한국지능정보시스템학회논문지 제10권 제3호 2004년 12월(pp.91~103)

시맨틱 웹 기반 시스템을 위한 에이전트 응용 프레임웍

이재호
서울시립대학교 전자전기컴퓨터공학부
(jaeho@uos.ac.kr)

다중 에이전트 시스템을 바탕으로 구축되는 시맨틱 웹 응용 프로그램은 에이전트 시스템이 제공하는 적절한 수준의 추상화에서 비롯되는 융통성을 유지하면서도 개발 및 운용의 효율성이 요구된다. 본 연구에서는 에이전트 수준의 추상화를 BDI 에이전트 구조를 기반으로 제공하면서 Java 기반 시스템의 효율성을 갖춘 새로운 에이전트 응용 프레임웍인 VivAce(Vivid Agent Computing Environment)를 소개하고 그 효율성을 대규모 에이전트 기반 시뮬레이션을 통하여 보인다. Vivid 에이전트는 소프트웨어에 의해 제어되는 시스템으로서 지식(knowledge), 지각(perception), 임무(task), 의도(intention)를 중심으로 상태(state)를 표현하며 활동(action)과 반응규칙(reaction rule)으로 행위(behavior)를 나타낸다. 본 논문에서는 먼저 에이전트 응용 프레임웍에 필요한 요소를 제시하고 이에 관련된 VivAce의 기능과 특징 및 이를 이용한 실험 결과를 제시한다.

논문접수일 : 2004년 10월 게재확정일 : 2004년 12월 교신저자 : 이재호

1. 서론

에이전트는 자신의 설계 목적을 만족하기 위하여 다른 에이전트와 상호작용할 수 있는 능력을 가진 자율적 시스템으로서 새로운 소프트웨어 공학 방법론으로 주목 받고 있다[9]. 에이전트 기반 시스템은 시맨틱 웹 응용 프로그램을 비롯한 비즈니스 프로세스 모델링, 원격통신, 컴퓨터 게임, 지능형 로봇, 군사 모의 실험 등의 광범위한 분야에 적용되고 있다. 에이전트 기반 방법론이 이러한 광범위한 분야에서 관심의 대상이 되어 모델링과 모의 실험에 지대한 영향을 발휘하는 것은 복합 시스템을 설계하고 가상적으로 시뮬레이션을 할 수 있는 다중 에이전트 시스템의 자연스런 능력에서 비롯된다.

서로 상호작용을 하는 다수의 자율적 에이전트로 구성되어있는 다중 에이전트 시스템은 다른 에이전트들뿐만 아니라 외부에서 발생하는 사건과 활동 결과의 불확실성까지도 고려하여 역할을 다해야 하는 복합적, 역동적이며 비결정적인 실제 환경을 모델링하고 시뮬레이션을 하기 위한 매우 효과적인 수단을 제공한다. 에이전트는 여러 쓰레드(thread)의 동시 제어 능력을 가진 독립적 개체로서 메시지 전달(message passing)과 같은 수단을 이용하여 주변 환경 및 다른 에이전트와 상호 소통 하는 단위로 볼 수 있다[18]. 특히 BDI 에이전트는 믿음(Belief), 욕구(Desire), 의도(Intention)와 같은 인간적 특징을 바탕으로 한 에이전트 모

델을 가지고 있다[14][15]. BDI 에이전트 구조는 UM-PRS [10], JAM [7], dMARS [4]와 같은 잘 알려진 에이전트 구조의 근간이 되고 있다. BDI 에이전트 구조는 믿음 베이스 (belief base), 현재 지닌 욕구(desire) 또는 목적(goal) 모음, 계획(plan) 모음, 의도구조(intention structure) 등으로 이루어진다. 계획 모음은 특정 상황에서 주어진 목적을 달성하기 위해서 수행하여야 할 일련의 활동과 검사 과정을 기술한 계획들의 모음이다. 의도구조는 실행을 위하여 이미 선택된 계획을 바탕으로 실행에 관련된 정보를 유지한다. 추론기(interpreter)는 이 들 구성 요소들을 관장하여 시스템의 현재 믿음과 목적을 기반으로 적합한 계획을 선택하여 의도구조에 기록하여 이를 실행하고 실행 상태를 감시한다.[17].

본 논문에서는 VivAce라고 명명한 에이전트 시스템을 제시한다. VivAce는 Java 프로그래밍 언어로 에이전트를 구현할 수 있어 Java의 실행 효율성을 지니면서 BDI 에이전트 구조가 갖는 에이전트 수준의 추상화의 장점을 동시에 갖고 있다. VivAce는 Vivid Agent Computing Environment의 머리 글자로 이루어진 약어로서 Michael Schroeder와 Gerd Wagner [17]가 제안한 Vivid 에이전트의 특징을 가진다. Vivid 에이전트는 소프트웨어에 의해 제어되는 시스템으로서 지식(knowledge), 지각(perception), 임무(task), 의도(intention)를 중심으로 상태(state)를 표현하며 활동(action)과 반응규칙(reaction rule)으로 행위(behavior)를 나타낸다. Vivid 에이전트의 기본 기능은 갱신과 추론 연산을 포함한 지식 시스템 및 지각 시스템과 사건(event)에 반응하고 활동하기 위한 체계 및 계획을 생성하고 수행하는 능력에서 비롯된다. 반응(reaction)은 현재의 지식 상태와 관계없는 즉각적인 활동이지만 에이전트의 숙고(deliberation)의 결과에 의해 영향을 받을 수도 있다. 사건은 에이전트가 제어할 수 없는 대상이며 이러한 사건에 의해서 반응이 유발된다 [17].

