다중 에이전트 환경에서의 커뮤니티 기반 유비쿼터스 시스템을 위한 모델과 개발 도구

(Community Computing Model and Development Tool for Community-based Ubiquitous System in Multi-agent Platform)

정 유 나 † 이 정 태 †† 김 민 구 ††
(Youna Jung) (Jungtae Lee) (Minkoo Kim)

요 약 최근 유비쿼터스 시스템을 설계하고 개발하기 위한 방법들 중 하나로서, 다중 에이전트 모델을 이용하는 방식이 연구되고 있다. 이러한 방식은 다중 에이전트 모델의 장점을 유비쿼터스시스템에서 수용할 수 있도록 하였다. 그러나, 특정 유비쿼터스 시스템에 대해서는 기존의 다중 에이전트 모델들만으로는 완벽하게 기술하기에 어려운 부분이 있다. 본 논문에서는 사용자의 요구가 동적으로 생성되면 개체들이 협업하여 이를 해결하는 협업 위주의 유비쿼터스 시스템에 초점을 맞추었다. 이러한 시스템에서는 서비스를 제공하기 위하여 형성되는 컴퓨팅 요소들간의 협업 조직이 매우 중요하지만, 일반적인 다중 에이전트 모델로는 이러한 협업 조직을 효과적으로 표현하기가 쉽지 않다. 즉, 기존의 모델만으로는 이러한 협업 조직의 동적인 생성과 소멸, 동적인 조직의 구성방식, 그리고 서비스를 제공하기 위한 조직 내에서 또한 조직들간의 협업 방식을 기술하기가 쉽지 않다는 것이다. 따라서 본 논문에서는 그러한 협업 조직을 커뮤니티라 하고, 협업 중심의 유비쿼터스 시스템을 커뮤니티에 기반하여 기술하는 추상화 모델로서 커뮤니티 컴퓨팅 모델을 제안하였다. 또한 본 논문에서는 유비쿼터스 시스템을 위한 체계적인 개발 체계가 수립되어 있지 않다는 점에 주목하고, 제안된 커뮤니티 컴퓨팅 모델을 기반으로 응용 시스템을 개발하기 위한 시스템 개발 과정을 제안하고 이를 빠르고 편리하게 도와주는 개발 도구를 구현하여 보았다. 마지막으로, 소규모의 유비쿼터스 시스템을 제안한 추상화 모델로 기술하고 개발 도구를 사용하여 다중 에이전트 플랫폼에서 개발하여, 제안한 방법의 실현성을 검증하여 보았다.

키워드 : 커뮤니티, 협업, 유비쿼터스, 다중 에이전트 시스템, MDA

Abstract To develop a ubiquitous system, several researches have been tried to apply multi-agent models to design a system. Even though current multi-agent models provide many benefits with ubiquitous system developments, there are still some deficiencies in completely supporting the characteristics of ubiquitous systems such as dynamic formation and termination of mission-oriented organizations and interrelationship between organizations. In addition, existing agent-based models only concern aboutanalysis and design of a system, then place a burden of implementation on developers. Therefore, in this paper, we propose the high-level abstraction model of a multi-agent based ubiquitous system and the development process concerning implementation as well as design of systems. In addition, we implemented a development toolkit, called as CDTK (Community computing system Development Tool Kit), then developed a small community computing system using the CDTK.

Key words : Community, Cooperation, Ubiquitous, Multi-agent System, MDA

· 본 연구는 21세기 프론티어 연구개발사업의 일환으로 추진되고 있는 정보통신부의 유비쿼터스컴퓨팅및네트워크원천기반기술개발사업의 지원에 의한 것임
† 학생회원 : 아주대학교 정보통신전문대학원
serazade@ajou.ac.kr
†† 종신회원 : 아주대학교 정보통신전문대학원 교수
jungtae@ajou.ac.kr
minkoo@ajou.ac.kr
논문접수 : 2006년 2월 16일
심사완료 : 2006년 10월 24일

1. 서 론

지금까지 복잡한 분산 시스템을 좀 더 쉽게 이해하고 모델화하여 개발을쉽게 하기 위해서 많은 소프트웨어 개발자들이 에이전트 기술을 이용하였다[1]. 최근 들어, 관심을 받고 있는 유비쿼터스 시스템도 일종의 복잡한 분산 시스템으로서, 사람의 생활에 필요한 서비스를 지원

하기 위한 다중 에이전트 시스템의 응용으로 볼 수 있다[2]. 이러한 다중 에이전트 기반의 유비쿼터스 응용 시스템은 단지 다중 에이전트 시스템의 응용 분야를 유비쿼터스로 확장하는 것으로만은 충분하지 않다. 왜냐하면 어떠한 유비쿼터스 시스템들은 고유의 요구사항을 가지기 때문이다. 본 논문에서는 서비스에 대한 요구가동적으로 생성되며 이러한 서비스를 개체간의 협업으로 제공하는 유비쿼터스 시스템들에 초점을 맞추었다. 이러한 시스템에서는 사용자의 요구가 동적으로 발생하고, 서비스를 제공하기 위한 개체들간의 협업 조직이 동적으로 생성되며, 조직 내에서 구성 멤버들간에 상호작용이 동적으로 이루어져야 한다. 그러나 기존의 다중 에이전트 모델들로는 이러한 협업 기반 유비쿼터스 시스템의 특징들을 완벽히 표현하기 어려운 면이 있다. 그러므로 협업 중심의 유비쿼터스 시스템에 적용 가능한 추상화 모델이 필요하다. 따라서 본 논문에서는 그러한 목적 지향적 협업 조직을 커뮤니티로 추상화하여 협업 위주의 유비쿼터스 시스템을 하나의 커뮤니티 컴퓨팅 시스템으로 기술하는 커뮤니티 컴퓨팅 모델을 제안한다. 이에 더하여 본 논문에서는 대부분의 모델들이 시스템을 추상적으로 기술하고 있어서, 이러한 모델과 실제 개발 시스템 사이에는 적지 않은 틈이 있는 것을 주목하였다. 따라서 본 논문에서는 다중 에이전트 기반의 협업 중심 유비쿼터스 시스템에 대하여 추상화 모델뿐만 아니라 구현을 고려하는 모델까지 정의하여 하나의 유비쿼터스 시스템을 개발하는 전체 과정을 제안하고 이를 도와주는 개발 도구를 구현하였다. 본 논문에서 제안하는 커뮤니티 기반의 추상화 모델과 MDA(Model Driven Architecture) 방식을 적용한 시스템 개발 과정과 개발을 쉽고 편리하게 도와주는 개발 도구를 이용하여, 협업 위주의 유비쿼터스 시스템을 편리하고 빠르게 개발할 수 있을 것으로 기대한다.

본 논문의 구성은 아래와 같다. 2장에서는 관련연구와 함께 연구의 필요성을 제시하고, 3장에서는 제안하고자 하는 협업 중심 유비쿼터스 시스템에 대한 추상화 모델과 시스템 개발 과정에 대해 간략히 소개한다. 이어서 4장에서는 개발 과정에 사용되는 모델들을 세부적으로 소개하고, 그들 간의 변환관계를 제시한다. 5장에서는 4장에서 제안한 모델들을 이용하여 소규모의 유비쿼터스 시스템을 구현한 결과를 소개하고, 5장에서는 구현 결과를 토대로 제안한모델과 개발 도구에 대한 평가와토론을 한다. 마지막으로 6장에서는 결론과 함께 향후 연구를 제시한다.

2. 관련 연구 및 연구의 필요성

에이전트를 이용한 시스템의 추상화와 개발 방법이 연구되면서 기존의 방식들로는 에이전트의 유동적이고 자율적인 문제 해결 방식과 에이전트들 간의 상호작용 및 에이전트 시스템의 조직적인 구조를 완벽히 표현할 수 없다는 결론을 내리고 에이전트 기반의 분석 및 설계를 위한 방법을 별도로 연구하였는데, 그 대표적인 예가 Gaia 프로젝트이다[3,4]. Gaia에서는 다중 에이전트 시스템을 서로 상호작용을 하는 여러 종류의 역할(role)들로 이루어진 조직들의 집합으로 정의하였다. Gaia가 제시한 방법론을 이용하여 시스템에 대한 요구사항을 모델로 표현하고 이를 더 구체적인 모델로 발전시키는 방식을 통하여 시스템에 대한 설계를 할 수 있다.

그러나, 이러한 Gaia를 포함한 여러 연구에서 제시한 에이전트 기반 설계 방법론을 유비쿼터스 시스템을 개발하는 것에 그대로 적용하기에는 어려움이 있다. 그 첫 번째 이유로서, Gaia등에서 제시한 에이전트 기반 방법론이 여전히 많은 협업 중심의 유비쿼터스 컴퓨팅의 특성을 완벽히 표현하기에는 적절하지 않다는 것이다. 이러한 유비쿼터스 시스템에서는 서비스를 제공하기 위하여 구성되는 협업 조직의 구조와 이들 간의 상호 관계는 매우 중요하다. 그러나, 기존의 에이전트 기반의 설계 방법들에서 이러한 조직의 구조는 역할(role)과 상호작용(interaction) 모델에서 암시적으로 표현될 뿐 외부적으로 나타내어지지 않는다. 그러나 유비쿼터스 시스템의 많은 부분을 차지하는 협업 기반의 유비쿼터스 시스템에서는 이러한 조직을 드러내어 정의하는 것이 중요하다. 왜냐하면 그러한 유비쿼터스 시스템의 목적은 협업 조직의 목적을 달성시키는 것으로 실현되기 때문이다. 또한 이러한 조직을 표현할 수 있는 에이전트 협업 모델인 AALADIN[5]이나 BRAIN[6]에서는 목적을 달성하기 위한 협업과정을 상세히 기술하기 어려웠으며, 협업 조직을 표현하고 있지만 이러한 조직의 동적인 생성과 소멸, 그리고 조직의 역할에 대한 멤버의 동적 할당 등은 고려하지 않고 있다. 그러므로 협업 위주의 유비쿼터스 시스템을 표현하기 위해서는 조직의 구조와 그들 간의 또는 그 내부에서의 상호 작용, 조직의 동적인 생성과 소멸, 그리고 조직을 생성하기 위한 컴퓨팅 요소의 동적 할당 등을 표현할 수 있는 추상화 모델이 필요하다.

