 비교하여 나타낸 것이다. Conversation Module은 KQML interpreter와 Application사이에위치하여, 각 에이전트에 들어오고 나가는 메시지의 pre/postcondition을 확인하고 Conversation 단위로 메시지를 관리한다.

각 performative의 policy는 다음과 같은 predicate을 사용하여 나타낸다.

-bel(A, P)는 에이전트 A에게 있어 P가 참임을 나타낸다.

-know(A, P)는 A는 P를 알고 있음을 나타낸다.

-want(A, P)는 A가 P로 나타낸 일이 일어나길 원함을 뜻한다.

-intend(A, P)는 P를 수행하고자 함을 나타낸다. 이들을 이용하여 각 performative의 의미와 사용조건을 나타냈다.

하나의 예로서 ask-if의 semantic을 보면 다음과 같다.

ask-if(A, B, X)

▼ A는 B가 content의 진실에 대하여 B가 믿는가.
want(A, know(A,Y)) Y는 다음 중에 하나임.

-bel(B, X), bel(B, NOT(X)), NOT(bel(B, X))

-Pre(A)는 다음과 같은 것으로 시작할 수 있다.

want(A, know(A, bel(B,X))) or
want(A, know(A, bel(B, NOT(X))) or
want(A, know(A, NOT(bel(B,X))))

▼ Pre(A): want(A, know(A,Y))
Pre(B): NONE

▼ Post(A): intend(A, know(A,Y))
Post(B): know(B, want(A, know(A, Y)))

▼ Completion(A): know(A, Y)

▼어떠한 것을 믿지 않는 다는 것은 특별한 경우를 제외하고는 그것의 반대를 믿고 있다는 것과 반드시 같은 것으로 취급되지 않는다.

이렇게 정의한 semantic을 사용하여 Performaive-Policy interface를 구현하게 된다. Conversation에서는 이 performative를 보낼 때, 지금 에이전트의 상태가 사용하기 위한 조건을 만족하는지를 검사하고, 사용이 가능할 경우 메시지를 전달하고, 에이전트의 상태를 업데이트하게 된다. 메시지를 받은 에이전트는 이 메시지가 자신이 행하고 있는 대화에 알맞은 것인지를 검사한 후, 처리하게 된다.

4.6 Agent와 퍼실리테이터와의 메시지 전달

KQML메시지와 에이전트의 상태는 간략히 필요한 부분만 표시하였다.

1) User 에이전트가 퍼실리테이터에게 자신의 존재를 알린다.

(register: sender agent1: receiver facilitator1: language KQML: content agent1)

User 에이전트의 메시지 보내기 전의 상태:

know(agent1,
know(facilitator1,presence(agent1)))
know(agent1,
know(facilitator1,symbol(agent1)))

퍼실리테이터의 메시지 받은 후의 상태:

know(facilitator1,presence(agent1))
know(facilitator1,symbol(agent1))

2) User에이전트는 자신이 선주 요구조건을 받아들일 수 있음을 퍼실리테이터에게 알린다.

(advertise: sender agent1: receiver facilitator1
:language KQML: content

(achieve: language prolog
:content "inputDimensions(done)"))
User 에이전트의 메시지 보내기 전의 상태 :
bel(agent1,
process(achieve,inputDimensions(done)))
퍼실리테이터의 메시지 받은 후의 상태:
know(facilitator1,
bel(agent1,
process(achieve,
inputDimensions(done))))

3) 퍼실리테이터는 추론을 통하여 User 에이전트가 작업을 할 수 있는 상황이 되면 작업을 요청한다.

(achieve: sender facilitator1: receiver agent1
:language prolog
:content "inputDimensions(done)"))
User 에이전트의 메시지 받기 전의 상태:
bel(agent1,
process(achieve,inputDimensions(done)))
퍼실리테이터의메시지 보낸 후의 상태:
intend(facilitator1,
bel(agent1,inputDimensions(done)))

4) User 에이전트는 작업을 마친 후 그 사실을 퍼실리테이터에게 알린다.