본 연구는 Vivid 에이전트의 일반 개념을 채택하고 BDI 에이전트 모델을 사용하여 시맨틱 웹을 비롯한 대규모 동시성을 지닌 다중 에이전트 시스템에 적합한 에이전트 응용 프레임웍을 제시한다. 이러한 에이전트 응용 프레임웍의 기능은 직접 제작한 단순 시뮬레이터와 로봇 게임을 위한 Robocode 시뮬레이터에 자율적 VivAce 에이전트를 적용한 실험 결과로 보인다.

에이전트 모델은 에이전트의 내부와 외부 사이의 개념적 경계선을 제공한다. 경계선의 내부에서는 다른 에이전트들로부터 독립성과 자율성을 확보하는데 필요한 자유 공간을 제공하며 경계선의 외부는 다른 에이전트들에게 충분한 수준의 추상화를 제공함으로써 개체 간의 복잡한 상호작용을 에이전트간의 협약과 규약(protocol)으로 단순화시키는 효과가 있다. 에이전트 개념에서 비롯되는 모듈화로 인하여 에이전트들을 독립적으로 개발하여 이를 시맨틱 웹 응용 프로그램과 같은 대규모 분산 병렬 시스템에 통합하는 일이 용이하게 된다. 에이전트 기반 시스템에서는 객체지향 방법론에서와 같은 메소드(method)의 직접적 호출 대신에 미리 정해진 목적기반 상호작용(goal-based interaction)을 사용하기 때문에 이러한 시스템 통합 과정에서 시스템의 변경을 필요로 하지 않게 된다. 목적기반 상호작용은 또한 시맨틱 웹 응용 프로그램이나 시뮬레이터에 적용하는데 있어 추가적인 장점을 제공한다. 객체지향 방법에서 사용되는 메소드 호출은 정형화된 결과를 만들어 낸다. 반면에 에이전트 기반 시스템에서는 추구하고자 하는 효과를 각각의 에이전트에게 목적의 형태로 부여하고 에이전트는 역동적으로 변화하는

상황에 적응하며 목적에 적합한 일련의 활동들을 자유롭게 선택하여 실행할 수 있게 됨으로써 역동적 환경과 이에 대한 대응 결과를 용이하게 사실적으로 모델링 할 수 있게 된다.

다음 2절에서는 먼저 BDI 에이전트 모델과 이를 구현한 대표적 에이전트 구조인 JAM을 소개하여 VivAce 개발 배경을 설명하고, 3절에서는 VivAce 구조를 기술한다. 4절에서는 관련 연구를 간략히 소개한 후 5절에서 결론과 향후 연구 방향을 소개한다.

2. BDI 모델과 JAM

믿음-욕구-의도 (Belief-Desire-Intention: BDI) 에이전트 모델은 목적과 계획 기반의 실용적 추론 구조이다. 에이전트는 사용자가 제공한 계획 모음에서 상황과 목적에 적합한 계획을 선택하여 실행함으로써 목적을 달성한다. BDI 에이전트는 따라서 사전에 계획을 미리 작성할 수 있는 복잡한 환경을 위한 실시간 추론 및 실행 시스템이라 할 수 있다. BDI 모델은 다음과 같은 컴포넌트로 구성된다.

- **믿음(Belief)**: 세상에 대한 지식과 모델을 일컫는다.
- **목적(Goal)**: 도달하고자 하는 상태로서 서로 모순적인 목적도 허용된다.
- **의도(Intention)**: 특정 상태에 도달하고자 한 다짐(commitment)으로 현재 실행 중인 계획으로 표현된다.
- **계획(Plan)**: 절차적 지식(procedural knowledge)으로서 계획의 실행이 적합한 상황 및 목적을 나타내고 그러한 상황과 목적이 주어졌을 때 행할 일련의 활동 등을 기술한다. 계획의 *초기조건(precondition)*은 계획의 실행이 적합한 상황을 기술하고 *본체(body)*에는 실행할 일련의 활동들을 기술한다. 또한 계획의 실행 중 지속적으로 만족되어야 할 조건인 *지속조건(context)*을 갖고 있다. 이밖에 계획이 성공적으로 수행되었을 경우와 실패한 경우에 각각 실행할 추가적인 일련의 활동들을 지정할 수 있다. 계획의 본체에 지정한 활동은 믿음이나 목적과 같은 BDI 에이전트의 다른 구성 요소들에 대한 연산을 포함한다. 특히 에이전트 자신이 주목적을 달성하기 위하여 만든 부가적인 목적인 부가목적(subgoal)을 만드는 활동도 가능하다.

BDI 에이전트 구조에서의 실행은 다음과 같은 주기로 이루어진다 [2]:

1. 세상과 에이전트 내부의 상태를 관찰하고 이에 따라 사건 큐(queue)를 갱신한다.
2. 사건 큐(queue)에 기록된 사건들과 일치하는 사건을 유발(trigger) 조건으로 가지고 있으며 초기조건이 만족되는 계획들을 선택하여 이를 위한 실행계획(instance)을 생성한다.
3. 생성된 실행계획 중 하나를 실제 실행을 위하여 선택한다.
4. 선택된 실행계획이 부가목적(subgoal)을 위한 경우에는 기존의 의도구조에, 새로운 목적을 위한 경우에는 새롭게 만든 의도구조 스택(stack)에 넣는다.
5. 의도구조 스택을 선택하고 이의 가장 상위 실행계획을 취하여 계획의 본체에 지정된 다음 실행 단계를 수행한다. 이 단계가 활동(action)이면 바로 실행하고 부가목적(subgoal)이면 사건 큐에 넣는다.