이를 위해서, 이전 연구에서는 목적을 가지는 협업 조직의 구조를 추상화하기 위하여 커뮤니티 개념을 도입하였다[7-9]. 커뮤니티 개념은 이미 여러 에이전트 협업 모델에서 소개된 바 있다[10,11]. 유비쿼터스 분야에서도 이러한 개념이 적용되었는데, 그 예가 PICO(Pervasive Information Community Organization) 프로젝트[12]이다. PICO는 유비쿼터스 환경에서 끊임 없이 서비스를 자동적으로 제공하는 것을 목적으로 이를 위해 디바이스와 소프트웨어 에이전트들 간의 협업을 기술하고 있다. PICO의 미들웨어 시스템은 실시간으로 사용자가 원하는 서비스를 제공하기 위하여 이를 처리할 수 있는 커뮤니티를 동적으로 생성하여 운용한다. PICO 프로젝

트의 주목할 만한 성과는 에이전트간의 협업을 위한 프레임워크로서 커뮤니티 컴퓨팅 개념을 소개한 것이라고 할 수 있다. 커뮤니티 컴퓨팅 개념은 이질적인 하드웨어와 에이전트 간의 협업과 의사소통을 가능하게 하는 플랫폼을 제공한다. PICO의 커뮤니티는 공통의 목적을 달성하기 위해 협력하는 하나 이상의 에이전트들로 정의되는데, 이는 에이전트들 간의 협업을 위한 프레임워크를 제공한다. 커뮤니티 컴퓨팅 개념은 이질적인 유비쿼터스 환경에서 자동적이고 연속적인 서비스를 제공하기 위하여 컴퓨팅 요소간의 실시간 협업이라는 요구사항을 충족시켜 주고 있다. 따라서 우리의 협업 기반 유비쿼터스 시스템을 기술하는 추상화 모델에서 컴퓨팅 요소들이 구성하는 목적지향적인 조직 구조를 이러한 커뮤니티 개념을 응용하여 기술하였다. 본 논문에서는 커뮤니티 개념으로 협업 구조를 모델링 한 유비쿼터스 시스템을 커뮤니티 컴퓨팅 시스템이라고 정의하여 사용하겠다.

본 논문에서 주목하는 또 하나의 사실은 기존의 에이전트 기반 모델들이 시스템에 대한 분석과 설계 과정만을 고려하고 있다는 것이다. 그러나 유비쿼터스 시스템에 대한 요구 분석과 설계를 구현으로 이끌어 낼 수 있는 체계적인 개발 과정이 필요하다. 이를 위해서 이전 연구에서는 MDA (Model Driven Architecture)[13] 방식을 적용하여 협업중심의 유비쿼터스 시스템을 위한 개발 과정을 제안하였다. MDA는 초기의 추상화된 모델로부터 구현을 고려하는 좀 더 자세한 모델들로 발전시켜가며 시스템을 개발해 가는 방식이다. 지금까지의 시스템 개발 과정에서 모델은 단지 구현을 위한 안내서 정도로만 여겨졌다. 그러나 사실 이러한 안내서만으로 고도로 복잡하게 분산되어 있는 시스템을 구현하는 것은 쉽지 않은 일이다. 따라서 MDA에서는 이러한 추상화된 설계에서 실제 시스템 구현까지 이어지는 전체 개발 과정을 추상화 수준이 다른 여러 모델들을 이용하여 기술한다. 이를 위해서, 먼저 시스템에 대한 요구분석으로부터 얻은 고 수준의 추상화 모델을 기술한 후, 이로부터 최종 시스템을 직접적으로 기술하는 구체적인 모델을 얻을 때까지 모델을 다듬어 나간다. 이러한 MDA 방식을 적용함으로써 초기에는 개발자들이 구현에 관련된 복잡한 문제에서 벗어나 시스템을 추상화 개념으로 기술하는 것에만 집중하도록 하고, 추상화 모델로부터 시스템을 개발할 때까지 더 구체적인 모델들로 변환시키면서 시스템을 개발하였다. 그러나 이전 연구에서는 커뮤니티 컴퓨팅의 개념 및 모델이 구체적으로 제시되지 않았으며, 이를 기반으로 시스템을 개발하여 모델검증하고 평가하는 작업이 이루어지지 못하였다. 따라서 본 논문에서는 기존의 연구를 발전시켜 좀 더 구체적인 모델을 제시하고, 이를 기반으로 한 시스템의 시뮬레이션을 수행하여 모델에 대한 평가를 하겠다. 구체화 된 커뮤니티 컴퓨팅 시스템의 개발과정과 모델들은 다음 장에서 상세히 다루도록 한다.

3. 커뮤니티 컴퓨팅 시스템의 개발 과정

본 논문에서는 다중 에이전트 기반의커뮤니티 컴퓨팅 시스템을 위한 추상화모델과 이로부터 시스템을 구현하는 체계적인 개발 과정을 제안하고자 한다. 제안한 추상화 모델에서는 커뮤니티 개념을이용하여 공통된 목적 아래 실시간으로 결성되는 협업 조직들과 그들 내부에서의 상호작용을 명시적으로 기술할 수 있다. 이를 기반하여 시스템을 구현하기 위해 MDA 개발 방식을 적용한 개발 과정을 통해 제안된 추상화 모델로부터 좀 더 구체적인 모델로의 변환과정을 통해 시스템 개발에이르도록 한다. MDA 개발 방식을 따르기 위해서 본 논문에서는 서로 다른 추상화 수준을 표현하는 모델들과 각 모델을 기술하기 위한 언어를 정의하였다. 이러한 커뮤니티 컴퓨팅 시스템의 개발 과정과 개발 후의 시스템 운영 환경을 그림 1과 그림 2에서 나타내어 보았다.

커뮤니티 컴퓨팅 시스템을 개발하기 위해서는 먼저 개발하고자 하는 시스템을 커뮤니티 개념에 입각하여 가장 고 수준의 추상화 모델인 CCM(Community Computing Model)으로 기술한다. CCM은 시스템에 대한 요구사항과 운영 환경을 모델링하는 것을 목적으로 한다. 이러한 CCM을 기술하기 위하여 CML(Community computing Modeling Language)를 정의하였다. CCM보다 좀 더 구체적으로 시스템을 기술하는 모델로서 CIM-PI(Platform Independent Community Implementation Model)을 정의하는데, 이는 CML에서 기술한 요구사항의 해결 방안이 실제 시스템에서 어떤 종류의 컴퓨팅요소들과 그들간의 어떠한 상호작용에 의하여 해결되는지를 플랫폼에 독립적인 수준으로 기술한다. CIM-PI의 구조적인 틀과 커뮤니티 구성 정보들은 CCM으로부터 생성하고, 그 외에 나머지는 추가정보를 이용하여 개발자가 채워 넣는다. 이러한 CIM-PI의 모델링 언어로서 CIL-PI를 정의하여 사용한다. 그 다음 단계로, 실제 시스템이 구현될 특정 플랫폼을 고려하여 시스템을 기술한CIM-PS(Platform Specific Community Implementation Model)으로 모델을 구체화시킨다. CIM-PS는 CIM-PI에 시스템이 어떻게 특정 에이전트 플랫폼에서 운용되는지에 대한 내용을 더하여 기술한 것으로, 이를 위한 모델링 언어로서 CIL-PS를 정의하여 사용한다. 이러한 CIM-PS의 틀도 CIM-PI로부터 추출되며, 나머지 부분은 개발자가 채운다. 위에서 제시한 CCM에서부터 CIM-PS까지의 모델 변환 과정을 통해서 시스템 구현을 위한 소스 코드의 틀을 생성해낼 수 있어서 실제로 이는 개발자의 프로그래밍 작업량을 현저하게 줄여준다. 즉, 제안된 개발 방식을 이용하면 기존의 거대한 유비쿼터스 시스템을 설계서만 가지고 프로그래밍

