論文
6-3-15

AUML기반의 소프트웨어 컴포넌트들의 협력성을 위한 검증 모델에 관한 연구

A Study About Verification Model for Cooperation of Software Components of AUML Base

한현관(Han Hyoun Gawn)[1] 박재복(Park Jae Bock)[2]

요약

AUML(Agent Unified Modeling Language)은 에이전트 소프트웨어 시스템의 명세화, 시각화, 생성을 목적으로 하는 언어이다. 본 연구에서는 소프트웨어의 복잡화, 대형화 추세에 자동화 응용 프로그램 생성 시스템들 중의 하나인 Together를 Agent의 BDI에 응용시키고 이를 컴포넌트 시스템의 상호 운영성에 대하여 고찰 한다. 상호 운용성은 컴포넌트간의 데이터 교환에 의해 이루어지며 컴포넌트의 타입이 다르더라도 서로 협력할 수 있는 표준 명세서(FIPA:Foundation for Inteligent Physical Agent)를 기반으로 ACL 메시지, 그리고 프로토콜을 사용하며 이를 객체지향 모델링을 통한 메타모델기반 등을 이용하여 구현 시 오류를 최소화하는 방법과 정확성과 일관성에 관하여 연구한다.

Abstract

AUML (Agent Unified Modeling Language) is specification anger of agent software system, sight anger, language that do creation by purpose.

Do so that may apply Together that is one of automation application program creation system to Agent's BDI in trend sophistication of software, large size Tuesday in this research and investigate this about operation between component system.

Standard detailed statement (FIPA:Foundation for Inteligent Physical Agent) that use can consist by data exchange between component and cooperate each other even if type of component is different mutually to base ACL message, and protocole use and study about method and accuracy and consistency that minimize error when embody this using meta model base etc.. through object intention modelling.

논문접수 : 2005. 6. 30.
심사완료 : 2005. 7. 15.

1) 정회원 : 대구산업정보대학 겸임교수

2) 정회원 : 경북과학대학 컴퓨터미디어학과 교수

1 서론

인터넷을 통한 e-비즈니스 시장의 폭발적인 성장은 자료 관리/공유, 전자결재, 그리고 문서 전달을 원활케 하는 소프트웨어 시스템의 개발을 요구하고 있다. 특히, 최근에 개발된 소프트웨어들은 대용량이면서 복잡한 모습을 보이고 있는데, 이를 해결하기 위해 객체지향 개발 방법론이 소개되어 많이 이용되고 있으며[3, 4, 5], 개발 방법론의 연구를 체계적으로 지원하기 위한 CASE 환경에 관한 연구도 진행되고 있다[1]. 그러나 CASE 도구들[7]은 서로 다른 형식의 모델링 설계 정보를 사용하기 때문에 소프트웨어 개발에 요구되는 정보 교환/공유를 힘들게 한다는 문제점을 가지고 있다.

본 논문에서는, 소프트웨어 개발의 전 과정을 지원하는 자동화 도구인 Together[7]를 다음과 같은 세 가지의 모덜러티(modality)들을 고려한다: 환경의 상태에 대한 에이전트의 정신적 태도(mental attitude)를 나타내는 믿음 *B*, 에이전트의 동기(motivation)를 나타내는 소망 *D*, 그리고 에이전트의 목표를 나타내는 의도 *I*. 또한 자원이 한정된 BDI 에이전트들이 서로 협상할 수 있도록 이들을 위한 통신행위들을 제안하고, 이러한 통신행위들을 사용하는 협상 프로토콜을 소개한다. 마지막으로 간단한 시나리오를 통하여 제안한 ACL의 응용 가능성을 검사하고, 이를 에이전트 지향 소프트웨어공학(AOSE: Agent-Oriented Software Engineering) 검증 모델에 적용 한다.

완전성을 검증하는 방법에는 정형명세기반 검증방법이 있으며, 일관성과 정확성을 검증하기 위한 방법에는 메타모델기반 검증방법이 있다.

이를 바탕으로 하여

- 검증규칙을 기반으로 다이어그램의 일관성을 검증
- 제약언어를 사용하여 검증규칙을 표현
- 이를 자동화된 응용프로그램 생성 도구 Case Tool을 적용시켜 상호 비교 분석 및 정확성과 일관성을 검사한다.