JAM은 위와 같은 BDI 모델은 갖는 Procedural Reasoning System (PRS)[6]에서 제시한 개념적 프레임웍을 Java로 구현한 것이다. JAM은 BDI 에이전트 구조로서 BDI의 구성요소를 가지고 있다. JAM의 계획(plan)은 BDI 모델의 계획과 마찬가지로 계획이 실행을 시작하기 위해서 만족되어야 할 초기 조건을 지정하는 초기조건(precondition)과 실행 중에 지속적으로 만족되어야 할 지속조건(context)을 가지고 있다. 초기조건은 JAM의 추론기에 의해서 주어진 목적과 상황에 적합한 계획을 선정하는 과정에서 한 번만 검사된다. 이를 검사하는 과정에서 부가 효과로 계획이 가지고 있는 변수에 특정 값이 부여될 수 있다. 이때 부여된 변수 값은 실행계획 안에서 유효하며 부가목적(subgoal) 생성 과정을 포함한 본체 내 활동에서 이용할 수 있다. 반면 지속조건은 계획의 초기 선택뿐만 아니라 계획의 본체에 지정된 활동을 수행하는 전 과정 동안 만족되어야 할 조건으로서 한 개의 *최소단위활동*(atomic action)가 실행될 때마다 추론기는 지속조건을 검사하여 계획의 지속 여부를 결정한다. 지속조건은 역동적인 환경의 변화에 효율적으로 대응하기 위한 적절한 수단이 된다. 실행 중 지속조건이 만족되지 않게 되면 현재 계획과 이 계획에서 파생된 부가목적에 의해 선택되어 실행 중인 모든 계획들의 실행이 취소되고 변화한 새로운 상황에 적합한 계획을 새롭게 찾게 된다. 이러한 실행의 전체 과정을 통한 연속적인 상황 감시 기능이 바로 JAM과 같은 에이전트 구조가 환경의 변화에 바로 대응하는 반응형 에이전트의 구현을 가능하게 한다.

에이전트가 지각한 정보와 내부 상태를 비롯한 지식을 술어(predicate)로 표현한 믿음은 데이터베이스에 저장된다. JAM은 이러한 믿음 데이터베이스를 가지고 선택된 계획을 구성하는 활동들을 실행함으로써 다른 시스템을 포함한 환경과 상호작용을 하게 된다. 전통적 숙고형(deliberative) 계획수립(planning) 시스템과는 달리 JAM은 지속적으로 세상에 대한 지식의 변화를 반영하여 결정을 내리며 이에 따라 목적지향성을 유지하면서도 예상치 못하게 변화하는 환경에 역동적으로 적절한 활동을 선택할 수 있는 것이다.

3. VivAce의 구조

JAM 에이전트의 근간을 이루는 BDI 모델과 앞서 설명한 실행 주기의 특성에서 살펴본 바와 같이 JAM 에이전트는 환경의 변화에 반응하는 반응형 에이전트의 성격을 갖는다. 이러한 성격은 환경의 변화에 신속히 대응해야 하는 실시간 시스템을 구성하기에 적합하다. JAM은 반응형 특성을 가진 PRS의 개념적 프레임웍을 구현한 에이전트 구조로서 시맨틱 웹 응용 프로그램을 위한 다중 에이전트 시스템이나 대규모 시뮬레이션 시스템 등에서 요구되는 다음과 같은 특성을 갖는다.

- **실시간 실행**: 에이전트는 주변 환경의 변화에 반응하며 환경의 변화에 대하여 예측 범위 내의 시간에 반응할 만큼 충분히 빠르다.
- **개입(interrupt) 가능한 실행**: 에이전트는 현재 실행 중인 활동을 잠시 멈추고 현재 활동보다 긴급하거나 중요한 활동으로 무리없이 전환할 수 있다.
- **다중 주의력 유지**: 다수의 에이전트 및 주변 환경과 상호작용을 원활하게 하기 위해서는 다중의 주의력을 동시에 유지할 수 있어야 한다. 이러한 능력은 에이전트의 활동에 개입(interrupt)할 수 있을 뿐만 아니라 필요에 따라

재개(resume)할 수 있어야 한다

- **계층적 계획 세분화(refinement) 및 수정(revision)**: 에이전트는 고수준으로 표현된 작업을 점진적으로 저수준의 작아진 단위의 작업으로 분해하여 세분화할 수 있어야 한다. 일반적으로 목적(goal)으로 표현된 고수준의 작업은 주어진 상황에 적합한 저수준의 부가목적(subgoal)의 형태로 세분되며 이 또한 더욱 저수준의 부가목적으로 점진적으로 세분되어 최종적으로 직접적인 활동에 이르게 된다.
- **의도적 행위**: 에이전트는 목적을 이루기 위해서 체계적 절차에 의해 일관된 방식으로 의도적으로 작업한다.
- **규정된 전략 고수**: 에이전트는 사용자나 다른 에이전트 관점에서 보기에 모순되지 않는 행위를 보인다.
- **목적 지향(goal-driven) 및 데이터 구동형(data-driven) 행위**: 에이전트는 목적 지향적이어서 도달하고자 하는 상태를 고수준으로 표현한 목적에 따라 행위를 결정하는 목적 지향성을 갖는다. 또한 저수준의 데이터 값의 변화에 의하여 행위가 결정되는 데이터 구동형 행위도 동시에 가진다.