(tell: sender agent1: receiver facilitator1
:language prolog
:content "inputDimensions(done)"))
User 에이전트의 메시지 보내기 전의 상태:
bel(agent1,
inputDimensions(done))
퍼실리테이터의메시지 받은 후의 상태:
know(facilitator1,
bel(agent1,inputDimensions(done)))

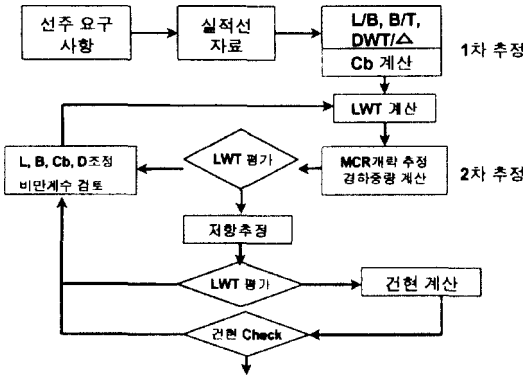

Fig. 16. flow of deciding principal dimensions of ship.

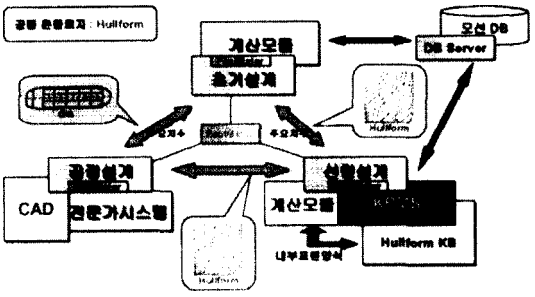

Fig. 17. Ship Design Agent System.

5) 퍼실리테이터는 바뀐 상황에서 작업을 수행할 수 있는 에이전트를 추론을 통해 찾아서 작업을 요청한다.

이러한 과정을 통하여 Fig. 16과 같은 선박의 초기 설계의 주요치수 결정의 과정을 추론을 통하여 구현하였다.

4.7 외부 에이전트와의 통신

초기설계 에이전트 시스템은 선박설계 에이전트 시스템의 일부이다.선박설계 에이전트 시스템은 Fig. 17에서 보는 바와 같이 초기설계 에이전트 시스템, 선형설계 에이전트 시스템, 그리고 공정설계 에이전트 시스템으로 구성되어 있다. 각 에이전트 시스템들과의 통신에는 JATLite를 이용하며 내부언어로 KIF를 사용한다.

4.8 구현된 선박 초기설계 에이전트 시스템의 실행 예

본 연구에서 구현한 초기설계 에이전트 시스템을 이용하여 주요치수를 추정하는 과정을 설명한다.

1. 퍼실리테이터가 실행된다. 처음 실행된 퍼실리테이터는 추론에 필요한 외부 Prolog 인터프리터를 실행시켜 자신의 프로세스로 가진다. 또, 파일로 작성되어 있는 초기설계에 관련된 Prolog Rule들과 Fact를 읽어와 초기화시킨다.

2. 퍼실리테이터가 KQML 메시지의 전달에 필요한 CORBA객체들을 생성하고 ORB에 등록한다.

3. 각 서브에이전트가 실행되면 KQML 메시지의 전달에 필요한 CORBA객체들을 생성하고 ORB에 등록시킨다.

4. 각 서브에이전트는 자신의 초기화 파일(Actor.ini)을 참조하여 자신의 이름과 할 수 있는 일들을 서술한 KQML메시지를 읽어온다.

5. 각 서브에이전트는 자신의 초기화 파일을 통해

얻어온 이름을 가지고 퍼실리테이터에게 자신을 등록한다. 이때 register performative를 사용하여 내용으론 자신의 이름을 전달한다.

예: 서브에이전트 agent1이 퍼실리테이터에게 등록하는 KQML 메시지

```
(register :sender agent1 :receiver facilitator1
 :reply-with agent101 :language KQML
 :content agent1)
```

이러한 메시지들은 conversation Module에서 conversation단위로 관리되게 된다.

6. 퍼실리테이터는 각 서브에이전트가 보내온 KQML 메시지가 지금 상황에 알맞게 사용되고 있는지를 Conversation Module에서 검사한다. Register의 경우, 퍼실리테이터에게 조건없이 사용할 수 있음으로 조건을 만족하고, 진행되는 대화가 없음으로 Conversation객체를 생성한다(Fig. 18).

Register메시지를 통하여 사용 가능한 서브에이전트의 이름을 얻고, 그 이름으로 각 서브에이전트의 통신에 필요한 CORBA 객체를 찾아, 이름을 Key값으로 하여 Hashtable이란 자료형태로 관리하게 된다.

7. 각 서브에이전트는 자신이 할 수 있는 일들을 Advertise performative를 이용하여 퍼실리테이터에게 알린다.