2 관련 연구

본 연구배경은 소프트웨어 공학에서의 컴포넌트를 재사용 가능한 컴포넌트로 만들기 위해서 필요한 에이전트 지향의 소프트웨어 공학과 에이전트 간 통신관계 등 고찰하며 아울러 컴포넌트, 상호운영에 대해 살펴보며 확장된 성능, 조건 등을 기술하기 위한 UML(Unified Modeling Language), 정확성을 위하여 UML 다이어그램의 메타모델(metamodel)등을 기술하여 분산된 컴포넌트들을 쉽게 이용하고 효율적인 시스템 구축을 위한 방안을 제공하는 것이 본 논문의 필요성이자 목적이다.

2.1 컴포넌트 기반의 소프트웨어 공학

컴포넌트 기반의 소프트웨어 공학 (Component-Base Software Engineering : CBSE)이라 함은 기존의 컴포넌트를 조합함으로써 시스템을 완성하는 것을 말하며 기존의 소프트웨어 시스템 개발 시 효율적으로 사용하지 못했던 기존의 방식의 해결책으로써 소프트웨어 재사용과 관리적 측면에서 더욱더 연구되고 개발되고 있다.

컴포넌트 기반의 소프트웨어 공학의 분야는 컴포넌트 디자인 및 합성 분야와 컴포넌트와 컴포넌트 사이의 상호 작용을 분석하고 기술하는 소프트웨어 아키텍처 디자인 분야로 이루어져 있다.

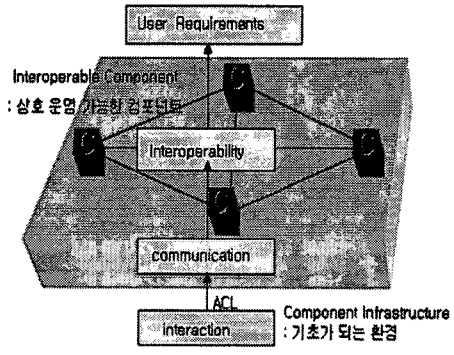
예를 들어 C++에서는 .h와 .cpp 파일을 나타내고, Java에서는 .java 파일로 나타낸다.

2.1.1 컴포넌트 상호 운용성

컴포넌트간의 상호 운용의 기본은 컴폰넌트간 통신이고, 이 통신은 프라스트럭쳐의 기초가 된다. 컴포넌트간 신과 인프라스트럭쳐는 분산된 자원을 공유하기 위한 필 수 조건이다.

두 개의 컴포넌트가 사용자 요구 사항을 수용하기 위한 행위가 상호작용이며, 이 데이터가 서로 주고받을 때를 통신이라 한다. 통신이 공통된 환경 즉 인프라스트럭쳐를 바탕으로 이루어 질 때, 상호 운용 가능한 컴

포넌트(interoperable component)라고 하며 상호 운용 가능한 컴포넌트는 협력과 조정이라는 기본 속성을 가진다. 아래의 [그림 1]은 컴포넌트간의 상호운용성의 개념을 표현 하였다.



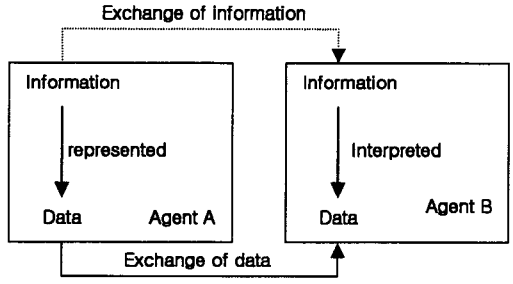
[그림 1] 컴포넌트 구조
[Fig.1] Structure of Component

2.2 에이전트 지향의 소프트웨어 공학

에이전트 지향의 소프트웨어 공학(AOSE: Agent-Oriented Software Engineering)은 1990년대 말부터 활성화 되었으며 크게 영역분석 및 모델링 기술, 에이전트 아키텍쳐 설계 기술, 에이전트 구현 도구 등이며 이질적이고 분산된 서로 다른 시스템에서의 통신과 협력에 의해서 사용자의 요구사항을 만족시키는 것이다.

컴포넌트가 다른 에이전트와 상호작용(Interaction)이 가능한 것은 정보의 교환을 의미 하며 정보의 교환(exchange of information)은 통신이며 이 정보는 데이터(data)로 되어 컴포넌트간의 협력(cooperation)이 이루어진다.

보내는 컴포넌트의 정보를 받는 컴포넌트는 정해진 해석(interpretation)규칙에 따라 정보를 해석하고 의미를 얻기 위해서는 구조화가 잘 되어야 하는데 이러한 구조화를 이루기 위해서 본 논문에서는 에이전트 통신 언어(ACL: Agent Communication Language)를 이용 한다 [2].