위에 나열한 다중 에이전트 시스템의 특징들은 모든 시스템에 요구되는 절대적 필수 조건을 말하는 것은 아니며 공통적으로 요구되는 부분적인 특징들을 나열한 것이다. JAM은 최소한 위에 나열한 특징들을 만족시키는 에이전트 구조이다. 그렇지만 대규모 다중 에이전트 기반 시뮬레이션 시스템이나 시맨틱 웹 기반 웹 응용 시스템을 JAM과 에이전트 구조로 구현하는데는 다음과 같은 한계가 따른다.

1. BDI 기반 에이전트 시스템에 필요한 계획은 기본적으로 텍스트 기반으로 이루어지며 이러한 텍스트로 이루어진 계획들은 파싱 과정을 통하여 시스템이 이해하고 이를 실행한다. 이때 계획의 본체에 지정된 *기본활동(primitive action)*들은 일반 프로그래밍 언어를 써서 먼저 지정된 후에 컴파일되어 기본활동으로 정의된다. 이러한 정의 과정에는 매개변수를 전달하고 이를 다시 해석하는 추가적인 노력이 필요하다. 특히 기본활동에 필요한 기능을 사용자가 이미 완성된 프로그램으로 가지고 있는 경우 이를 다시 에이전트 시스템에 통합하기 위해서 포장(wrapping)하는 수고를 들여야 한다.

2. JAM과 같은 에이전트 시스템의 구성 요소인 계획의 본체에 지정된 활동들은 일련의 기본활동들을 순차적으로 나열하여 연속된 활동으로 지정하거나, 일반 프로그래밍 언어에서 보는 것과 같은 구성자(constructor)를 써서 조건문이나 반복문 형태의 활동으로 지정된다. 이때 기본활동은 시작되면 종료(성공이나 실패)시까지 개입없이 수행되는 최소단위활동(atomic action)이다. 따라서 사용자가 기본활동의 규모를 지나치게 크게 만들면 추론기가 상황을 판단하고 필요에 의해 보다 중요하거나 긴급한 목적에 부합되는 다른 계획으로 전환하는데 문제가 발생하게 된다. 이러한 상황은 기본활동의 적절한 규모에 대한 심각한 고려없이 기존의 프로그램을 기본활동으로 변환하는 과정에서 종종 발생한다. 경우에 따라서는 소스 없이 이진코드로 제공된 라이브러리를 이용할 경우 이러한 상황을 피할 수 없게 된다. 예를 들어 카메라를 이용한 시각정보 처리 라이브러리를 이용하여 특정 개체를 인식하는 과정 전체가

이미 컴파일된 하나의 함수로 주어져 이를 기본활동으로 만들게 되면 인식이 끝날 때까지는 중간에 발생한 긴급한 일에 대한 대처를 할 수 없게 된다.

3. 계획의 지속조건(context)은 계획이 실행되는 전 과정을 통하여 만족되어야 할 조건이다. 실행 중인 계획의 실행을 지속하기 위해선 계획이 실행되는 동안 반복해서 지속조건을 검사하여 환경의 변화에 의해 지속조건이 더 이상 만족되지 않게 되었는지 확인하여야 한다. 이렇게 연속적으로 지속조건을 검사하는 것은 에이전트가 변화하는 환경에 반응적으로 대응하기 위해서는 불가피한 과정이다. JAM의 경우에는 하나의 기본활동 실행 시마다 현재의 계획과 현재의 계획을 선택하게 만든 목적을 만들어낸 상위 계획들의 지속조건을 모두 검사한다. 다시 말해 지속조건들을 매 실행 주기마다 폴링(polling)을 통하여 검사한다. 이러한 지속조건 검사를 매 주기마다 하는 대신에 환경의 변화에 의하여 조건이 변화하게 될 때만 수행할 수 있다면 지속조건 검사에 수반되는 오버헤드를 크게 줄일 수 있을 것이다.

VivAce는 위에 분석한 문제점들을 해결하는 방안을 제시한다. 해결을 위한 가장 첫번째 단계는 에이전트의 계획을 Java 객체로 직접 표현하는 것이다. 이러한 방안은 앞서 제시한 문제 1을 해결한다. VivAce는 그림1에 보인바와 같이 계획을 표현하기 위한 Java 추상 클래스를 제공한다. 사용자는 이를 확장하여 preCondition( )으로 표현된 초기조건과 context( )로 표현된 지속조건 및 body( )로 표현된 계획의 본체를 구현하여 새로운 계획을 만든다.

```
abstract public class Plan {
    abstract public boolean preCondition( );
    abstract public boolean context( );
    abstract public void body( );
}
```

[그림 1] Java 추상 계획 클래스

preCondition( )과 context( )는 내부에 임의의 조건을 구현하여 다만 진위 여부만을 결과로 반환하면 된다. body( ) 또한 임의의 Java 코드로 구현이 가능하다. VivAce는 BDI 에이전트에 필요한 구성 요소들을 미리 구현하여 이를 활용하기 위해 잘 정의된 응용프로그램 인터페이스(API)를 제공하는 역할을 한다. 그림 2 예제 에이전트 계획는 다음 절에서 소개할 시뮬레이터에 실제로 사용된 에이전트 구현 코드의 일부를 이해를 돕기 위하여 보인 것이다. 이러한 확장된 Java 클래스로부터 사용자는 필요한 만큼의 계획 클래스 객체를 만들어 사용하거나 객체 직렬화(object serialization)를 써서 저장하거나 다른 에이전트에게 전달하고 이를 실행 중에 적재(load)하여 사용할 수 있게 된다.

