플랜 및 상황기반 커뮤니티 상호작용 규정

변무홍° 이재호
서울시립대학교 전자전기컴퓨터공학부 인공지능연구실
{bwikbs, jaeho}@ece.uos.ac.kr

Plan- and Context- Based Specification of Community Interactions

Moohong Byun°, Jaeho Lee
Artificial Intelligence Laboratory, Department Of Electrical and Computer Engineering, University Of Seoul

요 약

분산된 환경이 특징인 유비쿼터스 시대는 기존의 소프트웨어 개발 방법론과 다른 방법론을 필요로 하게 되었다. 이를 위해서 제안된 커뮤니티 컴퓨팅이라는 개념을 소개하겠다. 또한 커뮤니티 컴퓨팅을 구현하는 수단인 CVM에 대해서 설명하고 이를 개선한 CVM에 대해서 설명하겠다

1. 서 론

모든 것이 한 곳에서 이루어지던 컴퓨팅 환경은 무선 기술의 발전과 저 전력화 소형화 되어가는 컴퓨팅 환경의 변화로 말미암아 유비쿼터스라는 새로운 패러다임으로 나타났다. PDA,핸드폰등의 발전과 무선 인프라의 보급 확대로 인해 유비쿼터스 컴퓨팅을 위한 기반 환경은 비약적으로 발전했다. 그러나 이를 위한 새로운 소프트웨어 패러다임은 하드웨어의 발전을 따라가지 못한 측면이 있다. 이에 새로운 소프트웨어 패러다임인 커뮤니티 컴퓨팅을 도입하고 이를 실현을 위한 기반 소프트웨어인 CVM(Community Virtual Machine)에 대해서 설명하겠다. 또한 기존의 CVM이 가지고 있는 문제점과 새로 제안되는 CVM이 기존의 CVM이 가지는 문제점을 어떻게 해결하려고 하는지 설명하겠다. 여기에는 수정된 CVM이 개별적인 환경에 분산되어 있는 에이전트들을 어떻게 제어 할 수 있는지에 대해 서술 할 것이며 CVM에 의해서 해석되고 에이전트에게 커뮤니티의 일원으로써의 행동을 강제하는 커뮤니티 프로그램은 어떤 형태로 작성될 수 있는지 제안 하도록 하겠다.

2. 커뮤니티 컴퓨팅의 정의

커뮤니티 컴퓨팅이라는 개념은 개체들이 공통된 목적을 이루기 위한 협업을 위해서 커뮤니티를 구성하듯 유비쿼터스 환경하에서 분산된 컴퓨팅 자원들이 특정한 목적을 서로 협업하여 추구하는 일련의 과정을 커뮤니티를 구성한다고 정의 하였다.

커뮤니티 컴퓨팅에서 정의하는 커뮤니티는 우리가 흔히 커뮤니티가 가진다고 생각하는 커뮤니티의 공통된 관심사, 정책, 역할 등의 요소를 가진다. 이는 커뮤니티 컴퓨팅이라는 패러다임 자체가 복잡한 분산 컴퓨팅 환경을 모델링 하는데 있어서 기존의 우리가 쉽게 유추할 수 있는 개념을 통해서 모델링 함으로써 좀더 직관적이고 자연스럽게 하기 위한 방편이었음을 감안한다면 이해가 될 것이다.

우선 커뮤니티가 가지는 역할, 정책, 구성원 등의 요소가 커뮤니티 컴퓨팅에서 어떠한 의미를 가지는 가에 대해 살펴보도록 하겠다. 커뮤니티의 목적은 에이전트들이 협업을 통해 이루어 내고자 하는 서비스 자체이다.

커뮤니티 컴퓨팅에서의 역할은 커뮤니티 프로그램의 관점에서 볼 때 각각의 에이전트들이 커뮤니티에서 담당하는 기능점(Function Point)에 대한 별칭이라고 볼 수 있다. 이러한 역할은 커뮤니티 에 참여하는 에이전트들에 의해서 할당될 수 있으며 이런 담당을 에이전트들에게 할당하는 방법도 주요한 논의거리 중에 한가지이다.

정책은 커뮤니티 차원에서 각각의 개별적인 에이전트들에게 가해지는 특별한 제약사항이라고 할 수 있으며 이는 개별 에이전트의 지식체계 내에 미리 약속된 형태로 전달되어 에이전트가 이를 준수 하도록 구성되던가 혹은 커뮤니티가 에이전트가 하는 행위자체가 유효한지 결정하는 형태로 구성이 가능하다.