[그림 2] 두 컴포넌트간의 정보 교환
[Fig.2] Exchange information of Two component

2.3 에이전트간 통신

2.3.1 FIPA 개요

분산된 자원의 공유와 관리를 위하여 에이전트가 특정 플랫폼에 상주하여 다른 에이전트와 메시지를 교환하고, 협력하여 문제를 해결할 수 있는 기반을 제공해야 한다. 이를 위한 해결 방안이 FIPA의 표준 명세서이다.

2.3.2 에이전트간 통신

에이전트는 어떤 행위를 수행할 것인가를 결정하기 위해서 그리고 그 행위와 다른 에이전트의 행위를 어떻게 조정할 것인가를 결정하기 위해서 서로 통신할 수 있어야 한다.

이러한 정보교환을 위해 에이전트가 사용하는 언어가 에이전트 통신언어(Agent Communication Language: ACL)이다. ACL의 주된 목적은 이질적인 에이전트들이

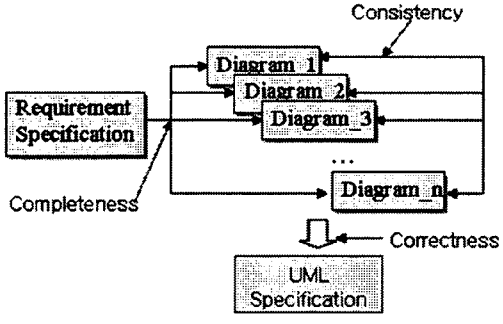
상호작용할 수 있게 하는 적당한 프레임워크를 모델링하고, 환경과 지식에 관한 정보를 전달하는 의미 있는 문장들을 교환하는 것이다. 따라서 ACL은 다음과 같은 바람직한 여러 가지 특징들을 가져야 한다. 에이전트는 자신의 환경과 목표에 대한 지식을 관리하고 사용하기 때문에 ACL은 지식표현 언어이어야 한다. 또한 ACL은 선언적이고, 구문적으로 간단하고, 사람들이 판독할 수 있어야 한다. 마지막으로 ACL은 통신행위에 대한 정확한 의미론(semantics)을 제공하여야 하며, 네트워크 환경

에 적합한 언어이어야 한다.

일반적으로 ACL은 담화행위(speech act) 언어 형태로 표현되는데, 여기서 담화행위는 화자 의 정신적 상태의 일부를 청자에게 전달하기 위해 화자가 수행하는 행위로 정의된다. 이러한 개념이 적용된 ACL들로써 KIF(Know-ledge Interchange Format), KQML(Knowledge Query Manipulation Language), FIPA(Foundation for Intelligent Physical Agents) ACL 등이 존재한다. 그러나 에이전트 시스템의 구조와 에이전트 응용 분야는 ACL의 설계에 상당한 영향을 미치며, 모든 구조와 응용 분야를 포함할 수 있는 ACL의 설계는 불가능하다.

2.4 모델의 정확성 검증

모델의 정확함을 파악하기 위해서는 완전성(Completeness), 일관성(consistency), 정확성(correctness)을 고려해야 하며 각 성질은 다음과 같이 정의 한다.



[그림 3] 다이어그램간의 정확함에 대한 개요
[Fig.3] Abstract about correctness between diagram

기존의 정확함에 관한 연구는 완전성을 검증하는 방법과 일관성과 정확성을 검증하는 방법에 관한 연구로 나누어 볼 수 있다.

완전성을 검증하는 방법에는 정형명세기반 검증방법이 있으며, 일관성과 정확성을 검증하기 위한 방법으로는 메타모델기반 검증방법이 있다.

일관성 및 정확성 검증에 관한 연구로서 메타모델기반 검증방법은 초기 작업으로 UML 다이어그램의 메타모델을 유도하고, 나타난 다이어그램 사이의 관계성을 바탕으로 일관성을 검증하기 위한 검증규칙을 유도한다.

제약언어를 이용한 검증방법은 객체지향모델이 정확하게 작성되어야함을 나타내는 제한조건을 사용하여 모델의 정확함을 검증하는 방법이다. 이 방법은 객체모델을 고유한 명세로 표현하고, 일관성 또는 완전성과 같은 제약조건을 특정한 제약 언어로 표현한다. 제약 조건은 검증시스템의 검증 알고리즘을 사용하고, 명세된 객체모델에 검증 알고리즘을 적용하는 방법을 취한다.