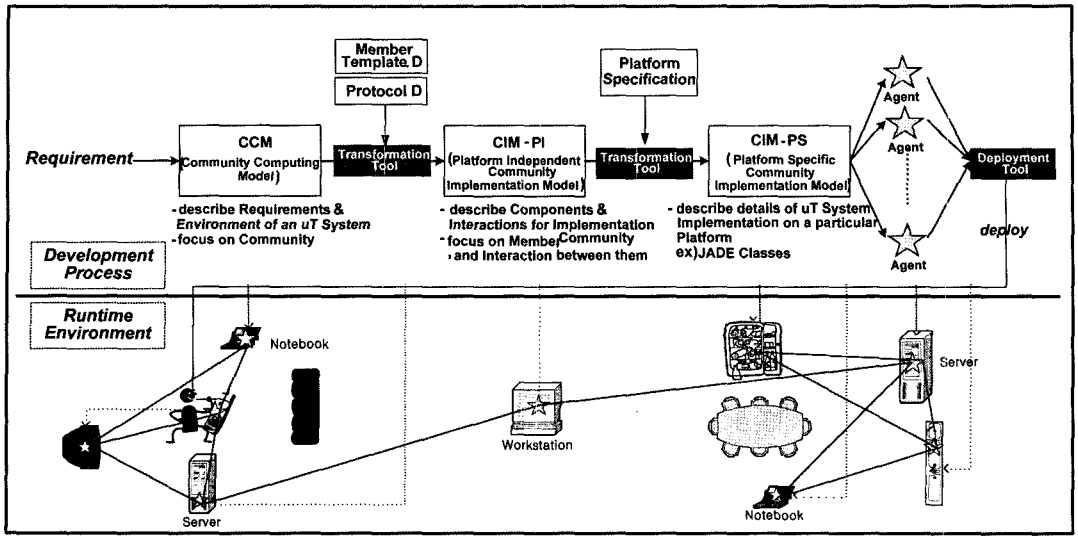


그림 1 커뮤니티 컴퓨팅 시스템의 개발 과정과 운용 환경

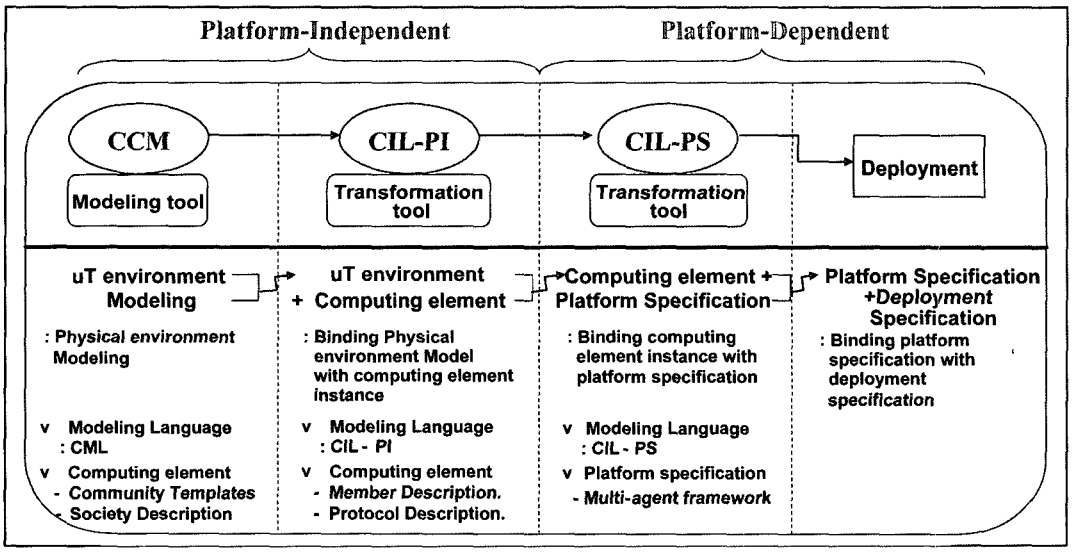


그림 2 커뮤니티 컴퓨팅 시스템 개발을 위한 추상화 모델들

하는 방식에 비해 좀 더 쉽고 체계적으로 시스템을 개발할 수 있다고 하겠다. 게다가 시스템의 전체 개발 과정을 커뮤니티라는 일관된 개념으로 체계적으로 관리할 수 있다는 것도 큰 장점이라고 할 수 있겠다. 개발자는 초기에 시스템을 커뮤니티의 관점에서 기술하는 것으로 유비쿼터스 시스템에서의 협업에만 초점을 맞추어 작업을 진행할 수 있고, 후에 컴퓨팅 요소들의 개별적인 행동에 대하여 초점을 맞추어 작업할 수 있다. 각 요소와 요소들간의 협업을 나누어 고려하면서 개발 작업을 진행함으로써 협업 중심의 유비쿼터스 시스템을 효율적으로 구현할 수 있도록 한다.

4. 커뮤니티 컴퓨팅 시스템의 개발 과정

4.1 커뮤니티 컴퓨팅 모델(Community Computing Model, CCM)

CCM은 커뮤니티 컴퓨팅 시스템에 대한 가장 높은 수준의 추상화 모델로서, 시스템의 환경과 요구사항을 기술하는 것을 목적으로 한다. 하나의 커뮤니티 컴퓨팅 시스템은 컴퓨팅 요소들과 커뮤니티들의집합으로 기술할 수 있다. 따라서 CCM에서는 전체 시스템을 사회(Society)라는 개념으로 추상화하고 사회 기술 부분에 커뮤니티 컴퓨팅 시스템에 존재할수 있는 컴퓨팅 요소

를 표현하고, 요소들간의 협업을 통하여 제공될 수 있는 서비스(해결될 수 있는 요구사항)를 커뮤니티로 표현한다. CCM을 좀 더 자세히 살펴보면, 어떤 에이전트가 커뮤니티의 특정 역할(Role)을 맡기 위한 조건(Cast)을 제시하여 커뮤니티에 대한 에이전트의 소속을 관리한다. 커뮤니티 자체도 자신이 달성할 목적(Goal)을 명시하여, 어떤 에이전트에 의해 그러한 목적이 인지되면 커뮤니티가 생성되고, 이후 목적이 달성되면 커뮤니티가 해체되는 것으로 커뮤니티의 생성과 해체를 관리한다. 또한 시스템 내의 모든 커뮤니티는 해당 사회에 속해있는 요소들로 구성된다고 보고, CCM의 사회기술 부분에서는 사회에 대한 에이전트의 소속을 관리하기 위해 특정 사회 내의 모든 에이전트 멤버가 가져야 하는 조건을 제시하도록 하였다. 이러한 CCM의 구조를 아래의 그림 3에서 나타내어 보았다.

CCM의 구조를 자세히 살펴보기 전에, 먼저 커뮤니티 컴퓨팅에서 사용되는 개념들을 소개한다.

- 멤버(Member): 커뮤니티를 이루는 구성 요소로서, 하나의 에이전트로 구현된다. 컴퓨팅 디바이스, 하드웨어, 사람, 또는 소프트웨어 모두가 멤버가 될 수 있다. 각 멤버는 커뮤니티에서 역할을 맡고 있지 않을 때에는 자신의 작업(예를 들어, 가로등이 빛을 밝힌다던가 시계가 시간을 표시하며 특정 시간에는 알람을 울리는 일등이 이에 속한다)을 수행하고 있다가 특정 목적(Goal)이 생겨 커뮤니티가 구성되면 커뮤니티에 속해 목적을 달성시키기 위해 자신의 역할을 수행한다. 커뮤니티에 속하여 역할을 수행하는 것도 결국은 목적을 달성하기 위한 프로토콜에 맞추어 자신의 고유 작업을 수행하는 것이다. 본 논문에서는 시스템에 존재할 수 있는 멤버의 종류와 이들이 가지는 속성(속성값은 멤버 개체에 따라 다르다)과 고유 작업들이 미리 정해져 있다고 가정하고 이를 모델에서 기술한다.
- 커뮤니티(Community): 여러 에이전트 멤버들이 특정 목적을 달성하기 위하여 형성하는 협업 조직으로서, 하나의 커뮤니티는 구성 멤버들이 맡을 역할(Role), 목적(Goal), 그리고 이를 달성하기 위한 방법을 기술한 프로토콜(Protocol)들로 기술한다. 이 때, 커뮤니티에서 해당 역할을 수행하는 멤버로 선택(cast)될 수 있는 조건을 제시하여 커뮤니티에 대한 멤버 개체(instance)의 소속을 관리한다. 특정 멤버에 의해 목적(goal)이 감지되면 해당 목적을 성취할 수 있는 커뮤니티를 알아내고, 커뮤니티를 구성하는 각 역할에 맞는 멤버들을 선택(Casting)하여 커뮤니티를 생성한다. 그리고, 이들 간의 협업으로 목적이 달성되면 해당 커뮤니티는 해체된다. 좀 더 자세한 내용은 5장에서 실제 시나리오를 바탕으로 기술하도록 하겠다. 본 논문에서는 시스템에 존재할 수 있는 커뮤니티의 종류와 그 커뮤니티의 구성이나 멤버들 간의 협업 방식이 미리 정해져 있다고 가정하고 모델을 작성하였다.
- 목적(Goal): 하나의 커뮤니티가 만족시켜야 할 특정 상황(Situation)으로서, 해당 커뮤니티를 구성하는 이유라고 할 수 있다.
- 프로토콜(Protocol): 커뮤니티 내의 멤버들이 특정 목적(goal)을 달성하기 위해 협업하는 방식을 기술한 것으로서, 여러 종류의 통신 메시지와 각 멤버의 자체 작업 수행들의 나열로 정의된다. 하나의 프로토콜을 수행하는 동안, 또 다른 목적(goal)이 발생하여 그를 달성하기 위한 또 하나의 프로토콜이 수행될 수도 있다.
- 사회 (Society): 전체 커뮤니티 컴퓨팅 시스템을 추상화한 개념이다. 커뮤니티 컴퓨팅 시스템이 처음 운용된 시점에서는 어떠한 커뮤니티도 없이 사회(Society)만 존재하는 형태가 될 것이다. 그 후에 여러 멤버 에이전트들이 그 사회(Society)에 소속됨으로써, 사회 내에 여러 멤버들이 존재하는 형태로 변할 것이다. 이윽고, 커뮤니티를 구성해야 하는 목적(Goal)이 생기면 비로소 커뮤니티가 구성되고 멤버들간의 협업에 의한 서비스가 제공된다. 사회가 소멸되기 전까지 이러한 커뮤니티들은 필요에따라 동적으로 생성되고 소멸된다.