두 번째와 세 번째로 제시된 문제점은 Java 언어의 쓰레드(thread) 기능을 활용하여 해결한다. 기본활동은 Java 문장 단위로 이루어지고 쓰레드 개입에 의한 제어로 인하여, 비대해진 기본활동의 크기에 의한 개입 불능 현상을 피할 수 있게 된다. 지속조건의 감시 또한 독립적인 쓰레드에서 진행되는 감시 기능과 이에 따른 비동기적 제어를 통하여 이루어진다. 이에 대한 자세한 사항은 이 절의 후반부에서 다시 다루어진다.

그림 4은 VivAce의 전반적 구조와 실험에 사용한 간단한 시뮬레이터의 구조를 함께 보여준다. 에이전트의 확대된 그림(좌측 상단)에서 보는 바

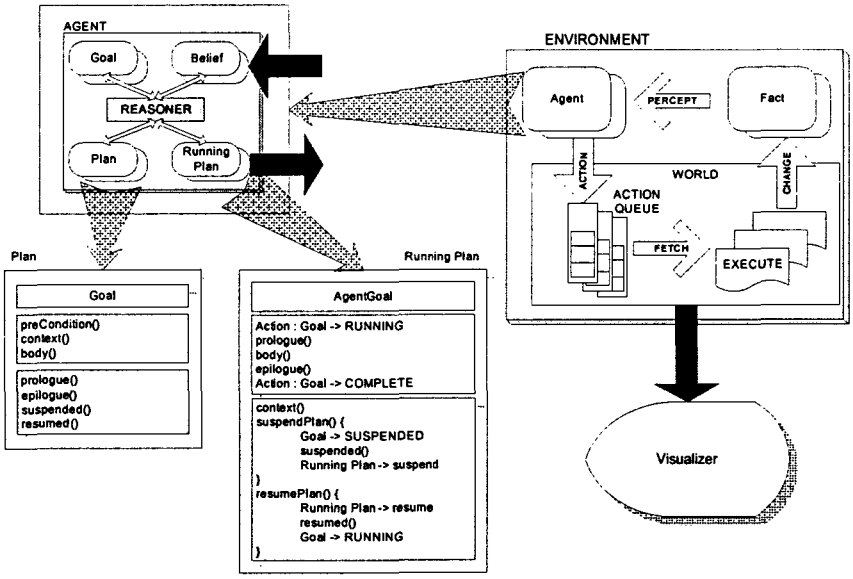
```
public class BouncePlan extends Plan {

    public boolean preCondition( ) { return true; }

    public boolean context( ) {
        int type = ((Integer)owner( ).getFact("TYPE")).intValue( );
        return (type+1)%4 == 0;
    }

    public void body( ) {
        while (true) {
            try {
                Thread.sleep(50);
            } catch(Exception e) { }

            int addX = ((Integer)owner( ).getFact("ADDX")).intValue( );
            int addY = ((Integer)owner( ).getFact("ADDY")).intValue( );

            if ((Boolean)owner( ).getFact("EDGEX")) {
                addX = -addX;
                owner( ).modifyFact("ADDX", new Integer(addX));
                owner( ).modifyFact("EDGEX", new Boolean(false));
                int type = ((Integer)owner( ).getFact("TYPE")).intValue( );
                type = (type+1) % 4;
                owner( ).modifyFact("TYPE", new Integer(type));
            }

            if ((Boolean)owner( ).getFact("EDGEY")) {
                addY = -addY;
                owner( ).modifyFact("ADDY", new Integer(addY));
                owner( ).modifyFact("EDGEY", new Boolean(false));
                int type = ((Integer)owner().getFact("TYPE")).intValue( );
                type = (type+1) % 4;
                owner( ).modifyFact("TYPE", new Integer(type));
            }
            Command command = new Command(owner( ).name( ), "MOVE", addX, addY);
            owner( ).environment( ).pushAction(owner( ).name( ), command);
        }
    }
}
```

[그림 2] 예제 에이전트 계획

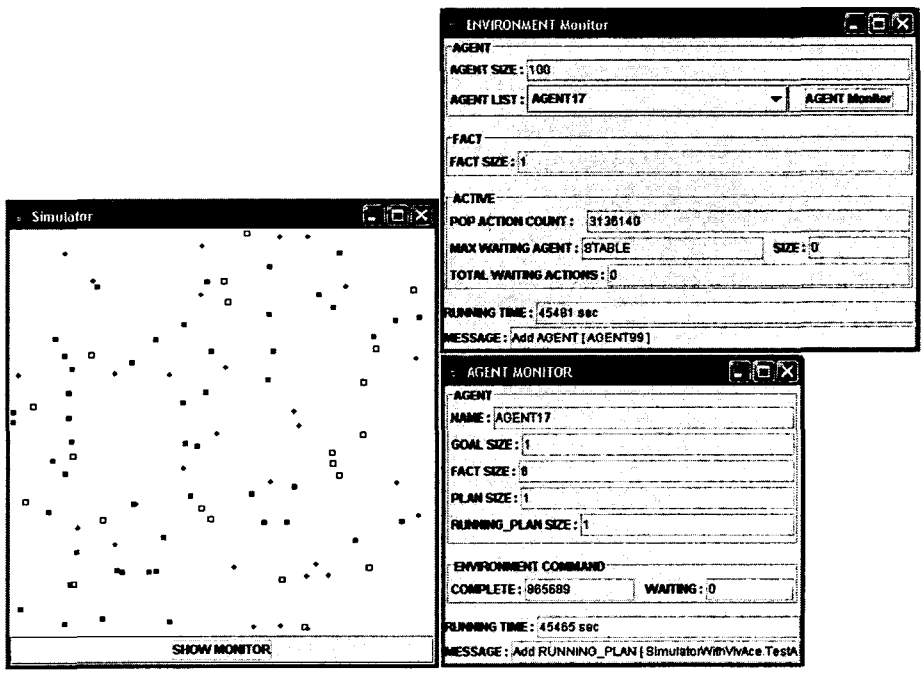
와 같이 VivAce는 계획, 목적, 믿음을 비롯한 BDI의 구성 요소를 그대로 물려받아 구현하고 있는 BDI 에이전트 구조를 기반으로 하고 있다.