2.4.1 OMG의 UML 메타모델

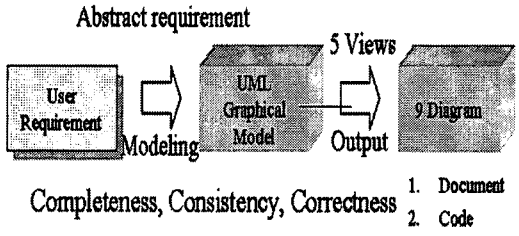
검증방법론 측면에서는 메타모델과 메타-메타모델 그리고 검증규칙을 기반으로 다이어그램의 일관성을 검증하는 방법을 택하며 특징적으로 UML의 표준제약으로 사용되고 있는 OCL을 이용하여 검증한다. 검증 방법의 전체 구조는 아래의 그림에서 나타내었으며 검증 과정은 다음과 같다. 첫째, 검증의 기초 작업으로 UML 명세로부터 각 다이어그램의 메타모델을 유도하고, 유도한 메타모델을 바탕으로 정적 다이어그램과 동적 다이어그램 사이의 관계를 표현하는 메타-메타모델을 유도 한다. 둘째 유도한 검증 규칙은 OCL을 이용하여 정형적으로 표현한다.

3 모델링의 개요 및 분석

3.1 UML 모델링

UML은 객체지향 분석과 설계를 위한 모델링 언어이며, 어휘와 규칙을 사용하여 시스템을 개념적이고 물리적으로 표현할 수 있게 한다[6]. 아래의 그림과 같이 추상적인 요구사항을 가시화하고, 사용자의 요구사항을 보다 완전하게 명세하며, 다양한 프로그래밍 언어와 연결되어 코드를 생성한다. 본 연구에서는

Agent의 BDI 예제로 하여 UML을 적용시켜 시스템을 분석 및 설계한다. 그리고 요구사항 분석 단계를 기반으로 객체 모델 모듈 다이어그램을 생성시키는데, 이를 자동으로 생성시키기 위해 Together를 이용한다.



[그림 4] UML의 역활
[Fig.4] UML's underpart

3.1.1 UML 표준을 지원하는 다이어그램들

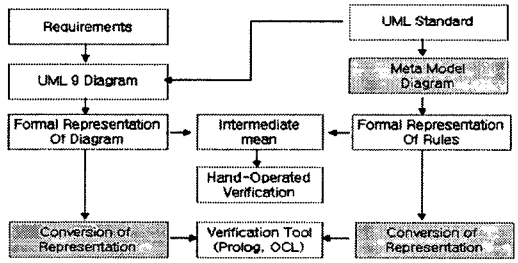
Use CASE 다이어그램은 행위와 Use-CASE 사이의 관계를 보여주며 시스템과 업무 사이의 전체적인 흐름을 파악할 수 있게 한다. 활동(activity) 다이어그램은 활동들을 표현하고, 활동들 사이의 전이와 결정점들을 나타낸다. 그리고 클래스(class) 다이어그램은 객체의 타입인 클래스를 표현하고, 클래스의 속성과 연산, 연관, 합성, 위임, 일반화, 그리고 패키지 등의 다른 클래스들과의 정적인 관계에 대한 제약 등을 표현할 수 있다.

한편, 순차(sequence) 다이어그램은 열 좌표축으로 시간 개념을 도입하고 행 좌표축으로 객체를 나열하여 그 사이의 상호작용을 표시한다. 마지막으로, 협동(collaboration) 다이어그램은 시나리오를 표현하는 또 다른 방법으로 객체들 사이의 상호작용과 연결을 나타내며, 특정한 객체들의 집합의 동적인 특성을 표현할 수도 있다.

3.1.2 일관성 및 정확성 검증

다이어그램간의 분석은 UML 9개 다이그램 사이의 관계성을 분석하여 다이어그램간의 일관성검증을 위한 관계요소를 파악하고 다이어그램간의 변환 오퍼레이션을 유도하여 특정 다이어그램으로부터 또 다른 다이어그램으로의 변환을 한다. 이러한 변환 오퍼레이션은 모델 검증, 모델 통합, 모델 분리 등에 이용한다.

아래의 그림은 일반적인 일관성 및 정확성 검증절차에 대한 개요를 표현 하였다.