구성원은 커뮤니티를 구성하게 되는 에이전트 자신이다. 본 논문에서는 분산된 자원들을 자율성을 가지는 에이전트로 정의 하였다

프로토콜은 커뮤니티에 속한 에이전트들이 상호작용하게 되는 흐름을 에이전트의 플랜형태로 서술한 것이다. 프로토콜에서 서술한 내용 자체는 에이전트들이 어떤 내용을 주고 받을 것인가에 대한 서술이지만 안에는 에이전트들 상호간의 공유하게 되는 온톨로지 요소와 에이전트가 수행하게 되는 기능들의 워크플로어(work flow) 요소가 혼재 되어 있다고 할 수 있다. 다시 말해서 에이전트가 커뮤니티 프로그램을 통해 수행하게 되는 일련의 시나리오 자체가 서술되어 있다고 볼 수 있다. 그러나 영화나 연극에서 볼 수 있는 순차적인 시나리오라기 보다는 에이전트들이 지키기를 강요 받는 일종의 지침서라는 표현이 정확할 것이다. 본 논문에서는 프로토콜을 두 부분으로 세분화 하여 '커뮤니티 상황' 부분과 '커뮤니티 플랜'으로 구분하였다. 커뮤니티 상황 부분은 온톨로지 정보와 특정한 조건에 대한 정보를 담고 있다. '커뮤니티 플랜'부분은 커뮤니티가 수행하는 일련의 협업과정에 대한 워크플로어와 이를 에이전트에게 실행시키기 위한 목적 정보를 서술하고 있다. 이 두 가지 부분은 커뮤니티 프로그램을 구성하는 핵심적인 요소이며 커뮤니티는 이를 통해 표현되게 된다.

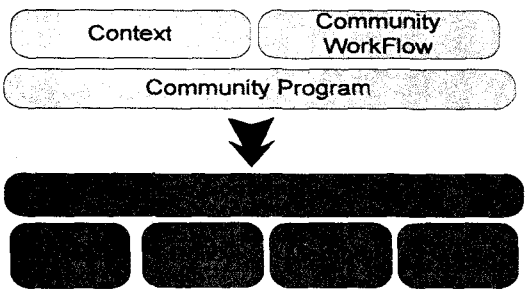

그림 1 커뮤니티 컴퓨팅

3. 기존의 CVM 구조

CVM이란 커뮤니티 가상 기계(Community Virtual Machine)의 약자이다. CVM이란 이름은 자바(Java)언어에서 자바로 쓰여진 프로그램이 실행되는 플랫폼 독립적인 실행 환경인 JVM(Java Virtual Machine)이듯 커뮤니티 프로그램이 실행되는 기반 환경이라는 뜻에서 붙여진 이름이다. JAVA프로그램이 자신이 실제로 수행되는 기계에 환경에 영향을 받지 않고 자바 바이너리로 컴파일 되고 가상의 컴퓨터 안에서 수행되는 것처럼 CVM도 에이전트들의 개별적인 환경에 구애 받지 않고 에이전트들을 서로 연결해주고 협업이 가능하도록 하는 환경을 구축해 줌으로써 커뮤니티를 구성하도록 도와준다.

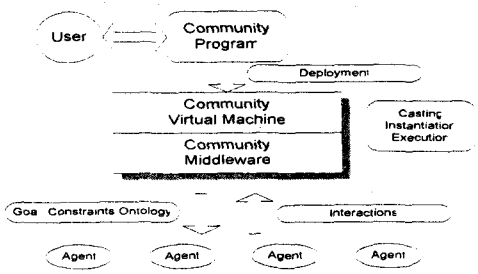

그림 2 Community Virtual Machine

그림2는 CVM이 동작하는 개념적인 모습을 나타낸 것이다. 사용자는 커뮤니티 프로그램을 작성하게 된다. 이 프로그램은 CVM을 통해서 개별 에이전트들에게 가공되어서 전달되게 된다. 에이전트들은 CVM으로 전달 받은 목적, 제약조건, 온톨로지 등을 사용해서 커뮤니티를 구성하고 커뮤니티가 구성된 목적을 수행하게 된다.

기존의 CVM이 어떤 형식으로 구성되어 있는지 간략하게 살펴 보고 어떠한 문제점이 있는지 알아보겠다. 기존의 CVM은 에이전트간의 협업을 달성하는데 가장 중요한 부분은 에이전트 간의 상호작용(Interaction), 즉 프로토콜(Protocol)이라고 보았다. 즉 에이전트들에게 어떤 커뮤니티의 일원으로써 역할을 해내게 하기 위한 Protocol을 서술하는 과정을 가장 필요한 부분으로 본 것이다. 이것을 가능하게 하기 위해서 에이전트 간의 상호작용을 AUML같은 형식을 써서 정형화 하여서 나타내었고 이를 에이전트에게 강제하는 방법으로써 플랜을 사용하였다. 커뮤니티의 목적을 이루기 위해 거쳐야 하는 일련의 과정을 AUML로 표현하고 이를 플랜으로 변환하는 방법을 제공함으로써 에이전트가 커뮤니티를 구성하게 하고 일원으로써 행동하게 만든 것이다.