VivAce의 에이전트 구조로서의 기능성과 Java 프로그램의 효율성을 실험하기 위한 환경으로 간단한 시뮬레이터를 개발하여 사용하였다. 그림 4는 벽으로 둘러싸인 2차원 공간에서 공들이 벽에 튀는 환경을 시뮬레이션하는 시뮬레이터와 함께 환경 및 각각의 에이전트의 상태를 감시하는 모니터를 같이 보여주고 있다. Pentium IV 노트북에서 500여 개의 에이전트가 동시에 활동하는 데에도 눈에 띄는 속도의 저하가 발생하지 않았다.

VivAce 에이전트의 기능을 시험하기 위해서 직접 개발한 단순 시뮬레이터 외에도 그림 6에 보인 바와 같이 IBM에서 제공하는 Robocode 시뮬레이터 [1]를 사용하여 실험을 진행하였다. 이 경우에는 Robocode의 에이전트에 대한 제약 조건을 극복하기 위하여 그림의 중간에 제시한 것과



[그림 3] VivAce 에이전트 구조와 시뮬레이터



[그림 4] 모니터를 갖춘 시뮬레이터

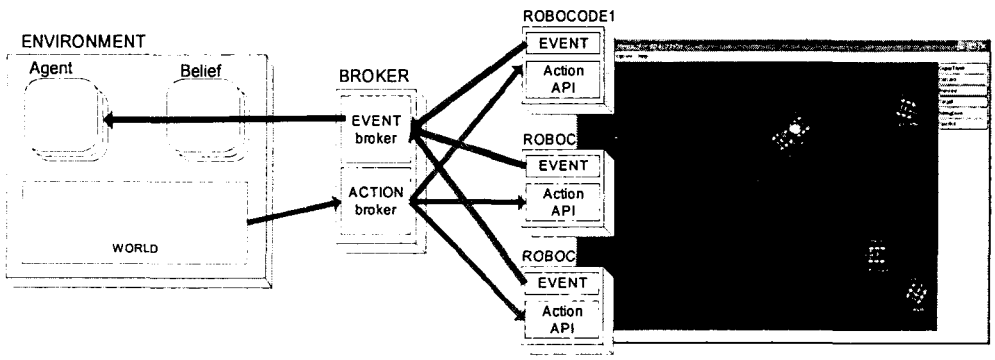
같은 미들웨어를 두어 통합하였다. 미들웨어 에이전트를 사용하여 대규모 다중 에이전트들을 통합하는 것은 다른 응용 분야에도 일반적으로 적용될 수 있는 방안이라 여겨진다.

이제 좀 더 구체적으로 앞서 제시한 문제들을 VivAce가 어떻게 해결하는지 제기된 문제 순서대로 설명한다.

1. VivAce의 계획은 단순한 Java 객체로서 사용자는 텍스트 기반의 기본 활동 지정 방식과는 달리 계획을 파싱을 통하여 계획 모음으로 읽어 들이는 과정이 불필요하다. VivAce에서 제공하는 추상 계획을 확장하여 preCondition( ), context( ), body( ) 메소드 들을 구현하는 것으로 족하며 BDI 에이전트 모델의 구성 요소인 믿음이나 목적 등은 이미 잘 정의된 VivAce의 응용 프로그램 인터페이스(API)를 써서 BDI의 규정에 부합하게 사용할 수 있다.

2. VivAce의 추론기(reasoner)는 그림 4에 보인 바와 같이 독립적인 Java 쓰레드로 구현되어 있어서 실행 가능한 계획들을 시작하거나 실행을 멈추는 등의 작업을 계획의 실행과 비동기적으로 수행할 수 있다. 추론기는 실행 중인 계획들의 지속조건을 독립적인 쓰레드 내에서 지속적으로 검사함으로써 조건을 만족하지 않는 계획의 실행을 비동기적으로 중지할 수 있다. 따라서 최소실행단위라는 개념은 더 이상 존재하지 않는다. 사실상 VivAce의 계획은 초기조건, 지속조건, 본체 외에도 아래의 보여진 바와 같은 세분화된 행위를 선택적으로 지정할 수 있다. prologue( ) 메소드는 계획이 처음으로 실행될 때 수행할 내용들을 지정하며 epilogue( ) 메소드는 계획의 실행이 종료될 때 수행할 내용을 지정한다. suspended()와 resumed( ) 메소드는 계획의 실행이 정지될 때와 정지 후 다시 재개될 때 수행할 내용을 각각 지정한다. 이러한 세부 수행 내용의 지정 기능과 비동기적 실행 제어를 통하여 앞서 제기된 기본활동의 문제를 해결한다.

```
abstract public class Plan {
   abstract boolean
            preCondition();
   abstract boolean context();
   abstract void body();
   void prologue() {}
   void epilogue() {}
   void suspended() {}
   void resumed() {}
}
```