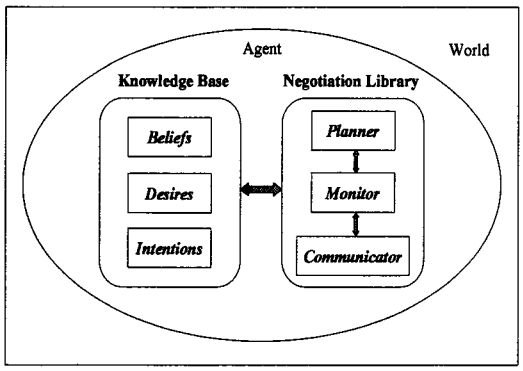
[그림 5] 일관성 및 정확성 검증절차에 대한 개요
[Fig.5] Abstract about consistency and accuracy verification procedure

3.2 Agent 모델링

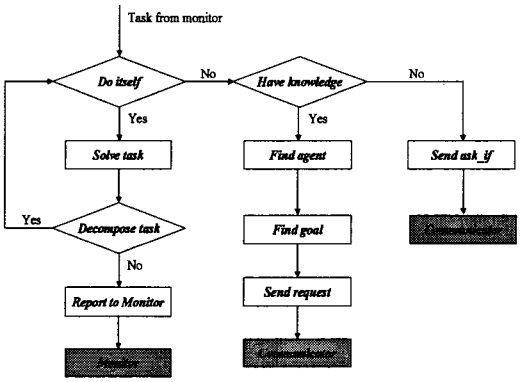
일반적으로 에이전트는 자신의 목표 달성에 필요한 지식, 작업을 계획하는데 필요한 지식, 그리고 다른 에이전트와 통신하기 위한 지식 등을 필요로 한다. 예를 들어, 에이전트는 문장들의 집합으로 표현 될 수 있는 지식을 필요로 한다. 문장들은 자신의 믿음과 능력에 관한 지식, 상호작용 가능한 다른 에이전트들에 관한 지식, 통신에 필요한 지식, 그리고 특정한 응용영역에 관한 지식 등을 서술한다.

에이전트를 설계하기 위해서[그림 6]과 같이 지식베이스와 협상 라이브러리로 구성된 BDI 에이전트 구조를 고려한다. 여기서 지식베이스는 그 에이전트의 능력과 다른 에이전트들의 능력에 관한 지식 그리고 문제 분해를 위한 규칙 등에 관한 지식을 포함하는 논리적인 문장들의 집합이다. 지식베이스에 있는 요소들은 그 에이전트의 정신적인 태도를 의미하는 술어(predicate)들로 표현한다. 반면에 협상 라이브러리는 *planner*, *monitor*, 그리고 *communicator*로 구성되는데, [그림 7]와 같은

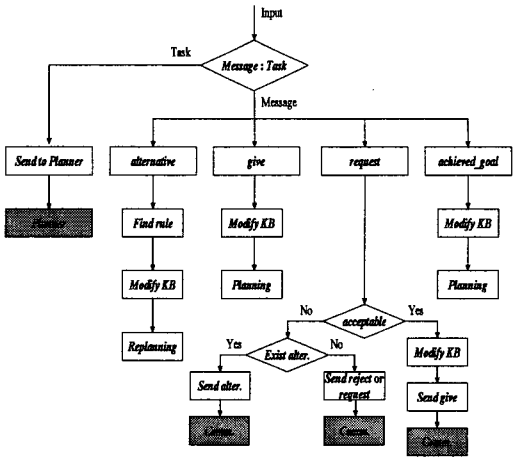
planner는 각각의 작업을 어떻게 해결할지를 결정하며, [그림 8]과 같은 monitor는 작업의 실행을 감시하고 메시지를 보낸 에이전트에게 결과를 보고하며, 마지막으로 [그림 9]와 같은 communicator는 소켓을 사용하여 다른 에이전트에게 메시지를 보내는 역할 그리고 수신한 메시지를 리다이렉팅하는 역할을 담당한다.



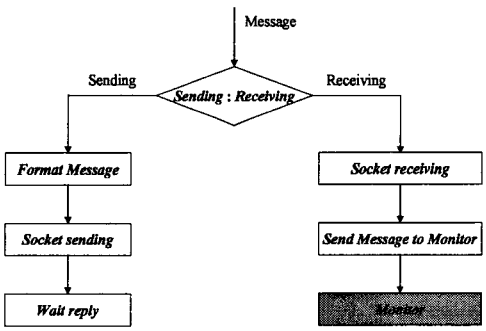
[그림 6] BDI 에이전트 아키텍처
[Fig. 6] BDI Agent Architecture



