소프트웨어 에이전트 기반의 모델링 방법에 대한 고찰

A Survey of Software Agent Based Modelling Method

김귀정
건양대학교

Kim Gui-Jung
KonYang University

요약

에이전트 지향 소프트웨어 공학은 가장 새롭게 대두되고 있는 소프트웨어 공학 분야 중 하나이다. 이 방법은 기존의 개발 방법과 비교하여 많은 장점을 가지고 있다. 그 중 소프트웨어 시스템의 활성 객체(active entity)를 에이전트로 하여금 고급 추상화 단계로 표현할 수 있도록 해준다. 본 논문은 에이전트 지향 소프트웨어에 대한 연구와 산업 현장에서의 적용을 개략적으로 알아보고자 한다.

Abstract

Agent-Oriented Software Engineering is the one of the most recent contributions to the field of Software Engineering. This method has many benefits compared to existing development approaches. In particular the ability to let agents represent high level abstraction of active entity in software system. This paper gives an overview of research for agent-oriented software and industrial applications.

I. 서 론

에이전트 지향 소프트웨어 공학은 소프트웨어 공학 연구 분야의 새로운 파라다임으로 인식되어 지고 있다[1]. 그러나 소프트웨어 산업 현장에서 새로운 파라다임으로 받아들여 지기 위해서는 사용하기 쉽고 강력한 방법론과 툴들이 개발되어야 한다.

그럼, 먼저 에이전트가 무엇인지에 대해 알아보도록 한다. 에이전트는 소프트웨어 에이전트(software agent) 또는 지능 에이전트(intelligent agent)라고 불린다. "지능(intelligent)"은 소프트웨어가 특정 형태의 행위를 수행할 수 있기 때문에 붙여진 것이고, "에이전트(agent)"는 다른 사람을 대신하여 행위를 할 수 있는 권한을 가진 사람을 의미한다. 소프트웨어 에이전트의 예를 보자. Microsoft Office의 동물 캐릭터 에이전트, 컴퓨터 바이러스, 컴퓨터 게임이나 시뮬레이션의 인공지능 플레이어, 그리고 Google과 같은 웹 스파이더 등이 있을 수 있다.

에이전트 지향 소프트웨어 공학이 비교적 새로운 영역이면서도 기존의 연구에 기반을 두기 때문에 비슷하게 연관되어지는 많이 용어들이 있다. 각 용어들의 정의와 그들 사이의 연관성은 다음과 같다. 에이전트지향 프로그래밍(agent-oriented programming:AOP)은 객체지향 프로그래밍(object-oriented programming: OOP)의 개선과 확장으로 볼 수 있다. "프로그래밍"이란 용어에서 알 수 있듯이 프로그래밍 언어와 유사한 개념을 가지고 있다. 에이전트지향 개발(agent-oriented development:AOD)은 객체지향 개발(object-oriented development:OOD)의 확장이다. "개발"이란 용어는 때때로 "프로그래밍"과 혼용하여 사용하지만, 일반적으로 요구사항 분석과 설계 그리고 프로그래밍에 이르는 모든 개발 과정을 포함한다. 에이전트 소프트웨어 공학(software engineering with agent), 에이전트 기반 소프트웨어 공학

(agent-based software engineering), 다중 에이전트 시스템 공학(multi-agent systems engineering), 그리고 에이전트지향 소프트웨어 공학(agent-oriented software engineering)은 의미적으로 모두 동일하다. 그러나, 다중 에이전트 시스템 공학은 방법론에 좀더 가까우며, 에이전트지향 소프트웨어 공학이 가장 많이 사용되는 용어이다. 에이전트 기반 컴퓨팅(agent-based computing)은 에이전트지향 소프트웨어 공학에 관련된 모든 이슈들을 설명하기 위해 적용될 수 있다.

본 논문에서는 에이전트 기반 시스템 개발을 위한 방법론의 발전 과정에 대한 전반적인 사항을 개략적으로 고찰하고자 한다. 소프트웨어 공학과 관련된 전반적인 고급 레벨의 방법론과 더불어 좀더 상세한 설계 방법론을 함께 알아보고자 한다.

Ⅱ. 에이전트지향 소프트웨어 공학

에이전트지향 소프트웨어 공학의 주요 목적은 방법론을 생성하고, 에이전트 기반 소프트웨어를 효율적으로 개발·유지보수 할 수 있는 툴을 만드는 것이다. 또한 소프트웨어는 융통성이 있어야 하며, 사용하기 쉽고, 범용적이어야 하며 높은 질을 가지고 있어야 한다. 본 장에서는 에이전트에 대해 자세히 살펴본다.