CCM은 커뮤니티를 기술하는 부분과 사회를 기술하는 부분으로 구성된다. 커뮤니티를 기술하는 부분에서는 시스템에 존재 가능한 모든 커뮤니티에 대한 구조와 목적(Goal)들, 그리고 그에 대한 프로토콜들을 정의한다. 이 때, CCM은 CML(Community computing Modeling Language)로 기술하였다. 제안된 모델과 이로부터의 커뮤니티 컴퓨팅 시스템의 개발 과정을 보이기 위해 어린 아이의 위치를 감지하여 집에서가깝고 안전한 범위에서 머물도록 하여 안전을 보장해주는 아이보호 시나리오를 기반으로 소규모의 커뮤니티 컴퓨팅 시스템을 구현하여 보았다. 본 논문의 5장에서 구현에 대한 내용을 자세히 다루도록 하겠다. 시스템 개발을 위한 첫 단계, 먼저 아이보호를 수행할 수 있는 커뮤니티 컴퓨팅 시스템에 대한 CCM을 작성한다. 그림 4에서 작성된 CCM을 보여주고 있다. 보다시피, 시스템에 존재할 수 있는 커뮤니티의 종류는 HOME과 CHILDCARE의 두 가지이다. 각 커뮤니티의 종류에 대해 필요한 역할들, 목적들, 그리고 사용하는 온톨로지를 명시한다. 역할은 그것의 속성들과 그 역할을 맡을 수 있는 멤버의 조건으로 정의하는데, 속성은 속성의 이름과 가능한 값들의 집합으로 표현하는데, 만약 가능한 값의 범위를 알 수 없을 경우에는 생략할 수 있다. 속성들 중에서그 역할로 필요한 멤버의 수를 말해주는 개수 속성은 가장 필수적인 속성으로, 역할 이름 바로 옆에 기술한다. 목적(Goal)은 목

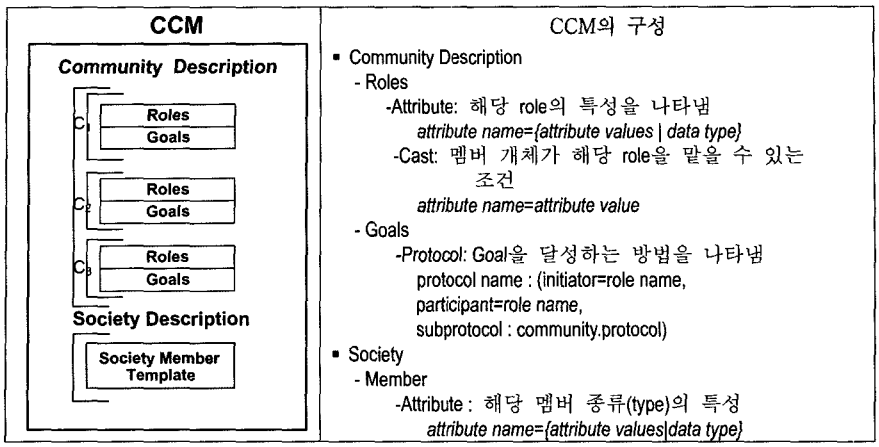


그림 3 CCM의 구성

```
Community Computing Model Description {
  Community Home {
    Role  HOME_STUFF:1~100000 {
      Attribute : ADDRESS; USE;
      Cast : USE=RESIDENTIAL; }
    Role  RESIDENT:1~20 {
      Attribute : LOCATION;
                  ADDRESS=HOME_STUFF.ADDRESS;
      Cast : LOCATION=HOME_STUFF.ADDRESS;   }
    Goal  ANNOUNCE_INFO {
      Protocol
          announce_information_to_human_at_home(info) :
          (Initiator=HOME_STUFF,RASIDENT,
          Participant=HOME_STUFF,RASIDENT)  } }
  Community CHILDCARE {
    Role CHILD:1 {
      Attribute: SAFETY_LEVEL={SAFE|
          LEVEL-1 WARNING| LEVEL-2 DANGER};
      Cast:SAFETY_LEVEL={ LEVEL-1 WARNING|
          LEVEL-2 DANGER};  }
    Role  FAMILY:1~50 {
      Attribute : RELATIONSHIP;
      Cast:RELATIONSHIP={CHILD.MOTHER|
          CHILD.FATHER|CHILD.SISTER|CHILD.BROTHER|
          CHILD.GRANDMOTHER|CHILD.GRANDFATHER};
    Role  NEIGHTBOR:1~20 {
      Attribute : RELATIONSHIP;
      Cast : RELATIONSHIP=CHILD.NEIGHTBOR;   }
    Role  WATCHER:0~2 {
      Attribute : MONITORING SERVICE=
        {AVAILABLE|NOT AVAILABLE}; LOCATION;
      Cast : MONITORING SERVICE=AVAILABLE;
          LOCATION=CHILD.LOCATION; }
    Goal CHILDCARE SERVICE {
      Protocol  take_a_child_home :
        (Initiator=CHILD,
        Participant= FAMILY, NEIGHBOR, WATCHER)} }
  Society  GHODAMCITY{
    Member GHODAMCITIZEN {
      Attribute : LOCATION=GHODAMCITY;
          CITY_ADDRESS=GHODAM;
          SAFETY_LEVEL={SAFE| LEVEL-1 WARNING|
          LEVEL-2 DANGER};
      Cast : ADDRESS.CITY= GHODAM;   }  }  }
```

그림 4 CML로 기술된 아이보호 CCM의 예

적의 이름과 이를 달성하게 해주는 프로토콜로 정의되는데, 이 때 프로토콜은 참여 역할들과 프로토콜 수행 도중에 수행될 수도 있는 또 다른 프로토콜을 명시하는 것으로 표현된다. 그리고 만약 커뮤니티가 특정 온톨로지를 사용한다면 그 이름을 기술하여 시스템 운용 중에 사용할 수 있도록하였다. 사회(Society) 기술 부분에서는 그 사회의 이름과 멤버 개체가 그 사회에 속할 수 있는 조건을 기술한다. 그림 3에서의 예를 보면, GHODAMCITY라는 사회에 소속하기 위해서는 멤버의 ADDRESS.CITY라는 속성의 값이 반드시 GHODAM이어야 한다는 조건을 명시하고 있다.

4.2 플랫폼 독립적 커뮤니티 컴퓨팅 구현 모델(Platform Independent Community Computing Implementation Model, CIM-PI)

CIM-PI는 시스템의 구현을 고려하여 컴퓨팅 요소들과 이들 간의 협업관계를 좀 더 자세히 기술하는 것을 목적으로 한다. 이 때, 특정 에이전트 플랫폼에 국한되지 않는 수준으로 기술하도록 한다. CIM-PI에서는 CCM의 커뮤니티와 사회(Society) 기술을 좀 더 구체적인 내용으로 다듬는데, 커뮤니티 기술 부분에서는 커뮤니티의 특정 역할(Role)을 맡을 수 있는 멤버 타입에 대한 정보와 프로토콜의 세부적인 흐름에 대한 정보를 추가적으로 명시한다. 프로토콜 기술 부분에서는 커뮤니티를 구성하는 각 역할들이 목적(Goal)을 달성하기 위해 수행해야 하는 일들을 기술하는데, 이를 위해 Occam을 기반으로 하는 프로토콜 기술 언어를 따로 정의하여 기술하였다. Occam[11]은 시스템의 구성요소들이 독립적으로 운영되면서 서로 상호작용을 하는 시스템을 구현하기 위한 프

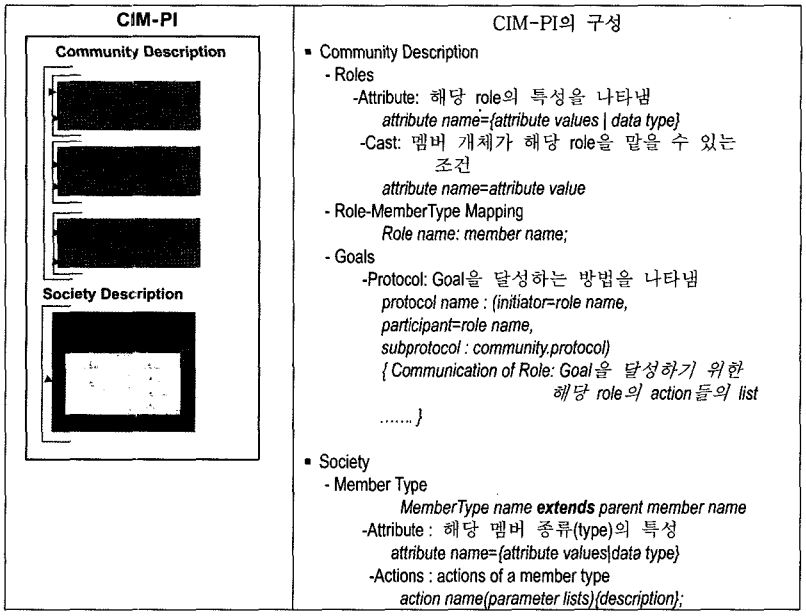


그림 5 CIM-PI의 구성

로그래밍 언어이다. Occam의 가치는 한 시스템에서 병행되는 작업들을 효율적으로 관리할 수 있도록 하는 것으로, 이러한 Occam의 장점을 수용하기 위하여 본 논문에서는 Occam의 구성자(Constructor)들 중에서 SEQ, PAR, ALT, IF, EXIT를 차용하여 커뮤니티의 프로토콜을 기술하였다. 프로토콜을 수행하기 위해 멤버는 자신의 고유 작업을 수행하거나 SEND와 RECEIVE 같은 기본적인 메시지 전송 작업을 수행하는데, 이때 멤버들 간에 주고받는 통신 메시지의 종류는 FIPA(Foundation for Intelligent Physical Agents)[15]가 정한 메시지종류로 한정하여 사용하였다. 제안된 모델에서는 프로토콜을 기술할 수 있는 구성자, 기본 메시지 전송 함수, 그리고 주고받을 수 있는 메시지 종류만을 제공할뿐 실제 프로토콜을 설계하여 작성하는 것은 프로토콜 설계자이다. 본 논문에서는 에이전트들 간의 협업을 효과적으로 하기 위해 어떻게 프로토콜을 작성해야 하는지는 고려하지 않으며, 이는 프로토콜 설계자가 효과적으로 기술한다고 가정한다. CIM-PI의 사회(Society) 기술 부분에서는 시스템에 존재할 수 있는 모든 멤버 타입들을 기술하는데, 실제 시스템이 운영되면 각 컴퓨팅 요소 즉, 하드웨어, 소프트웨어, 또는 사람 들은 특정 멤버 타입의 한 개체(instance)로서 표현된다. 또한 멤버 타입간의 계층 관계를 표현하기 위하여 *extends*라는 키워드를 사용하였는데, 여기에서 계층관계의 의미는 하위 멤버가 상위 멤버의 속성을 모두 상속받는 것을 의미한다. 예를 들어, Streetlamp가 Electronic Appliance의 하위 멤버라는 것은 Streetlamp가 Electronic Appliance의 속성을 모두 상속받아 자신의 속성으로 가진다는 것이다. 이러한 CIM-PI의 구조는 아래의 그림 5에서 볼 수 있으며, 이러한 CIM-PI를 기술하기 위해 CIL-PI(Platform Independent Community Computing Implementation Language)를 정의하여 사용하였으며, 이에 대한 예는 아래의 그림 6에서 볼 수 있다.