[그림 7] Planner의 구조
[Fig.7] Structure of Planner



[그림 8] Monitor의 구조
[Fig. 8] Structure of Monitor



[그림 9] Communicator의 구조
[Fig. 9] Structure of Communicator

4. 구현

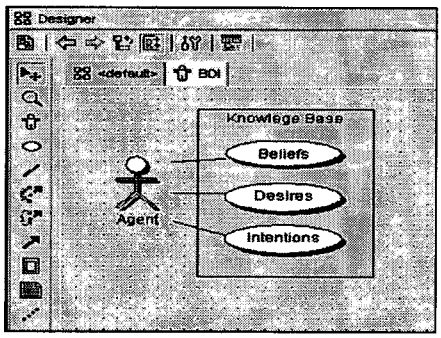
4.1 Agent와 UML 기반의 산출물

산출물이란 모델링 전반에서 생성되는 문서 형태의 출력물을 의미한다. CASE 도구의 산출물과 Agent을 UML 사양에는 없지만 일반적으로 많이 사용하는 요구사항을 반영하기 위해 후보 클래스의 생성과 관계를 설정하였으며, 분석과 설계에서는 UML 표준을 지원 다이어그램을 그대로 적용하였다[6]. 아울러 정확성을 위하여 UML 다이어그램의 메타모델이

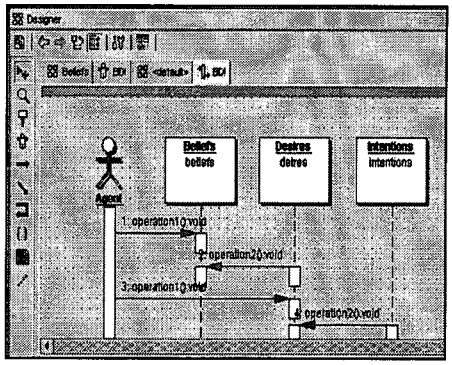
(metamodel)을 적용 시켰으며 이를 위하여 OMG에서 UML의 문법적 의미를 설명하기 위해 제시, 설계, 검증, 관계성 파악 하고 있는 메타모델을 이용 하였다.

아울러 본 연구에서 제안한 ACL을 자원이 한정된 BDI 에이전트 사이의 협상 시스템에 적용하여 응용 가능성을 검사한다. 실험은 Java 2 S나 1.4와 XSB Prolog 2.5를 지원하는 InterProlog 2.0.1로 수행 하였다.

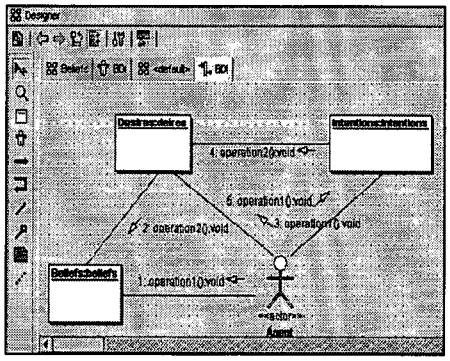
아래의 그림은 Agent관련 BDI위한 적용 예를 다이어그램으로 표현하기 위해서 4+1 관점에 따른 메타모델을 제시 하였다. 즉 개발 하고자 하는 시스템을 바라보는 시각에 따라 사용 사례(use case)관점, 논리(logical) 관점, 프로세스(process) 관점, 구현(implementation) 관점 등을 고려하였으며 [그림 10]과 같이 에이전트의 능력과 문제 분해를 위한 UseCase Diagram을 나타내었으며 [그림 14] planner는 각각의 작업을 어떻게 해결할지를 결정하기 위해서 각각의 작업을 객체의 타입인 클래스로 Diagram으로 표현 하였다. [그림 11]과 [그림 12]는 지식베이스의 관계를 순차와 협동으로 표현하여 상호 관계를 표시 하였다. 마지막으로 [그림 13]은 monitor로써 작업의 실행을 감시하고 메시지를 보낸 에이전트의 결과를 보고하는 과정을 활동(activity) 다이어그램으로 표현하며 활동들 사이의 전이와 결정점들을 나타내었다.



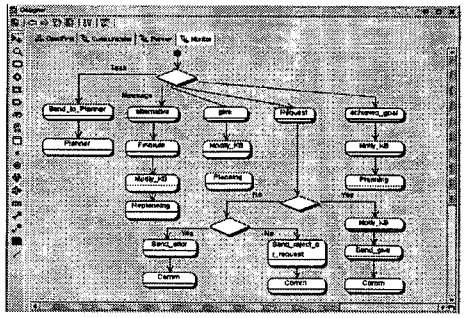
[그림 10] Use Case Diagram
[Fig.10] Use Case Diagram



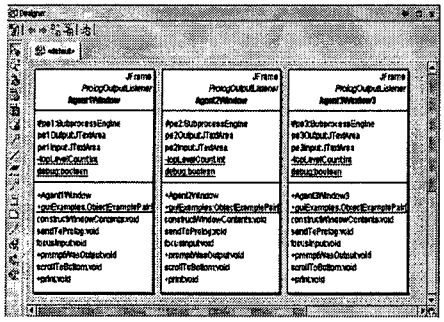
[그림 11] Sequence Diagram
[그림.11] Sequence Diagram