먼저, 에이전트가 다른 객체와 차이점은 무엇인가? 에이전트지향 프로그램은 객체지향 프로그램의 확장으로 볼 수 있으며, 또한 구조적 프로그램의 계승이라고도 볼 수 있다. 객체지향 프로그램에서 주요 엔티티는 객체이다. 객체는 자료구조의 논리적 결합과 그들 사이의 함수이다. 객체는 실세계에서의 수동 엔티티를 추상화하기 위해 사용된다. 반면 에이전트는 활성 엔티티를 효율적으로 추상화할 수 있다는 점에서 객체를 발전시킨 것이라고 볼 수 있다. 에이전트가 객체와 유사한 것은 사실이나 에이전트는 믿음, 약속, 책임 등과 같은 비물질적 컴포넌트를 표현할 수 있는 구조를 가지고 있다. 또한 에이전트지향 프로그램이 객체지향 프로그램과 다른 중요한 차이점은 객체는 외부로부터 제어되지만, 에이전트는 외부로부터 직접적으로 제어될 수 없는 독립된 행위를 가지고 있다.

그럼, 에이전트는 모든 소프트웨어 문제를 해결할 수 있을까? 이 질문에 대해서는 그간 에이전트지향 소프트웨어 엔지니어링의 능력에 대해 지나치게 낙관적인 연구가 이루어져 왔으며, 이 점이 위험요인이 될 수 있는 것은 사실이다. 에이전트지향 소프트웨어 엔지니어링의 잠재적 함정에 대해 논의한 연구를 살펴보면, 5개의 그룹으로 나누어 잠재적 함정을 분류하였다[2]. 전략적(political) 함정, 개념적(conceptual) 함정, 분석(analysis) 함정, 설계(design) 함정, 에이전트 레벨(agent-level) 함정이 그것이다. 전략적 함정은 에이전트를 지나치게 강조하거나 에이전트를 유일한 해결책으로 인식할 때 생길 수 있다. 개념적 함정은 개발자가 에이전트는 소프트웨어(특히 멀티스레드 소프트웨어)라는 사실을 간과할 때 발생할 수 있다. 분석과 설계 함정은 개발자가 에이전트 이외의 다른 소프트웨어 공학 방법론 등을 무시할 때 발생할 수 있으며, 에이전트 레벨함정은 개발자가 에이전트 시스템에 지나치게 많거나 너무 적은 인공 지능을 사용하고자 할 때 발생할 수 있다. 이러한 함정들을 정확히 인식할 때 비로소 올바른 에이전트지향 소프트웨어 엔지니어링을 이용할 수 있다는 사실을 인지해야 할 것이다.

Ⅲ. 에이전트기반 시스템 개발 방법론

1. Gaia 방법론

에이전트지향 분석·설계를 위한 Gaia 방법론은 에이전트 구조와 같은 최하위 레벨에서부터 에이전트 조직 구조와 같은 대규모 레벨을 모두 지원하는 일반적인 방법론이다[3]. 기존의 방법론은 에이전트의 독립적 특성과 문제 해결 능력을 표현할 수 없었으며, 또한 에이전트가 조직을 구성하고 상호작용하

는 방법을 모델링 할 수 없었다. 이에 대한 해결책을 제시한 것이 Gaia 방법론이다. Gaia 방법론을 사용함으로써 소프트웨어 설계자는 시스템 요구사항을 바탕으로 구현이 용이한 효율적인 설계를 할 수 있게 되었다.

Gaia 분석 과정에서 첫 번째 단계는 시스템에서의 역할(role)을 정의하는 것이다. 두 번째 단계는 정의한 역할 사이의 상호작용(interaction)을 모델링한다.. 역할은 책임(responsibility), 허용(permission), 활동(activity) 그리고 프로토콜(protocol)의 4개 속성으로 구성된다. 책임은 생명(liveness) 속성과 안전(safety) 속성을 나뉘는데, 생명 속성은 시스템에 이로운 것을 추가하는 역할을 하며 안전속성은 시스템에 해로운 일이 발생하는 것으로부터 시스템을 보호하는 역할을 한다. 허용은 접근이 허가되는 정보를 나타내는 역할이다. 활동은 다른 역할과의 상호작용 없이 수행되는 역할을 말하며, 프로토콜은 상호작용의 특별한 패턴이다. Gaia는 역할과 역할이 가지고 있는 속성을 표현하기 위한 형식화된 오퍼레이터와 템플릿을 가지고 있으며, 또한 상호작용을 표현하기 위한 스키마도 가지고 있다.