예: 서브에이전트 agent1은 선주의 요구조건을 받아들일 수 있는 능력을 가지고 있음을 알리는 KQML 메시지

```
(advertise :sender agent1 :receiver facilitator1
 :reply-with agent102 :language KQML
 :content (achieve :sender facilitator1
  :receiver agent1 :in-reply-to agent102
  :language prolog
  :content "inputDimensions(done)") )
```

Advertise는 에이전트가 어떤 일을 수행할 수 있음을 알리는 performative며, 내용은 꼭 KQML문장이와야한다. 따라서 language는 KQML이란 값이고

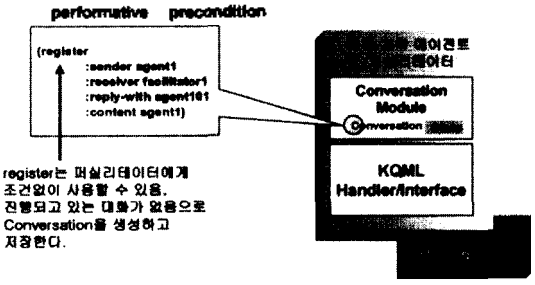

Fig. 18. Handling of 'Resister' message in Facilitator.

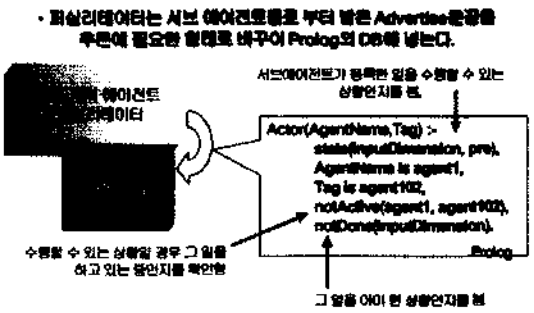

Fig. 19. Handling of 'Advertise' message in Facilitator.

content란엔 KQML메시지가 들어간다. Achieve는 content값을 참으로 만들라는 performative이다. 즉 위의 문장은 "선주의 요구조건을 입력받았다는 상태가 참으로 되게 만드는 일을 에이전트 1이 할 수 있다"란 뜻이다.

8. 퍼실리테이터는 각 서브에이전트가 보내온 advertise 문장들을 추론에 사용하기 위하여 Prolog 안에 넣는다(Fig. 19).

9. 퍼실리테이터는 현재 일을 수행할 수 있는 서브에이전트가 어떠한 것인지를 추론을 통하여 찾아낸다. 초기상태에서는 아무런 일이 진행되고 있지 않기 때문에 선주의 요구조건을 입력받아야한다. 따라서 등록된 agent1이 작업을 수행할 수 있다는 결과를 얻어내게 된다.

10. 퍼실리테이터는 agent1에게 선주의 요구조건을 입력받으라는 KQML메시지를 보낸다. 모든 KQML메시지들은 사용가능한지 Conversation Module에서 검사한다(Fig. 20).

11. 퍼실리테이터에게 작업을 수행하도록 요구받은 서브에이전트 agent1은 자신의 conversation module을 통하여 메시지를 검증한 후, 올바를 경우 자신에게 등록된 actor클래스를 통하여 작업을 수행한다. Fig. 21은 작업을 수행하는 agent1의 GUI이다.

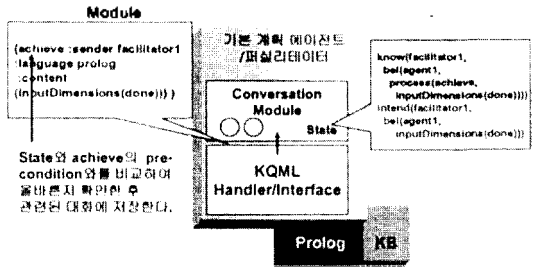

Fig. 20. Conversation Module : Conversation Module checks all message to decide if they are allowable.

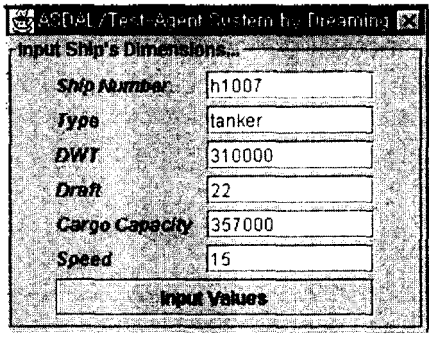

Fig. 21. GUI of user subagent.