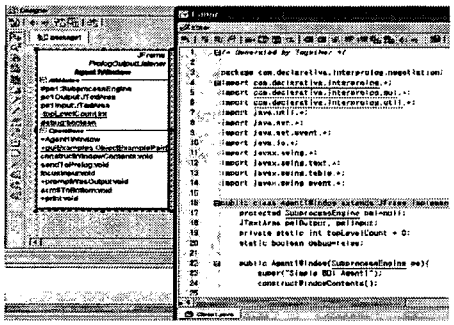
[그림 12] Collaboration Diagram
[Fig.12] Collaboration Diagram



[그림 13] Activity Diagram
[fig 13] Activity Diagram

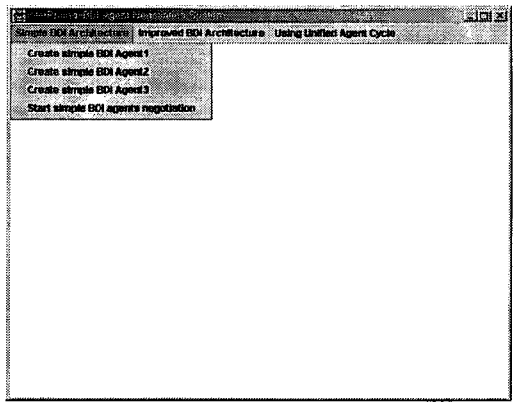


[그림 14] Class Diagram
[Fig.14] Class Diagram



[그림 15] Class Diagram 소스
[Fig.15] Class Diagram Sauce

아래의 그림들은 BDI 에이전트들을 생성하고 제안한 통신언어를 사용하여 서로 협상할 수 있게 한다. NegotiationWindow 클래스는 이와 같은 작업을 수행하며 [그림 16]와 같은 최상위 창을 표시한다. 또한 <표 1>는 NegotiationWindow 클래스에 포함된 메소드들의 역할을 간단히 소개하고 있다. [그림 16]에서 세 개의 BDI 에이전트들을 차례로 생성시킨 다음에 *Start simple BDI agents negotiation* 항목을 클릭 하여 에이전트들 사이의 협상을 진행시킨다. 한편 스트림 중심 혹은 메시지 중심의 여러 통신 메카니즘들이 존재하지만 여기서는 소켓을 이용한 버퍼링된 메시지 기반 통신 메카니즘을 사용한다. 즉, 통신과정은 잘 정의된 한계(boundary)를 가지는 메시지를 교환한다. <표 2>에서는 협상과정 동안 에이전트들이 주고받는 메시지들을 나열하였고, [그림17]은 에이전트 $a1$의 협상과정을 예로 보여주고 있다.



[그림 16] BDI 에이전트를 위한 협상 시스템
[Fig. 16] Negotiation System for BDI Agents

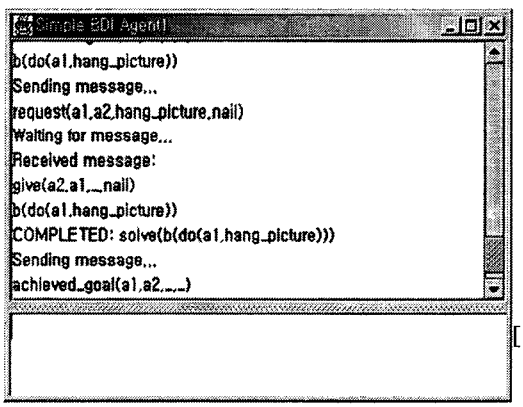


그림 17] 에이전트 $a1$의 협상과정
[Fig. 17] Negotiation Process of Agent $a1$

<표 1> NegotiationWindow 클래스의 메소드들
<Table 1> Methods of NegotiationWindow Class

Method names	Roles
negotiationWindow	Constructor to create the instance
constructWindowContents	Creates a 600 500 pane
constructMenu	Creates menubar and menu, handles action events
createSimpleBDIAgent1	Creates agent1's window with Prolog engine
createSimpleBDIAgent2	Creates agent2's window with Prolog engine
createSimpleBDIAgent3	Creates agent3's window with Prolog engine
addItemtoMenu	Adds items to menu
main	Displays system information and take an array as a default Prolog engine

<표 2> 협상과정 동안의전송 메시지들
<Table 2> Transmission Messages during Negotiation Processes