[그림 5] 브로커를 통한 Robocde 시뮬레이터와 연결된 VivAce

3. 계획의 지속조건은 여전히 계획의 실행 중에 지속적으로 검사할 필요가 있다. VivAce의 추론기는 지속조건을 얼마만한 주기와 어떤 방법으로 검사할 것인가에 대한 재량을 가진다. 다시 말해 기본활동의 최소실행단위의 크기에 의해서 실행주기가 결정되는 것이 아니라 추론기의 재량에 의해 실행주기가 결정된다. 이 밖에도 믿음 데이터베이스에 기록된 사실과 계획의 지속조건에 표현된 사실 간의 인과관계를 미리 파악해서 믿음 데이터베이스의 변화된 사실에 의해 영향을 받는 계획의 지속조건만을 선택적으로 판단할 수 있는 방안을 RETE 알고리즘[5]을 써서 개발하고 있다. 이러한 기능을 추가하고자 하는 연구도 추론기 주도 하에 이루어지는 지속조건의 판단과 실행주기에 관한 추론기의 재량 확대에서 비롯된다.

4. 관련 연구

본 연구에서 실험을 위해 사용한 상호협력하는 지능형 에이전트를 이용한 실세계 모델링 및 시뮬레이션 시스템은 웹의 발달과 더불어 그 중요성이 인식되고 있다. 이에 대한 주요 관련 연구를 살펴본다. MICE [12]는 다양한 조건 하에서 지능형 에이전트간의 상호작용을 실험하기 위한 선구적 시뮬레이터로서 2차원 세계를 이산 사건으로 다루어 시뮬레이션을 수행한다. MICE의 또 다른 특징은 시뮬레이션 결과를 재연할 수 있다는 것이다. MICE는 시간과 사건 뿐만 아니라 에이전트의 활동과 환경의 변화도 이산적 단위로 표현하기 때문에 이러한 재연 기능이 손쉽게 이루어 질 수 있다.

Tileworld [13]는 로봇 에이전트와 역동적이고 예측할 수 없는 환경을 시뮬레이션 할 수 있다. 또한 매개변수를 통하여 로봇 에이전트와 환경의 다양한 특성을 제어할 수 있는 특징을 가지고 있다.

JAMES [16]는 Java-based Agent Modeling Environment for Simulation의 약어로서 숙고형 이동 에이전트를 이산 사건 시뮬레이터로 실현한다. JAMES는 분산 환경에서의 상호작용을 분석하고 증진하는데 있어서 에이전트 중심 소프트웨어의 개념과 역할의 중요성을 부각한다.

Go! [3]는 안전한 우량의 에이전트 기반 응용 시스템 개발을 위한 다용도 프로그래밍 언어이다. 다중 쓰레드 기반이며 엄격한 데이터형(type)을 가지고 있는 고수준 언어이다. Go!는 관계(relation), 함수(function), 활동절차(action procedure)에 대한 정의 등을 가지고 있다. 쓰레드를 통하여 활동절차를 실행하고 함수를 호출하며 필요에 따라 관계에 대한 질의를 수행한다. 다른 에이전트와의 통신도 쓰레드에서 이루어지며 이를 통한 비동기적 공조(coordination)를 이룬다. 이를 이용하여 무도실에서 반응-숙고 복합형 에이전트로 구성된 다수의 에이전트가 상호작용을 하는 예를 보여준다.

Logan과 Theodoropoulos [11]는 분산 이산 사건 시뮬레이션 기술을 응용하여 다중 에이전트 시스템을 시뮬레이션 하는 방안을 논의한다. 이들은 에이전트들의 환경을 효율적으로 분산시키는 것이 에이전트 기반 시스템을 구성하는데 핵심 문제임을 파악하고 환경을 작은 단위로 분해(decomposition)하여 부하균형(load balancing)을 촉진하는 방안을 제시한다.

Q [8]는 에이전트의 외형적 역할을 기초로 에이전트와 사용자 간의 상호작용을 기술하기 위한 언어이다. Q는 또한 컴퓨팅 전문가와 시나리오

작가 간의 인터페이스 역할을 한다. 에이전트의 내부 메커니즘에 의존하는 대신에 Q는 시나리오 작가로 하여금 에이전트의 행위를 기술하는데 집중하도록 도와준다.

5. 결론 및 향후 연구

본 연구를 통하여 VivAce라 명명한 다중 에이전트 프레임웍을 제시하였다. VivAce는 Java 프로그램의 효율성과 BDI 에이전트 모델의 장점을 활용하도록 설계되었다. 개발된 에이전트 프레임웍의 효율성을 검증하기 위하여 간단한 시뮬레이터를 직접 제작하여 JAM 에이전트와 VivAce 에이전트로 구현된 시뮬레이션 시스템을 시험하였다. 노트북 컴퓨터에서 시험한 결과 JAM으로 구현된 에이전트들이 200여 개가 될 무렵 상당한 실행 속도의 저하를 초래한 반면 VivAce 에이전트는 500여 개에 이르도록 눈에 띄는 속도의 저하가 일어나지 않았다. 물론 이는 대략적인 측정치로서 향후 표준적인 실제 환경에서의 체계적인 실험이 필요한 부분이다.

VivAce나 JAM과 같은 에이전트 프레임웍을 다른 시스템들과 정량적으로 비교하는 것은 매우 어려운 일이지만 VivAce에 적용된 개념은 대규모 다중 에이전트 응용 시스템의 개발을 위한 진일보라 생각한다. 또한 VivAce에서 추구한 작고 효율적 에이전트는 다중 에이전트 시스템을 바탕으로 구축되는 시맨틱 웹 응용 프로그램의 개발이나 유비쿼터스 환경에도 적절하리라 생각하며 이에 대한 연구를 지속할 계획이다.