4.3 플랫폼 기반 커뮤니티 컴퓨팅 구현 모델(Platform Specific Community Computing Implementation Model, CIM-PS)

CIM-PS는 CIM-PI를 특정 플랫폼 환경에서 어떻게 구현할 것인지를 기술한 것으로, 여기에서의 플랫폼은 JAM[16]이나 JADE[17]와 같은 에이전트 플랫폼을 의미한다. 즉, CIM-PI를 특정 플랫폼에 맞게 기술한 것이 CIM-PS이다. 예를 들어, 개발자가 JADE 플랫폼에서 시스템을 구현하고자 한다면, CIM-PI를 JADE 코드로 변환시켜 CIM-PS를 작성하고, JAM 플랫폼을 선택하면 CIM-PI를 JAM 파일로 구현하여 CIM-PS를 작성한다. JADE 플랫폼을 이용하여 시스템을 구현하는 경우, 하나의 멤버는 하나의 JADE 에이전트로 구현되며, 멤버가 협력을 위해 진행되어야 할 프로토콜들은 각각 JADE의 행동(Behavior)으로 구현되어 에이전트의 기능을 수행할 수 있도록 한다. CIM-PI를 JADE 플랫폼을 기반으로 하는 CIM-PS로 변환시키는 자세한 과정은 아래의 표 1에서 보이고 있다.

```
Platform Independent Community Implementation Description {
  Community Home {
    Role   HOME_STUFF:1~100000 {
      Attribute : ADDRESS; USE;
      Cast : USE=RESIDENTIAL; }
    Role   RESIDENT:1~20 {
      Attribute : LOCATION;
                  ADDRESS=HOME_STUFF.ADDRESS;
      Cast : LOCATION=HOME_STUFF.ADDRESS;   }
  HOME_STUFF:Household_appliance;   RASIDENT:Human ;
   Protocol announce_information_to_human_at_home {
    Communication of Initiator {
        SEND(MsgType="inform", ToWhom=Participant,
                  InformedData); }
    Communication of Participant {
        IF(RECEIVE(MsgType="inform",FromWho=Initiator,InformedData))
          Display_Info(InformedData);
        END IF}    }   }
  Community CHILDCARE {
    Role CHILD:1 {
      Attribute: SAFETY_LEVEL={SAFE|
          LEVEL-1 WARNING| LEVEL-2 DANGER};
      Cast:SAFETY_LEVEL={ LEVEL-1 WARNING|
          LEVEL-2 DANGER};   }
    Role   FAMILY:1~50 {
      Attribute : RELATIONSHIP;
      Cast:RELATIONSHIP={CHILD.MOTHER|
        CHILD.FATHER|CHILD.SISTER|CHILD.BROTHER|
        CHILD.GRANDMOTHER|CHILD.GRANDFATHER};
    Role    NEIGHTBOR:1~20 {
      Attribute : RELATIONSHIP;
      Cast : RELATIONSHIP=CHILD.NEIGHTBOR;   }
    Role   WATCHER:0~2 {
      Attribute : MONITORING SERVICE=
        {AVAILABLE|NOT AVAILABLE}; LOCATION;
      Cast : MONITORING SERVICE=AVAILABLE;
          LOCATION=CHILD.LOCATION; }
  CHILD:Human,Smartbelt ; FAMILY:Human ; NEIGHBOR:Human ;
  WATCHER:Camcoder,Camera,Streetlamp;

   Protocol take_a_child_home {
          communication of child {
   .....

Society: GHODAMCITY {
  Member Society Member {
   Attribute :   LOCATION=GHODAMCITY;
                 CITY_ADDRESS=GHODAM;
              SAFETY_LEVEL={SAFE|   LEVEL-1   WARNING|
              LEVEL-2 DANGER};
   Action :    Wait_for_Msg(MsgType="inform", FromWho,
            InformedData);   }
  Member Animate Object extends Society Member {
   Attribute : SPECIES=STRING; GENUS=STRING;
              FAMILY=STRING; }
  Member Human extends Animate Object {
   Attribute : SEX={MALE|FEMALE}; AGE=(0~150) ;
        RELATIONSHIP=STRING; JOB=STRING;
   Actions : Choose_the_nearest_family(Location, NearestFamily);
          Request_for_picture(RequestWho=WATCHER.id,
              Location, RequestedPicture);
          Take_to(Who, Where);
          Choose_the_nearest_person(Location,NearestPerson);
          Choose_responce(Choice1, Choice2, Choice); }
  Member Inanimate Object extends Society Member {
   Attribute : STATUS=STRING;   }
  Member Electronic Appliance extends Inanimate Object {
   Attribute : ELECTRONIC_POWER=STRING ;
        WEIGHT=INTEGER; HIGHT=INTEGER;
        USAGE={HOME|INDUSTRY|RESEARCH};   }
  Member Home Appliance extends Electronic Appliance {
   Attribute : USAGE=HOME;
        ASSIGNED_ROOM={LIVINGROOM|KITCHEN|BEDROO
        M|BATH|READING};
   Actions : Display_Info(InformedData);   }
  Member Streetlamp extends Electronic Appliance {
   Attribute : MONITORING SERVICE={AVAILABLE|NOT
              AVAILABLE};
            LIGHTENING={YES|NO};
    Actions : Send_picture(ToWhom, RequestedPicture);}}}
```

그림 6 CIL-PI로 기술된 아이보호 CIM-PI의 예

표 1 CIM-PI에서 CIM-PS(JADE)로의 변환

CIM-PI		CIM-PS (JADE)	설명
Community_Template		Agent	각 커뮤니티는 Jade 에이전트로 변환되고, 이 에이전트 내에서 해당 커뮤니티의 속성 및 멤버 정보 등이 필드와 메서드로 변환됨.
Member_Role_Binding		Java Class	멤버 타입의 역할로의 바인딩을 위해서 별도의 클래스 템플릿을 작성한 후 변환. 이러한 java클래스는 커뮤니티 에이전트에 역할을 부여하기 위한 필드와 이와 관련된 메서드로 변환됨.
Communicative_Action	SEND	Agent의 Method	메시지를 보내는 메서드 템플릿을 만들고 멤버 에이전트가 수행할 프로토콜(behavior) 클래스의 메서드로 변환
	RECEIVE	Agent의 Method	메시지를 받는 메서드 템플릿을 만들고 멤버 에이전트가 수행할 프로토콜(behavior) 클래스의 메서드로 변환
Construct	WAIT	-	별도의 메서드를 생성하지 않고 JADE의 통신 메커니즘을 이용하여 block() 상태로 다음 행동까지 대기상태로 만듬
	EXIT	return statement	현재 수행하고 있는 Behavior를 중지하고 return 문으로 변환
	SEQ	-	Behavior내에 순차적으로 SEQ 블럭을 변환(일반 코드)
	PAR	Java Thread	PAR 블록마다 자바 Thread로 변환
	ALT	IF statement	if else 문으로 구성하여 각 블록 변환
Society_Description		Agent	Society는 Jade 에이전트로 변환되며, 이 에이전트 내에 커뮤니티 템플릿 정보 및 멤버 템플릿 정보 등을 변환
Member_Action_Description		Cyclic Behavior	멤버가 가지는 행동을 멤버 에이전트가 수행하는 Behavior로 정의하도록 변환

본 논문에서는 JADE 플랫폼에서 아이보호 시스템을 구현하여 이를 시뮬레이션으로 보여주고자 한다. 이 때, 사용되는 CIM-PS의 일부를 아래의 그림 7에서 각 GHODAMCITY 사회, HOME 커뮤니티, HomeApplicance 멤버 타입, CHILDCARE 커뮤니티의 아이보호 목적을 달성하기 위한 Family 역할이 수행하는 take_a_child_home 프로토콜에 대한 CIM-PS로서 나타내어 보았다. 이때 개발자가 추가하거나 수정한 코드는 음영으로 표시하여 개발도구에 의해 생성되는 코드와 구별하였다.

```
package GHODAMCITY.agent;
import jade.core.Agent;
import jade.core.behaviours.*;
import jade.domain.*;
import jade.domain.FIPAAgentManagement.*;
import java.util.*;
import GHODAMCITY.behaviour.*;
import GHODAMCITY.utility.*;
public class GHODAMCITY extends SocietyTemplate {
   private HashMap memberList = new HashMap();
    private ArrayList communityList = new ArrayList();
   protected String societyType;
   protected String societyName;
   public GHODAMCITY() {
            super();
            societyType = "SOCIETY";
            societyName = "GHODAMCITY";      }
protected void setup() {
            DFAgentDescription dfd = new DFAgentDescription();
            dfd.setName(getAID());
            dfd.addServices(setServiceDescription());
            try {
                 DFService.register(this, dfd);
   } catch (FIPAException fe) {
                      fe.printStackTrace();          }
    addBehaviour(new WaitingRegistration(this));
      addBehaviour(new WaitingCommunityCreation());
      setCommunities(); }
public void addMember(final String memberName, final String key) {
       addBehaviour(new OneShotBehaviour() {
                  public void action() {
                       memberList.put(memberName, key);
                  } } );    }
  protected ServiceDescription setServiceDescription() {
     ServiceDescription serviceDesc =
          new ServiceDescription();
    serviceDesc.setType(societyType);
      serviceDesc.setName(societyName);
      return serviceDesc; }
 protected void takeDown() {
   try {
            DFService.deregister(this);
   } catch (FIPAException fe) {
       fe.printStackTrace();      } }
 public void setCommunities() {
   communityList.add("HOME");
    communityList.add("CHILDCARE"); }    }
```