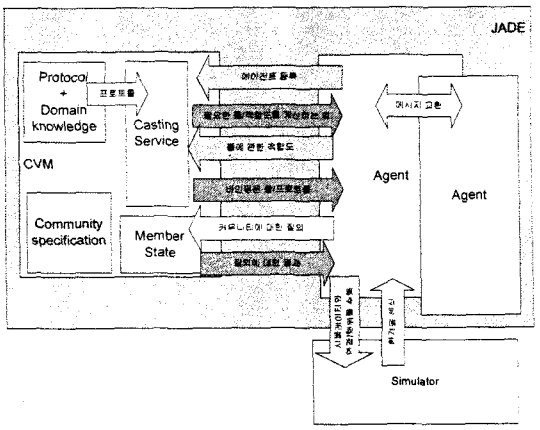

그림 3 기존의 CVM의 동작 구조

그림3은 기존CVM이 동작하는 모습을 간략하게 나타낸

그림이다. 에이전트간의 통신이 가능하게 해주는 커뮤니티 미들웨어는 JADE를 사용하고 있다. 커뮤니티 프로그램을 작성하면 프로토콜과 도메인지식 커뮤니티에 대한 메타 서술로 분리되어 CVM에 저장되게 된다. 커뮤니티에 대한 메타 서술은 커뮤니티의 이름, 설명 등으로써 CVM내에서 개별적인 커뮤니티들을 관리하는데 쓰이게 된다.

프로토콜은 에이전트들의 역할을 할당하는 부분인 캐스팅 서비스(Casting Service)에 전달된다. 캐스팅 서비스는 CVM에 등록된 에이전트들과 프로토콜에 쓰여 있는 역할들을 연결하게 되는데 여기에는 C-NET의 간단한 변형 알고리즘이 쓰이게 된다. 이 CVM은 에이전트의 행동이 CVM에서 전달받아 이루어 진다는 것과 에이전트의 역할의 할당, 에이전트간의 상호작용을 지원하는 인프라가 구축되어 있는 점등을 특징으로 가지고 있었다.

그러나 기존의 CVM은 다음과 같은 문제점도 가지고 있었다. 첫 번째 온톨로지에 대한 지원이 없었다. CVM과 커뮤니티 프로그램 개별 에이전트가 가지는 플랜들은 모두 온톨로지가 사전에 정의 되어 있다는 가정하에 작성되었다. 두 번째 초기단계에서 프로토콜이 에이전트들에서 일시에 전달되었기 때문에 일단 커뮤니티가 시작되면 커뮤니티 자체에 대한 제어가 어려웠다. 즉 커뮤니티가 어떤 상태인지 파악한다거나 동적으로 가용하지 않은 에이전트를 교체한다거나 하는 동작이 이루어 지지 않았다. 이를 해결 하기 위해서 기존의 CVM의 동작 방식에 수정을 가하였다. 다음 절에서는 수정된 CVM에 대한 설명을 하도록 하겠다.

4. 개선된 CVM

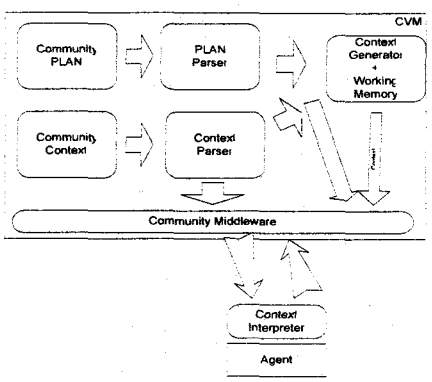

그림 4 수정된 CVM

커뮤니티 프로그램에서 서술된 커뮤니티 플랜은 파싱 과정을 거쳐서 커뮤니티의 상황을 발생시키는 부분과 에이전트들에게 보내지게 된다.

개별적인 에이전트들에게 전달되는 커뮤니티 플랜은 에이전트들이 커뮤니티 내에서 달성해야 하는 골의 순서를 표현하게 되며 에이전트가 수행할 수 있는 플랜의 형태로 변환되어서 전달된다. 이 플랜은 에이전트가 수행해야 할 골과 선행조건으로 구성되어 있으며 선행조건은 커뮤니티 상황을 사용하게 된다.