반면에 VivAce에서 추구한 프로그래밍의 편의성과 작고 효율적 에이전트라는 특징은 기존 JAM과 같은 전통적 에이전트 갖는 특징을 부분적으로 잃는 결과도 가져옴을 인정하지 않을 수 없다. 특히 에이전트의 자가성찰(reflexive)이나 메타 수준(meta-level) 추론에 있어서는 JAM과 같이 계획이 정해진 규격의 텍스트로 되어 있는 경우에 추론을 하기에 용이한 점이 있다. 향후에는 시맨틱 웹의 표준 온톨로지와 이를 지원하는 추론 기능을 적극 활용하여 VivAce에서도 다양한 메타 수준의 추론이 가능하도록 연구를 진행할 계획이다.

6. 감사의 글

이 논문은 2003년도 서울시립대학교 학술연구조성비에 의하여 연구되었음.

7. 참고문헌

[1] alphaWorks Forums. Robocode. http://robocode.alphaworks.ibm.com.

[2] Davide Ancona and Viviana Mascardi. Coo-BDI: Extending the bdi model with cooperativity. In João A. Leite, Andrea Omicini, Leon Sterling, and Paolo Torroni, editors, Declarative Agent Languages and Technologies, volume 2990 of Lecture Notes in Computer Science, pages 109-134. Springer Verlag, 2004.

[3] Keith L. Clark and Frank G. McCabe. Go! for multi-threaded deliberative agents. In Jo ao A. Leite, Andrea Omicini, Leon Sterling, and Paolo Torroni, editors, *Declarative Agent Languages and Technologies*, volume 2990 of *Lecture Notes in Computer Science*, pages 54-75.

Springer-Verlag, 2004.

[4] Mark d'Inverno, David Kinny, Michael Luck, and Michael Wooldridge. A formal specification of dMARS. In *Agent Theories, Architectures, and Languages*, pages 155-176, 1997.

[5] Charles Forgy. Rete: A fast algorithm for the many patterns/many objects match problem. *Artificial Intelligence*, 19(1):17-37, 1982.

[6] Michael P. Georgeff and Franc¸ois F´elix Ingrand. Decision-making in an embedded reasoning system. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 972-978, Detroit, Michigan, 1989.

[7] Marcus Huber. JAM: A BDI-theoretic mobile agent. In *Proceedings of the Third International Conference on Autonomous Agents (Agents '99)*, Seattle, Washington, May 1999.

[8] Toru Ishida. Q: A scenario description language for interactive agents. *IEEE Comuter*, 35(11):42-47, November 2002.

[9] Nicholas R. Jennings, Katia Sycara, and Michael Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Mult-Agent Systems*, 1:7-38, 1998.

[10] Jaeho Lee, Marcus J. Huber, Edmund H. Durfee, and Patrick G. Kenny. UM-PRS: an implementation of the procedural reasoning system for multirobot applications. In *Conference on Intelligent Robotics in Field, Factory, Service, and Space (CIRFFSS '94)*, pages 842-849, Houston, Texas, March 1994.

[11] Brian Logan and Georgios Theodoropoulos. The distributed simulation of multiagent systems. *Proceedings of the IEEE*, 89(2) :174-185, 2001.

[12] Thomas A. Montgomery, Jaeho Lee, David J. Musliner, Edmund H. Durfee, Daniel Damouth, and Young pa So. MICE users guide. Technical Report CSE-TR-64-90, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, Michigan 48109, January 1992.

[13] Martha E. Pollack and Marc Ringuette. Introducing the tileworld: Experimentally evaluating agent architectures. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 183-189, Boston, Massachusetts, 1990.

[14] Anand S. Rao and Michael P. Georgeff. Modeling rational agents within a BDI-architecture. In James Allen, Richard Fikes, and Erik Sandewall, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference (KR'91)*, pages 473-484. Morgan Kaufmann, Cambridge, MA, April 1991.

[15] Anand S. Rao and Michael P. Georgeff. BDI agents: From theory to practice. In *Proceedings of the First International Conference on Multiagent Systems*, pages 312-319, San Francisco, California, June 1995.

[16] Bernd Schattenberg and Adelinde M. Uhrmacher. Planning agents in JAMES. *Proceedings of the IEEE*, 89(2):158-173, February 2001.

[17] Michael Schroeder and Gerd Wagner. Vivid agents: Theory, architecture, and applications. *Applied Artificial Intelligence*, 14(7):645-675, August 2000.

[18] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115-152, 1995.

Abstract

An Agent Application Framework for Applications based on the Semantic Web

Jaeho Lee*

Multi-agent systems for semantic web applications require efficient implementation of agent architectures without sacrificing the flexibility and the level of abstraction that agent architectures provide. In this paper, we present an agent system, called VivAce, which is implemented in Java to achieve both high efficiency and the level of abstraction provided by the BDI agent architecture. VivAce (Vivid Agent Computing Environment) has the characteristics of a vivid agent through the BDI agent model. A vivid agent is a software-controlled system whose state comprises the mental components of knowledge, perceptions, tasks, and intentions, and whose behavior is represented by means of action and reaction rules. We first identify the requirements for multi-agent systems and then present the relevant features of VivAce and experimental results.

Key words : Agent Architecture, Multi-Agent Systems, Semantic Web

* Dept. of Electrical and Computer Engineering, The University of Seoul