(a) GHODAMCITY 사회에 대한 CIM-PS, GHODAMCITY.java

```
package GHODAMCITY.agent;
import jade.core.*;
import jade.core.behaviours.*;
import java.util.ArrayList;
import jade.domain.FIPAAgentManagement.*;
import jade.domain.*;
import GHODAMCITY.behaviour.*;
import GHODAMCITY.utility.*;
public class HOME extends CommunityTemplate {
      protected AID initiator;
      protected String communityName;
      protected String communityType;
      protected ArrayList members = new ArrayList();
      protected RoleBinding[] roleBinding;
  public HOME() {
      super();
      communityType = "COMMUNITY";
      communityName = "HOME";   }
protected void setup() {
    DFAgentDescription dfd = new DFAgentDescription();
    dfd.setName(getAID());
    dfd.addServices(setServiceDescription());
     try {
          DFService.register(this, dfd);
    } catch (FIPAException fe) {
          fe.printStackTrace();   }
    addBehaviour(new MemberCasting(this));
    addBehaviour(new SendProtocol(this)); }
protected void addProtocol(String protocolName) {
    try {
      SimpleBehaviour behaviour =
public ArrayList getMemberList() {
    return members;
}
public String getProtocol(AID memberAID) {
    for (int i = 0; i < roleBinding.length; i++) {
      AID[] aids = roleBinding[i].getMemberAIDs();
      for (int j = 0; j < aids.length; j++) {
        if (aids[j].equals(memberAID)) {
          return roleBinding[i].getProtocol();
    } } }
     return null;
}
public RoleBinding[] getRoleBinding() {
    return roleBinding;
}
protected ServiceDescription setServiceDescription() {
    ServiceDescription serviceDesc = new ServiceDescription();
    serviceDesc.setType(communityType);
    serviceDesc.setName(communityName);
    return serviceDesc;
}
protected void setRoleBindings() {
    roleBinding = new RoleBinding[2];
    roleBinding[0] = new RoleBinding("HOME_STUFF",
          "Household_appliance",   "CommunicationofInitiator");
    roleBinding[1] = new RoleBinding("RASIDENT", "Human",

"CommunicationofParticipant");
}
protected void takeDown() {
    try {
```

```
        (SimpleBehaviour) Class.forName(protocolName).newInstance();
            addBehaviour(behaviour);
    } catch (ClassNotFoundException e) {
      System.out.println(" Behaviour Not Found : " + protocolName);
    } catch (Exception e) {
          e.printStackTrace();
}   }
```

```
      DFService.deregister(this);
    } catch (FIPAException fe) {
      fe.printStackTrace();
    }  }
public void updateMemberList(AID member) {
    members.add(member);
}}
```

(b) HOME 커뮤니티에 대한 CIM-PS, HOME.java

```
package GHODAMCITY.agent;
import jade.core.*;
import jade.lang.acl.*;
import jade.domain.*;
import jade.domain.FIPAAgentManagement.*;
import jade.core.behaviours.*;
import java.lang.reflect.Constructor;
import java.util.*;
import GHODAMCITY.behaviour.*;
import GHODAMCITY.utility.*;
public class HomeAppliance extends ElectronicAppliance{
    protected String memberName;
    protected HashMap communityMembers = new HashMap();
    private String usage = "HOME";
    private SelectiveValue assigned_room = new SelectiveValue
          ("LIVINGROOM|KITCHEN|BEDROOM|BATH|READING");
    public HomeAppliance() {
      super();
      memberType = "HomeAppliance";
      memberName = "HomeAppliance";
    }
protected void setup() {
    DFAgentDescription dfd = new DFAgentDescription();
    dfd.setName(getAID());
    dfd.addServices(setServiceDescription());
    try {
      DFService.register(this, dfd);
    }  catch (FIPAException fe) {
      fe.printStackTrace();
    }
    addBehaviour(new Registration());
    addBehaviour(new ResponseToCasting());
    addBehaviour(new ReceiveProtocol(this));
    addBehaviour(new CommunityCreation());
    addBehaviour(new ReceiveMemberList());}
```

```
protected void takeDown() { }
public void addProtocol(String protocolName) {
    try {
      Class c = Class.forName(protocolName);
      Constructor con = c.getDeclaredConstructors()[0];
      addBehaviour((SimpleBehaviour) con.newInstance
                  (new Object[] {this}));
    } catch (ClassNotFoundException e) {
      System.out.println(" Behaviour Not Found : " + protocolName);
    } catch (Exception e) {
      e.printStackTrace(); }}
protected ServiceDescription setServiceDescription() {
    ServiceDescription serviceDesc = new ServiceDescription();
    serviceDesc.setType(memberType);
    serviceDesc.setName(memberName);
    return serviceDesc; }
public boolean done() {
    return false;  }
public void setCommunityMember(String memberDesc) {
    StringTokenizer st = new StringTokenizer(memberDesc, ";");
    while(st.hasMoreTokens()){
      communityMembers.put(st.nextToken(),
          new AID(st.nextToken(), true));  }}
public AID getCommunityMemberAID(String memberName) {
    return (AID)communityMembers.get(memberName);  }
public String getPackagePath() {
    return "GHODAMCITY.agent"; }
public String Display_Info(String InformedData) {
    String returnString = new String();
    return returnString; }
public class Display_Info extends SimpleBehaviour {
    public Display_Info (String InformedData) {  }
    public void action() {  }
    public boolean done() {
    return false; } }    }
```

(c) HomeApplicance 멤버 타입에 대한 CIM-PS, HomeAppliance.java

```
package GHODAMCITY.behaviour;
import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.*;
import jade.domain.*;
import jade.domain.FIPAAgentManagement.*;
import GHODAMCITY.utility.*;
import GHODAMCITY.agent.*;
import GHODAMCITY.behaviour.*;
public class CommunicationofFAMILY extends SimpleBehaviour {
    private int status = 0;
    private MemberTemplate owner;
    private String child = "CHILD";
    private String choice = "CHOICE";
    private String child_location = "CHILD_LOCATION";
    private String nearestperson = "NEIGHBOR";
    private String neighbor = "NEIGHBOR";
    public CommunicationofFAMILY(MemberTemplate member) {
```

```
    else if (receiveMsg(getMsgType("inform"), getAgent(neighbor)) != null) {
          done();
      break; } }}
private String receiveMsg(int type, AID sender) {
    MessageTemplate template;
    ACLMessage msg;
    template=
     MessageTemplate.and(MessageTemplate.MatchPerformative(type),
        MessageTemplate.MatchSender(sender));
    while(true) {
      msg = myAgent.receive(template);
      if(msg != null) {
return msg.getContent();      }  } }
private void sendMsg(int type, AID receiver, String parameter) {
    ACLMessage msg = new ACLMessage(type);
    msg.addReceiver(receiver);
    msg.setContent(parameter);
    myAgent.send(msg);}
```

```
    owner = member;  }
public void action() {
  while(true) {
   if (receiveMsg(getMsgType("inform"), getAgent(child)) != null) {
   invokeMethod("Choose_responce",  new  String[] {"IGNORE",
    "CHILDCARE", choice});
    choice = "CHILDCARE";
    if (choice == "IGNORE") {
    sendMsg(getMsgType("inform"), getAgent("CHILD"), "IGNORE");
    done();
    } else if (choice == "CHILDCARE") {
     invokeMethod("Choose_the_nearest_person", new String[]
     {child_location, nearestperson});
    sendMsg(getMsgType("propose"), getAgent(nearestperson),
    "CHILDCARE" + "," + child_location);                    }
  } else if (receiveMsg(getMsgType("reject-proposal"),
     getAgent(child))  = null) {
     invokeMethod("Choose_the_nearest_person",new String[]
            {child_location, nearestperson});
     sendMsg(getMsgType("propose"), getAgent(nearestperson),
     CHILDCARE" + "," + child_location);
  } else if (receiveMsg(getMsgType("accept-proposal"),
       getAgent(neighbor)) != null) {
       invokeMethod("Wait_for_Msg", new String[] {"inform",
       neighbor,"DONE"});     }
```

```
private AID getAgent(String agentType) {
    return
    ((MemberTemplate)myAgent).getCommunityMemberAID(agentType);}
private int getMsgType(String type) {
    return ACLMessage.getInteger(type); }
public boolean done() {
    return true; }
private String invokeMethod(String methodName, String[] args) {
    try{
      Class[] ARGS_CLASSES = new Class[args.length];
      for(int i=0; i<args.length;i++) {
      ARGS_CLASSES[i] = String.class;
    }
    Class c = Class.forName(owner.getPackagePath() + "." +
         owner.getMemberType());
    java.lang.reflect.Method m =
         c.getMethod(methodName, ARGS_CLASSES);
    return (String)m.invoke(owner, args);
    } catch (Exception e) {
    e.printStackTrace();
    }
    return null;
}
public String getPackagePath() {
 return "GHODAMCITY.behaviour";    }}
```