전송 방향	전송 메시지
a1 a2, a1 a3	*ask_if(a1, a2, b(have(a2, nail))), ask_if(a1, a3, b(have(a3, nail)))*
a2 a1	*inform(a2, a1, b(have(a2, nail)))*
a1 a2	*request(a1, a2, hang_picture, nail)*
a2 a1	*request(a2, a1, hang_mirror, hammer)*
a1 a2	*alternative(a1, a2, hang_mirror, [screwdriver, screw, mirror])*
a2 a1	*request(a2, a1, hang_mirror, screwdriver)*
a1 a2	*give(a1, a2, screwdriver)*
a2 a1	*request(a2, a1, hang_mirror, screw)*
a1 a2	*give(a1, a2, screw)*
a2 a1, a2 a3	*achieved_goal(a2, a1), achieved_goal(a2, a3)*
a3 a1	*request(a3, a1, hang_clock, hanger_nail)*
a1 a3	*give(a1, a3, hanger_nail)*
a3 a1, a3 a2	*achieved_goal(a3, a1), achieved_goal(a3, a2)*
a1 a2	*request(a1, a2, hang_picture, nail)*
a2 a1	*give(a2, a1, nail)*
a1 a2, a1 a3	*achieved_goal(a1, a2), achieved_goal(a1, a3)*

5 평가 및 결론

정보 시스템이 다양화, 분산화, 그리고 웹 기반 환경으로 변화하고 있기 때문에 시스템의 분석과 설계에 대한 중요성이 강조되고 있다. 본 연구에서는, 업무에 필요한 명세서 내용과 구현 결과의 일치성을 도모하여 관련 시스템이 발생시킬 수 있는 문제점들을 해결할 수 있는 방법을 제시하기 위해, 자동 생성 시스템을 실제의 응용 프로그램에 적용하고 이를 UML 관점에서 분석하였다. 또한, UML 산출물에 대한 Agent 응용과 그리고 UML 산출물에 대한 BDI 검증 방법을 개략적으로 소개하였다.

아울러 본 연구에서는 논리 프로그래밍 환경에서 자원이 한정된 BDI 에이전트들 사이의 협상을 위한 통신언어를 제안하였고, 에이전트가 가진 믿음, 소망, 그리고 의도를 메타술어로 간주하고 에이전트를 위한 간단한 협상 메케니즘을 소개하였다.

일반적으로 ACL은 에이전트의 구조와 특정 응용영역에 크게 의존하며, 이것에 따라 협상 프로토콜 역시 달라야 할 것이다. 특정 응용 에이전트 예를 들어, 사무자동화 협상 에이전트, 전자상거래 시스템 협상 에이전트, 혹은 경매 협상 에이전트를 위한 ACL 및 협상 프로토콜의 개발이 필요할 것이다. 또한 수신한 통신행위를 통해 다른 에이전트의 지식에 관해 가설적으로 추론할 수 있는 즉, 수신한 통신행위로부터 상대방 에이전트의 의도, 목표, 혹은 지식에 대해 추론할 수 있는 프레임워크의 개발이 필요하다. 이러한 추론 결과들은 에이전트들 사이의 협동, 경쟁, 혹은 목표 불일치 상황에서 협상의 중요한 증거로 작용할 것이다.

참고문헌

[1] 한현관, 이명진, "UML에 기초한 웹 어플리케이션 자동 생성 CASE 도구의 분석", 한국컴퓨터산업교육학회 논문집 3(12), 2002.

[2] B. Chaib-draa and F. Dignum, Trends in Agent Communication Language,

Computational Intelligence, 18(2), 200
[3] G. Booch, "*Object-Oriented Analysis and Design*", 2nd Edition, Benjamin/Cummings, 1994.
[4] G. Booch, J. Rumbaugh, and I. Jacobson, "*The Unified Modeling Language User Guide*", Addison-Wesley, 2001.
[5] I. Jacobson, "*Object-Oriented Software Engineering: A Use Case Driven Approach*", Addison-Wesley, 1999.
[6] Object Management Group (OMG), "*Unified Modeling Language Specification* Ver. 1.3", 1999.
[7] Together, http://www.togethersoft.com/.

한현관

1992년 경일대학교 정보전자학과 졸업(학사)
1993년 대구대학교 산업대학원 정보 관리학(공학 석사)
2003년 영남대학교 대학원 컴퓨터공학과(박사 수료)
2005년 현재 대구산업정보대학 겸임교수
관심분야 : UML, 인공지능, 에이전트 등

박재복

2005년 현재 경북과학대학 컴퓨터미디어학과 교수
관심분야 : 멀티미디어, 소프트웨어공학