커뮤니티 상황에 대한 서술은 상황 파서를 거쳐서 미들웨어와 상황 생성기에 전달되게 된다. 미들웨어에 전달된 상황 서술은 개별에이전트들이 가지고 있는 상황 변환기에 전달되게 되며 이로써 에이전트는 커뮤니티에서 일어나는 상황에 대한 온톨로지를 이해할 수 있게 된다. 말하자면 일종의 통역사를 가지게 되는 것이다. 상황 발생기로 들어간 상황 서술은 커뮤니티 상황 발생기에서 상황을 발생시키게 된다. 상황은 일종의 이벤트로써 관련된 에이전트와 연결되어 있는 상황 해석기에 전달되어 월드모델에 적재되어 에이전트가 작업을 수행할 수 있도록 해준다.

CVM이 각 역할에 해당하는 에이전트를 할당하는 방법에도 변화가 생겼다. 기존의 CVM에서는 커뮤니티를 초기에 생성하는 시점에서 각 역할에 필요한 에이전트들을 일괄적으로 할당하고 커뮤니티가 일단 시작되고 나면 추가적인 할당을 하지 않았다. 아울러 CVM은 각 에이전트와 역할에 대한 할당 정보만을 유지할 뿐 어떠한 추가 정보도 유지하고 있지 않았다. 이러한 방식은 중간에 에이전트가 이상이 생겨서 작동이 멈추면 커뮤니티가 멈춰버리는 문제점이 있었다. 멈춰버리는 에이전트를 검출하는 방법을 사용하고 커뮤니티를 다시 시작하거나 특정부분으로 롤백(roll back)하는 방법을 생각해 볼 수도 있으나 근본적인 문제점은 CVM이 커뮤니티가 현재 처해 있는 상황에 대한 정보를 가지고 있지 않기 때문이다. 이에 현재 커뮤니티가 돌아가고 있는 상황에 대한 정보를 CVM이 유지하도록 하였다. 구체적으로 이러한 정보를 유지하는 곳은 CVM내에 상황발생기이다.

상황 발생기는 룰 엔진을 가지고 있으며 커뮤니티가 서비스를 수행하는데 있어서 발생하는 상황이 룰로 표현되어 있다. 커뮤니티플랜이 실행되는 상태도 상황으로 표현되어 있으며 에이전트가 순차적인 일을 수행할 때 다음작업을 수행하기 위한 선행조건으로 사용된다.

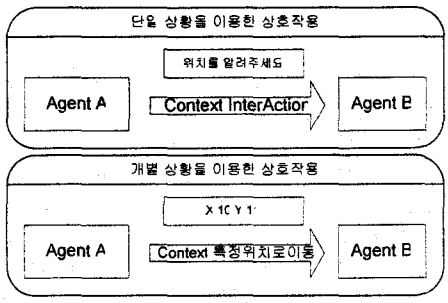

그림 5 상황을 이용한 상호작용

상황은 에이전트 사이의 협업을 위한 통로로도 사용된

다. 즉, 서로 다른 에이전트들이 협업을 위한 의미를 이해하는데 사용되는 것이다. 상황이 나타내는 정도는 범용적일 수도 있고 아주 개별적일 수도 있다. 가령 A라는 에이전트와 B라는 에이전트가 서로의 위치 정보를 주고 받으면서 특정한 작업을 수행한다고 할 때 A와 B 사이의 상황은 단순하게 상호작용이라는 상황에 주고 받는 메시지를 담아서 주고 받거나 '이러한 일을 처리해 달라'는 상황을 모두 정의 해놓고 각 상황마다 개별적으로 메시지를 담아서 보낼 수도 있는 것이다.

5. 커뮤니티 프로그램

사용자가 커뮤니티를 개발하기 위해 프로그래밍하는 부분이 커뮤니티 프로그램이다. 커뮤니티 프로그램은 최종목적을 달성하기 위해서 에이전트들이 진행해야 하는 일련의 과정을 담고 있다. 이 프로그램은 CVM을 거쳐서 에이전트들에게 전달되며 독립적인 에이전트들이 공통의 목표를 위해서 협업하도록 강제하는 방법이다.