(d) CHILDCARE 커뮤니티의 아이보호 목적을 달성하기 위한 Family 역할이 수행하는 take_a_child_home 프로토콜에 대한 CIM-PS, CommunicationofFamily.java

그림 7 JADE 플랫폼에서 구현한 아이보호 CIM-PS

4.4 모델 변환

MDA 방식을 적용하여 커뮤니티 컴퓨팅 시스템을 개발하기 위해서는 고 수준의 추상화 모델에서 저 수준의 구현 모델까지의 모델 변환 과정을 거친다. 아래의 그림 8에서 모델을 변환시켜 가면서 실제 시스템 구현을 위한 소스 코드를 생성해 내는 과정을 도식화하여 보았다. 먼저, 구현하고자 하는 커뮤니티 컴퓨팅 시스템을 CCM으로 기술하고, 이를 변환하여 CIM-PI의 틀을 잡고 나머지 부분은 멤버 타입 정의(Member Type Description), 멤버간의 계층 정보(Member Type Hierarchy), 역할을 맡을 수 있는 멤버 타입 정의(Role-Member Binding), 프로토콜(Protocol Description) 등의 추가적인 정보를

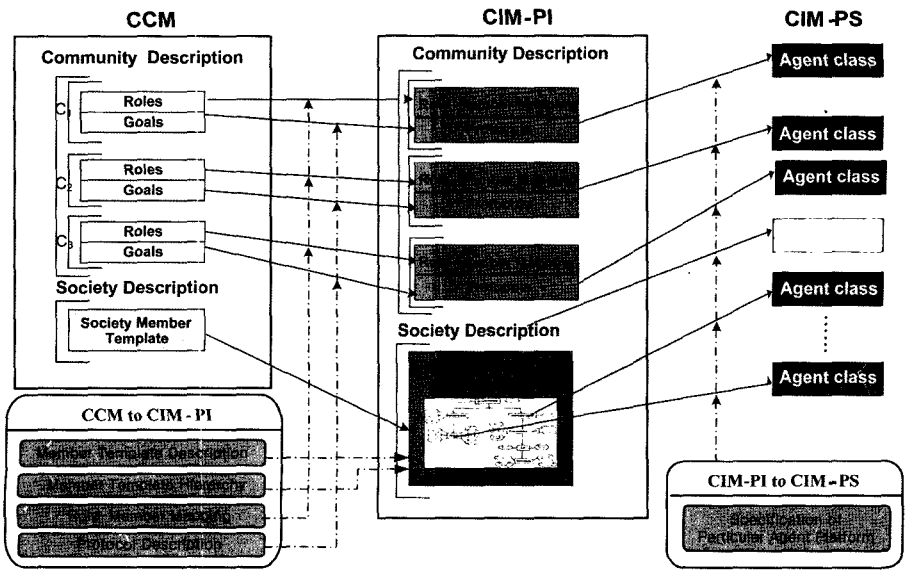


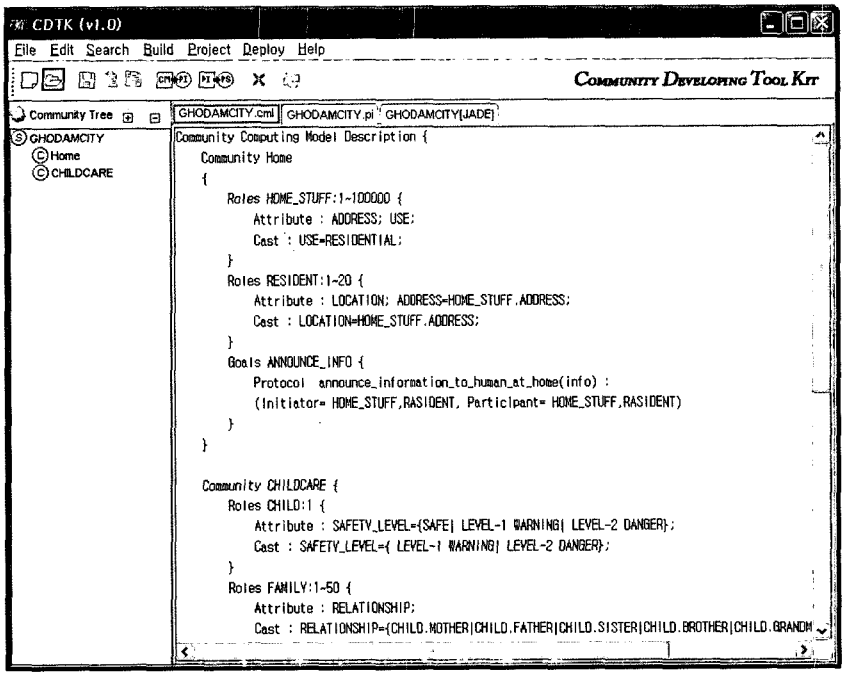
그림 8 CCM에서부터 CIM-PS까지의 모델 변환 과정

채워서 CIM-PI를 완성한다. 이 때 CCM의 커뮤니티 기술 부분에 있는 역할 기술은 CIM-PI에서 역할을맡을 수 있는 멤버 타입을 기술하고 목적(Goal)기술은 CIM-PI에서 이를 위한 자세한 프로토콜을 기술하는 것으로 확장된다. 또한 CCM의 사회(Society) 기술에서 사회에 속한 모든 멤버의 상위 멤버가 되는 '사회 멤버'만 기술한 것에서 CIM-PI에서는 이러한 '사회 멤버'로부터 상속받아 존재하는 모든 시스템의 멤버 타입을 기술하는 것으로 확장된다. 그 다음 단계로서, CIM-PI는 시스템이 구현될특정 플랫폼의 정보를 받아 CIM-PS로 변환되는데 이 과정에서 CIM-PS의 대략적인 틀이 생성되고 각 멤버의 자체 작업에 대한 프로그래밍 부분이나 그 외에 수정해야 할 부분은 플랫폼에 맞게 개발자가 채운다. 완성된 CIM-PS는 각 멤버 타입, 각 커뮤니티 타입, 그리고 하나의 사회(Society)에 대한 에이전트를 기술하게 된다. 이러한 코드는 실제 시스템의 운용을 위하여 각 컴퓨팅 요소에 배치되고 이러한 에이전트 개체들이 시스템을 구성하는 것이다.

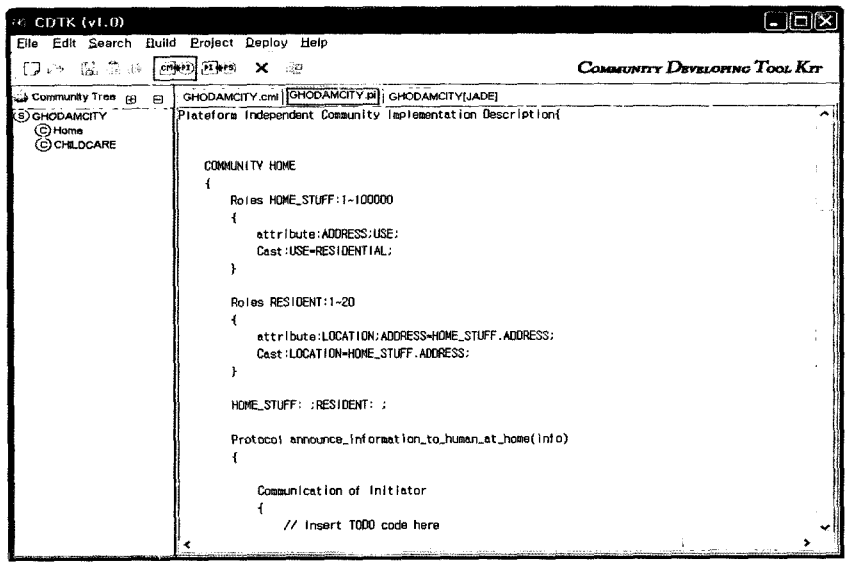
5. 시스템 구현

본 논문에서는 MDA 방식을 이용한 커뮤니티 컴퓨팅 시스템의 개발을 좀 더 편리하게 하기 위해 시스템 개발 도구(Community computing system Development ToolKit, CDTK)를 구현하였다. 개발자가 시스템을 CCM로 기술하면(그림 9(a)), 커뮤니티 컴퓨팅 시스템 개발 도구를 이용하여 이를 CIM-PI로 변환한다(그림 9(b)). 이러한 변환 작업을 통하여 생성한 CIM-PI의 틀에 개발자가 추가정보를 입력하여 CIM-PI를 완성시킨다. 그 다음 완성된 CIM-PI를 CIM-PS로 변환시켜 협업에 관련된 에이전트 코드를 생성한다. 그러나 협업에 사용되는 각 멤버들의 고유 작업은 개발자가 추가로 프로그래밍 해야 한다. 이러한 추가 작업과 자동 생성된 코드에 대한 약간의 수정을 가하여 자바 파일들을 완성한다(그림 9(c)). 본 논문에서 개발한 아이보호 커뮤니티 컴퓨팅 시스템은 JADE 플랫폼을 기반으로 하였다. JADE(Java Agent Development Framework)[17]는 FIPA 표준을 준수하는 다중 에이전트 시스템을 개발하기 위한 소프트웨어 개발 프레임워크로서 현재 많은 에이전트 시스템의 개발을 위하여 사용되고 있는 플랫폼이다.