Gaia 설계 과정의 첫 번째 단계는 역할을 에이전트 타입과 사상시키고 각 타입에 대한 에이전트 객체를 생성한다. 두 번째 단계는 에이전트의 역할을 충족시키기 위해 필요한 서비스 모델을 결정한다. 마지막으로 에이전트 간의 상호작용을 표현하기 위한 전달 모델을 생성한다.

Gaia 방법은 인터넷 애플리케이션과 같은 개방적이고 예측할 수 없는 도메인에서는 사용하기 적합하지 않으며, 반면에 비개방적 도메인 에이전트 시스템을 개발하기 위해서 좋은 방법론이 될 수 있다.

2. 멀티에이전트 시스템 공학 방법론

멀티에이전트 시스템 공학 방법론은 범용성과 지원되는 응용 도메인의 관점에서는 Gaia 방법론과 유사하지만, 멀티에이전트 시스템 툴을 통한 자동 코드 생성이 가능하다는 점에서 더 진보된 방법이라 할 수 있다[4]. 멀티에이전트 시스템 공학 방법론은 에이전트기반 시스템을 생성하기 위한 강력한 툴과 입증된 방법론의 부재에서 기인한다. 멀티에이전트 시스템 공학 방법론의 목적은 초기 시스템 상세화에서부터 구현에 이르기까지 설계자를 단계별로 효율적으로 유도하는데 있다. 이 방법론의 도메인 제약은 Gaia와 비슷하며, 더불어 에이전트 상호작용은 1:1만 가능하다.

멀티에이전트 시스템 공학 방법론은 7 단계로 진행된다.

◦1 단계 : 목표 수집

초기 시스템 상세화를 시스템 목표의 구조적 계층관계로 변환한다. 이것은 초기 시스템 상세화 요구사항을 기반으로 목표를 판별하고, 구조적이고 순서화된 계층관계에서의 중요도에 따라 목표를 정렬함으로써 이루어진다.

◦2 단계 : 유즈케이스 적용

초기 시스템 상세화에 따라 유즈케이스와 시퀀스 다이어그램을 생성한다. 유즈케이스는 시스템과 그 내부에 있는 다양한 역할들 사이의 논리적 상호작용을 표현한다. 시퀀스 다이어그램은 역할들 사이에 연결되어야 하는 최소한의 메시지 수를 결정한다.

◦3 단계 : 역할 정제

1 단계에서 정의된 목표를 위한 역할을 생성한다. 일반적으로 각 목표는 하나의 역할로 표현되지만, 때로는 연관된 여러 개의 목표가 하나의 역할로 매핑될 수도 있다.

◦4 단계 : 에이전트 클래스 생성

역할은 에이전트 클래스 다이어그램을 사용하여 에이전트 클래스로 사상된다. 에이전트 클래스 다이어그램은 객체 다이어그램과 유사하다.

• 5 단계 : 상호작용 구축

에이전트 상호작용을 정의하는 상태 다이어그램을 이용하여 관계 프로토콜을 정의한다.

• 6 단계 : 에이전트 클래스 조립

에이전트 클래스의 내부 기능을 정의한다. 선택된 기능은 서로 다른 5가지 타입의 에이전트 구조에 기반을 둔다. BDI(Belief-Desire-Intention), 반작용, 계획 수립, 지식 기반, 사용자 정의 구조가 그것이다.

• 7 단계 : 시스템 설계

에이전트 클래스에 기반 한 실제 에이전트 인스턴스를 생성한다. 최종 결과는 배치 다이어그램으로 표현된다.