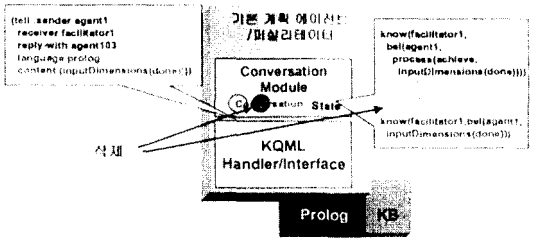

Fig. 22. Conversation Module in Facilitator.

12. 작업을 수행한 서브에이전트 agent1은 입력받은 결과를 데이터 베이스에 저장하고, 작업을 수행했음을 퍼실리테이터에게 알린다.

13. 퍼실리테이터의 conversation module은 agent1에게 전달한 메시지의 대화가 작업이 종료됨에 따라 마친 것을 확인한 후, 해당 conversation 객체를 지운다 (Fig. 22). 그리고 추론을 통하여, 주요치수의 1차 추정을 처리할 수 있다는 결과를 얻어, 주요 치수 에이전트에게 일을 수행할 것을 KQML메시지를 통해 알린다.

14. 주요 치수의 추정이 끝나면 퍼실리테이터는 중량 추정을 할 수 있음을 등록한 서브에이전트에게 일을 수행할 것을 요구한다.

15. 주요 치수의 추정과 중량 추정이 끝나면 퍼실리테이터는 이를 바탕으로 추정치가 올바른지 check 해야 함을 추론으로 얻어내고, check 서브에이전트에게 일을 수행하도록 한다(Fig. 23).

16. 퍼실리테이터는 이후 건현과 선박의 추진력(power)에 대한 1차 추정을 동시에 시킨다.

17. 앞의 15번과 같이 check 에이전트에게 얻어진 추정치가 올바른지 확인하도록 한다.

18. 기준선박의 선형을 앞에서 구한 조건에 맞추어 variation을 한 후, Midship에 대한 가이드 값을 생성한 후, 사용자에게 GUI를 통해 조정을 요구한다 (Fig. 24).

19. 구획배치 서브에이전트는 대략적인 구획배치

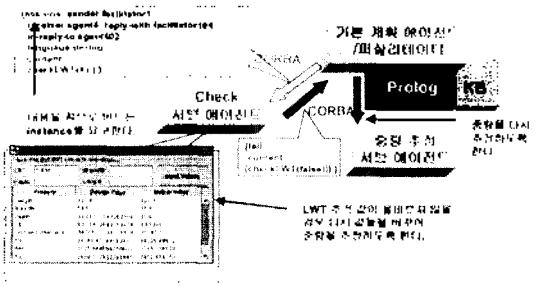

Fig. 23. GUI of Check SubAgent and Flow of checking the assumed weight information.

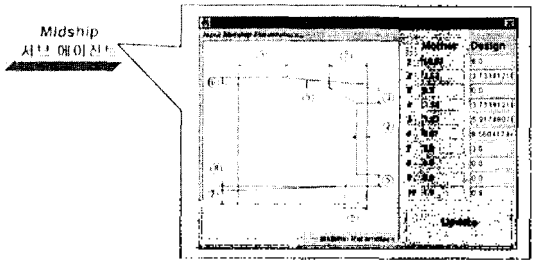

Fig. 24. GUI of midship subagent.

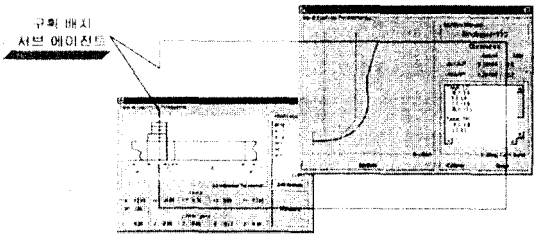

Fig. 25. GUI of general arrangement subagent.

를 한 후, 사용자에게 가이드 값을 제시하고, 확인을 요구한다. (Fig. 25)

20. 구획배치 서브에이전트는 입력받은 값을 바탕으로 외부 선박계산 프로그램을 사용하기 위한 input 파일을 생성하고, 그 수행결과를 GUI를 통하여 사용자에게 보여준다. (Fig. 26)

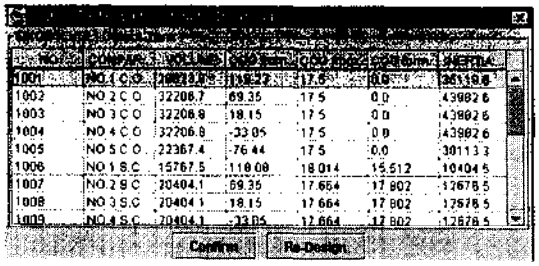

Fig. 26. Output of external ship calculation program.