커뮤니티 프로그램은 크게 두 가지 부분으로 서술되어 있다. 첫 번째는 커뮤니티가 사용하는 특정한 상황을 정의하는 부분인 커뮤니티 상황 부분이다. 커뮤니티 상황 부분은 상황을 나타내는 유일한 이름과 그 상황이 영향을 미치는 역할, 상황이 가지는 변수의 자료형, 상황과 상황 사이의 관계를 서술하는 룰로 구성되어 있다.

```
<CommunityContext>
<Context name="interaction" >
   <Role>ALL</Role>
   <Parameter name="message" type="String"></Parameter>
</Context>

<Context name="fire" >
   <Role>fireman</Role>
</Context>
<Context name="emergency" >
   <Role>fireman</Role>
</Context>

<ContextRule>
<Rule name="warning">
<LHS>fire</LHS>
<RHS>emergency</RHS>
</Rule>
</ContextRule>
</CommunityContext>|
```

그림 6 커뮤니티 컨텍스트 예

커뮤니티 프로그램을 구성하는 두 번째 요소는 커뮤니티에서 에이전트들이 해야 하는 일련의 과정을 나타내는 커뮤니티 플랜이다. 커뮤니티 플랜을 구성하는 기본 단위는 커뮤니티가 에이전트들에게 요구하는 골이다. OR, SEQUENCE, PARALLEL등의 제어를 위한 예약어를 제외하면 커뮤니티플랜은 골과 골을 수행할 역할의 서술만으로 구성된다. 그림7은 연설을 위한 커뮤니티 구성 예 이다.

SEQUENCE는 순차적인 수행을 나타낸다. 즉 연설은 연사가 발표를 시작하고 청중들이 듣기를 시작한다. 연사의 발표가 시작되지 않으면 청중들도 듣기를 시작하지 않는다. PARALLEL은 동시에 수행됨을 나타낸다. 청중들의 듣기가 끝나면 청중들은 커피를 마시고 연사는 밥을 먹게 되며 두 개의 일이 모두 끝나면 공통으로 연설을 종료하는 작업을 수행하게 된다.

```
<CommunityPlan>
<Sequence>

<Step role="speaker" >
<perform>"StartSpeech"</perform>
</Step>

<Step role="audience" >
<perform>"StartListening"</perform>
</Step>

<PARALLEL>

<Step role="audience" >
<perform>"CoffeeBreak"</perform>
</Step>

<Step role="speaker" >
<perform>"LUNCH"</perform>
</Step>

</PARALLEL>

<Step role="ALL" >
<perform>"EndSpeech"</perform>
</Step>

</Sequence>
</CommunityPlan>
```

그림 7 커뮤니티 플랜의 예

6. 결론

분산되어 있는 에이전트들에게 외부에서 프로그램을 주입하여 협업을 가능케 하는 것이 커뮤니티 컴퓨팅이 추구하는 목표이다. 이를 위해서 외부에서 주입하는 프로그램은 어떤 모습을 하고 있어야 하는 지와 어떠한 인프라 서비스가 필요한지에 대한 방안으로 나온 것이 CVM이다.

기존의 제안된 CVM은 에이전트간의 상호작용을 커뮤니티 차원에서 하는데 중점을 두었었다. 그러나 커뮤니티 프로그램에 에이전트 종속적인 서술이 들어가고 커뮤니티를 동적으로 구성하는데 어려움이 있었다. 에이전트에 종속적인 서술은 온톨로지 지원이 미비했기 때문이며 이를 해결하기 위해서 수정된 CVM에서는 상황을 정의하였다. 커뮤니티 워크플로어에서 분리된 상황에 대한 서술은 커뮤니티 온톨로지의 추출을 어느 정도 가능케 하였다.

이는 이질적인 에이전트에서 돌아가는 커뮤니티 프로그램을 위해서 필요한 부분이다. 그러나 아직 커뮤니티의 온톨로지가 정의 되었다고 하기에는 미비한 점이 많으며 에이전트간의 협업을 보다 잘 표현하기 위한 서술도 추가로 연구 되어야 할 것이다.

7. Acknowledgment

본 논문은 과학기술부 특정연구개발사업 과제인 "자동차 모듈 설계용 e-엔지니어링 프레임웍 개발"의 일부이며, 연구수행에 지원해 주신 관계자 여러분께 감사 드립니다.

8. 참조

[1] Marcus Huber, "JAM, A BDI-theoretic Mobile Agent," *Proceedings of the Third International Conference on Autonomous Agents (Agents-99)*, May, 1999.

[2] 변무흥,곽별샘,이재호, "유비쿼터스 환경을 위한 실행가능한 에이전트 상호규정", 한국지능정보시스템학회-웹코리아 포럼 2005 년 공동 추계정기학술대회, pp.163-170, 2005 년 11 월.

[3] The Foundation of Intelligent Physical Agents (FIPA), http://www.fipa.org

[4] The FIPA Agent UML Web Site, http://www.auml.org/