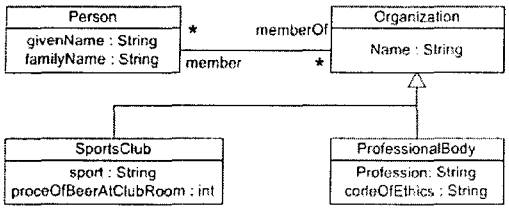
Ⅳ. UML을 이용한 설계

UML은 객체지향 분석과 설계를 위한 그래픽 표현과 언어이다. 객체지향 모델링 파라다임은 소프트웨어 산업현장에서의 주요 기술이 되어 왔으며, 객체지향 모델링은 인간의 직관적인 모델을 가장 잘 표현할 수 있는 방법이라고 널리 인식되어져 오고 있다. UML은 OMG[5](Object Management Group)에 의해 표준화되었으며, 많은 도구들이 UML을 사용하여 모델을 생성하고 편집할 수 있도록 지원하고 있다. UML은 산업현장에서 널리 사용되고 있으며 또한 빠른 속도로 확산되고 있는 현실에 비추어 볼 때, 모델링 언어로써 UML을 이용하는 것은 매우 타당한 이유라 하겠다. 그러나, UML은 멀티에이전트 시스템 모델링을 위해 전통적으로 사용되어 왔던 논리 기반 형식과 커다란 차이가 있다. UML은 서로 연관되어 질수 있는 관계와 사용자 모델에서 나타날 수 있는 구조의 형식을 정의하는 메타모델과 도형적 구문으로써 표현된다. 표준 XML 기반 형식인 XMI(XML Model Interchange format)는 모델이 UML 메타모델의 인스턴스로 인코딩 될 수 있도록 해준다. 전통적인 지식 표현 방법과 이러한 차이가 있음에도 불구하고 UML 모델이 선언적 지식 표현 파라다임이 될 수 있는 여러 가지 특징들이 있다.

- UML로 표현된 지식은 표준 그래픽 표현 방법을 이용하여 인간이 직접적으로 이해할 수 있다.
- UML로 표현된 지식은 객체지향 모델링의 특성인 모듈화로 구성되기 때문에 변경이 용이하다.
- 새로운 지식은 추론에 의해 UML 모델로부터 얻어질 수 있다. 특히, UML은 관계 제약어인 OCL(object Constraint Language)을 가지고 있는데, OCL은 새로운 모델 요소를 정의하고, 임의의 제약조건을 가능한 모델 인스턴스에 부여하는 역할을 한다.

이러한 관점에서 UML은 멀티에이전트 시스템을 표현하는데 있어서 적당한 방법이 될 수 있다. 특히, UML 클래스 다이어그램은 클래스와 속성 그리고 그들 사이의 관계를 정의하는데 풍부한 표현력을 제공한다. 그림 1은 사람과 조직체를 표현하는 클래스와 그들 사이의 관계를 클래스 다이어그램으로 나타낸 것이다. 「Organization」 클래스는 「SportsClub」과 「ProfessionalBody」 두 개의 상세화된 클래스를 가지고 있다. UML 클래스 다이어그램의 그래픽 의미는 다음과 같다.

- 사각형은 클래스를 나타낸다. : 클래스 이름은 위쪽에 쓴다.(추상 클래스일 경우에는 이탤릭체를 사용한다) 속성은 아래 부분에 표현한다.
- 클래스 사이의 선은 연관관계를 나타낸다. : 관계는 "역할이름"을 가질 수 있다. 얼마나 많은 객체들이 관계에 참여하는지를 나타내기 위해 숫자를 사용한다. "*"은 0또는 그 이상을 나타낸다.
- 화살표를 가진 선은 일반화를 나타낸다. 화살표 머리 부분의 클래스가 더 일반화된 개념임을 의미한다.



▶▶ 그림 1. UML 클래스 다이어그램

V. 결 론

본 논문에서는 에이전트 기반 시스템 개발을 위한 방법론의 발전 과정에 대한 전반적인 사항을 개괄적으로 고찰하였다. 또한 에이전트 시스템에서 UML을 사용한다면 에이전트 기반 시스템 개념의 이해를 돕는데 매우 효과적일 것이다. 이에 본 연구는 멀티에이전트 시스템 모델링을 위한 UML 사용의 가능성을 살펴보았다. 향후에는 방법론에 대한 실제적인 테스팅과 실험을 통해 좀더 완전한 분석이 이루어져야 할 것이다.

▌참 고 문 헌▐

[1] Lind J., "Issue in Agent-Oriented Software Engineering," The First International Workshop on Agent-Oriented Software Engineering (AOSE-2000), 2000.

[2] Wooldridge M. J. and Jennings N. R., "Pitfalls of agent-oriented development," Proceedings of the second international conference on Autonomous agents, pp.385-391, 1998.

[3] Wooldridge M. J., Jennings N. R. and Kinny D., "The Gaia methodology for agent-oriented analysis and design," Autonomous Agents and Multi-Agent Systems, Vol.3, No.3, pp.285-312, Sep., 2000.

[4] Wood M. F. and DeLoach S. A., "An Overview of the Multiagent Systems Engineering Methodology," The First International Workshop on Agent-Oriented Software Engineering(AOSE-2000), 2000.

[5] http://www.ontoknowledge.org/oil, 2000.