5. 결론과 향후 연구 과제

본 논문은 지식을 표현할 수 있는 에이전트 통신 언어(ACL)를 이용하여 추론 기능을 갖는 하나의 설계 에이전트 시스템을 구현하였다. 선박 초기설계에 이전트 시스템의 설계업무 제어(Design Flow Control)는 Prolog를 이용하여 추론으로 구현하였으며, 에이전트간의 통신 기능과 효율적인 분산 환경 프로그래밍을 위해 CORBA를 도입한 에이전트 아키텍쳐를 구현하였고, 에이전트간의 메시지들은 KQML과 Prolog를 사용하였다. 또 설계선 및 기준선의 데이터 관리와 공유를 위한 데이터베이스 서버를 JDBC를 이용하여 구축하였다.

향후 연구과제로는 실제 설계업무에 대응하는 정교한 design flow control과 KQML 처리기를 각 언어의 사양에 충실한 모듈로 확장해야 할 것이며, 효과적인 지식의 공유를 위한 온톨로지(Ontology) 구축 등에 대하여 연구되어야 할 것이다.

후 기

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

참고문헌

1. R. Patil, R. Files, P. Patel-Schneider, D. McKay, T. Finin, T. Gruber and R. Neches, "The DARPA Knowledge Sharing Effort: Progress Report", *Principals of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*, 1992.
2. T. Finin, Y. Labrou and J. Mayfield, "KQML as an agent communication language", *Software Agents*, MIT Press, 1995. Available as http://www.cs.umbc.edu/kqml/papers/.
3. Object Management Group, The Common Object Request Broker: Architecture and Specification, 2.0ed, July, 1995.
4. Robert Orfali and Dan Harkey, Client/Server Programming with JAVA and CORBA, John Wiley & Sons, 1998.
5. http://www.networking.ibm.com/iag/iagsoft.htm, IBM Agent Building Environment
6. http://www.alphaworks.ibm.com/formula/jkqml, JKQML, IBM Yamato Lab.
7. Dominique BENECH, Thierry DESPRATS, Yves RAYNAUD, "COBALT: An Architecture for Intelligent Agent-based Management", NOMS'98-15-20 Feb 1998, Available as http://www.irit.fr/~Dominique.Benech/docs/noms98/index.htm
8. 이규열, 연윤석, 김수영, 윤덕영, "KII를 토대로 한 선박설계 에이전트 시스템 사양개발 및 그 구현 예에 관한 연구", *조선학회 1997년도 춘계학술대회 논문집*, 1997. 4.
9. 이상욱, 이규열, "에이전트 기반의 시스템 통합을 위한 에이전트 기본 아키텍쳐에 관한 연구", *조선학회 1997년도 추계학술대회 논문집* 1997. 11.
10. M. Genesereth, R. Fikes, *et al.* "Knowledge Interchange Format (KIF), version 3.0 reference manual. Technical report", Computer Science Department, Stanford University, 1992. Available as http://www-ksl.stanford.edu/abstracts_by_author/Genesereth,M papers.html.
11. http://java.stanford.edu/java_agent/html/JATLiteHomepage.
12. Y. Labrou and T. Finin, "A semantics approach for KQML", *Third International Conference on Information and Knowledge Management(CIKM'94)*, ACM Press, November 1994.

김 동 현

1997년 서울대학교 조선해양공학과 학사
1999년 서울대학교 조선해양공학과 석사
1999년 ~현재 현대중공업 산업기술연구소 자동화연구실 공정팀
관심분야: 일정계획, 시뮬레이션, CIM

이 규 열

1971년 서울대학교 공과대학 조선공학과 학사
1975년 독일 하노버 공과대학 조선공학 석사
1982년 독일 하노버 공과대학 조선공학 박사
1975년~1983년 독일 하노버 공과대학 선박설계 및 이론연구소, 주정부 연구원
1983년~1984년 한국기계연구원 선박해양공학연구센터, 선박설계, 생산자동화 연구사업(CSDP) 단장
1984년~현재 서울대학교 공과대학 조선해양공학과 부교수
관심분야: 최적설계, 형상모델링, CALS

이 상 욱

1996년 서울대학교 조선해양공학과 학사
1998년 서울대학교 조선해양공학과 석사
1998년~현재 서울대학교 조선해양공학과 박사과정
관심분야: Agent-based Engineering, Computer-Aided Geometric Design