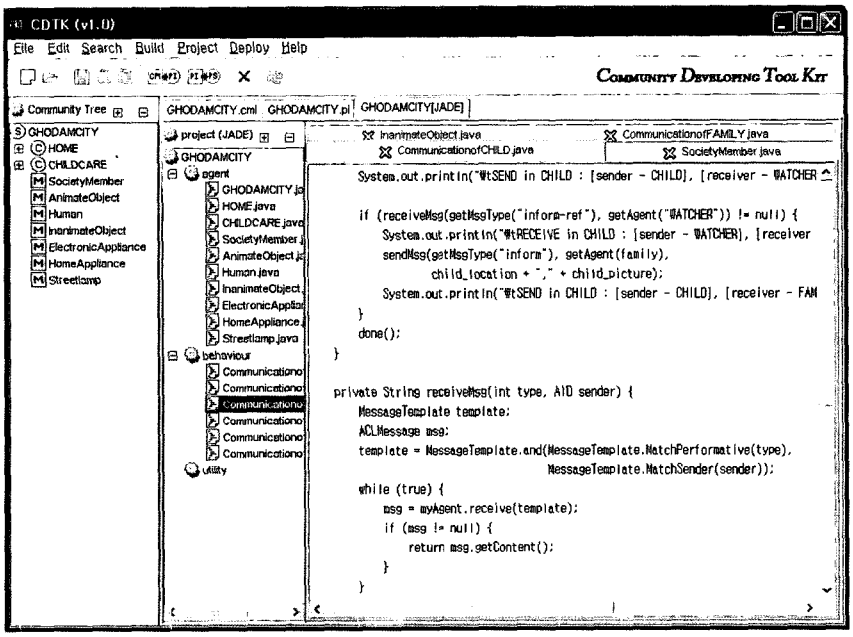
커뮤니티 컴퓨팅 시스템을 개발하고 나서 이를 각 컴퓨팅 요소에 배치한 다음, 실제 시스템을 운용하기 위해서는 시스템 운영을 위한 계산 모델이 필요하다. 따라서 본 논문에서는 커뮤니티 컴퓨팅 시스템의 계산 모델을 아래의 그림 10과 같이 제안한다. 제안한 계산 모델에 의한 시스템 운용은 다음과 같은 흐름을 보일 것이다. 먼저, 시스템을 처음 시작할 당시에는 사회(Society)를 대표하는 사회 관리자(Society Manager)와 각 컴퓨팅



(a) GHODAMCITY CCM



(b) GHODAMCITY CCM으로부터 변환된 CIM-PI



(c) GHODAM CIM-PI로부터 변환된 CIM-PS

그림 9 CDTK를 이용한 커뮤니티 컴퓨팅 시스템의 개발

요소를 표현하는 여러 종류의멤버 에이전트들만이 존재한다. 이 때, 사회 관리자는 시스템에 존재 가능한 모든 커뮤니티 타입과 멤버 타입, 그리고 자신에 속할 수 있는 멤버의 조건 등을 저장하고 있다. 각 멤버 에이전트는 자신의 속성값과 자신이 수행할 수 있는 일(Action)을 가지고 있으며, 특정 커뮤니티에 소속될 경우 목적 달성을 위해 자신의 역할을 수행할 수 있는 구조를 가진다. 시스템이 시작된 이후, 각 멤버 객체들은 사회(Society)에 등록하고 사회 관리자는 이러한 멤버들의 정보를 유지한다. 그러다가 어떤 멤버 에이전트가 커뮤니티의 협업으로 해결해야 하는 문제를 인지하면 이를 사회 관리자에게 보고하고, 사회관리자는 이 문제를 해

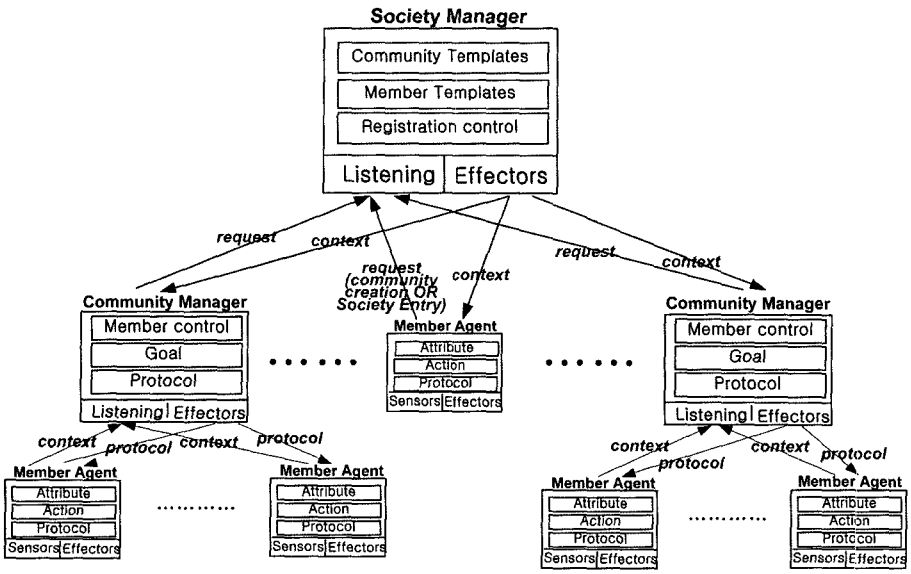


그림 10 커뮤니티 컴퓨팅 시스템의 계산 모델

결할 수 있는 커뮤니티 관리자(Community Manager)를 생성한다. 생성된 커뮤니티 관리자는 각 역할을 맡을 수 있는 멤버 에이전트 객체를 모아서커뮤니티를 구성하고, 각 멤버 개체에게 해당 프로토콜들을 분배하여 멤버 개체간의 협업으로 목적을 달성하도록 한다. 이를 위해 커뮤니티 관리자는 특정 역할을 맡을 수 있는 멤버 개체의 조건과 달성해야 할 목적, 그리고 프로토콜에 대한 정보를 저장하고 있다. 프로토콜을 성공적으로 수행하여 목적이 달성되면 커뮤니티 관리자가 이를 사회 관리자에게 알리고 커뮤니티를 해체시킨다.

본 논문에서 제안한 커뮤니티 컴퓨팅의 모델과 시스템 개발 과정에 대한 실현성과 효율성을 검증해보기 위하여 앞서 예로 보인 아이보호 시나리오를 기반으로 소규모의 커뮤니티 컴퓨팅 시스템을 개발하여 아이보호 시나리오를 시뮬레이션 하였다. 아래의 그림 9에서 시뮬레이션의 한 장면을 보이고있다. 구현된 커뮤니티 컴퓨팅 시스템에서 아이보호 시나리오가 수행되는 과정을 자세히 설명하면 다음과 같다. CDTK를 이용하여 생성된 에이전트 코드를 각 컴퓨팅 요소(실험에서는 노트북과 데스크톱을 이용)에 배치한 후 시스템을 시작시킨다. 시스템이 시작되면 먼저 개발된 시스템을 위한 사회 관리자(Society Manager)가 생성되고, 자신의 존재를 시스템에 속하는 모든 컴퓨팅 요소에게 생성 직후부터 주기적으로 알린다. 사회 관리자의 메시지를 받은 각 요소들은 해당 사회에 속할 것인지를 결정하고, 만약 속하고자 한다면 사회 관리자에게 자신의 존재를 등록한다. 이러한 과정으로 사회 관리자는 자신의 사회에 속하는 컴퓨팅 요소, 즉 사회의 멤버들에 대한 정보(멤버 타입, 현재 속해있는 커뮤니티, 위치 정보, 멤버의 속성값 등)를 저장하고 주기적인 업데이트를 통하여 최근 정보를 유지한다. 만약, 멤버의 상태가 해당 사회의 멤버가 되는 조건에 맞지 않거나 혹은 멤버가 탈퇴를 원할 경우에는 사회 관리자의 멤버 리스트에서 삭제되며 더 이상 멤버로서 관리되지 않는다. 멤버들의 등록이 끝나고 아직은 어떠한 목적도 생성되지 않아 커뮤니티가 존재하지 않는 상태에서 아이가 집을 나선다. 이때 아이의 엄마는 아이의 정보를 담은 스마트벨트(Smartbelt)를 채워주고 그때부터 스마트벨트는 유비쿼터스 시스템에서 아이를 대변하는 디바이스가 된다. 집을 나선 아이가 근처의 안전 범위 내에 있다가 주택가를 벗어나 차가 다니는 위험 범위로 들어가게 되면, 스마트벨트가 현재 위치가위험 범위임을 감지하고, '아이보호' 목적을 생성시켜 이를 사회 관리자에게 알린다. 스마트벨트의 메시지를 받은 시스템의 사회 관리자는 '아이보호' 목적을 달성할 수 있는 아이보호 커뮤니티가 필요하다고 판단하고 이를 위한 커뮤니티 관리자(Community Manager)를 생성하여 사회 관리자가 저장하고 있는 아이보호 커뮤니티 정보를 전달한다. 생성된 커뮤니티 관리자는 아이보호 커뮤니티의 역할 정보를 보고 커뮤니티를 생성하기 위해 필요한멤버들의 수와 각 멤버의 조건을파악하고, 사회 관리자가 저장하고 있는 멤버 정보를 참조하여 조건에 맞는 멤버들에게 아이보호 커뮤니티에 특정 역할로 참여해 줄 수 있는지를 묻는 멤버 캐스팅(Member Casting)을 수행한다. 시나리오의 아이보호

Scene#1	Member 등록

Description

커뮤니티 컴퓨팅에 참여하는 멤버는 사회(Society)에 등록되어야 한다. 사회에 등록된 멤버는 사회 및 커뮤니티, 그리고 사회의 다른 멤버와 협력할 수 있다. 아이 보호 시나리오에서, 아이와 아이 가족, 이웃, 각종 개체들이 'GHODAMCITY' 사회에 등록한다.

CIM-PI Script

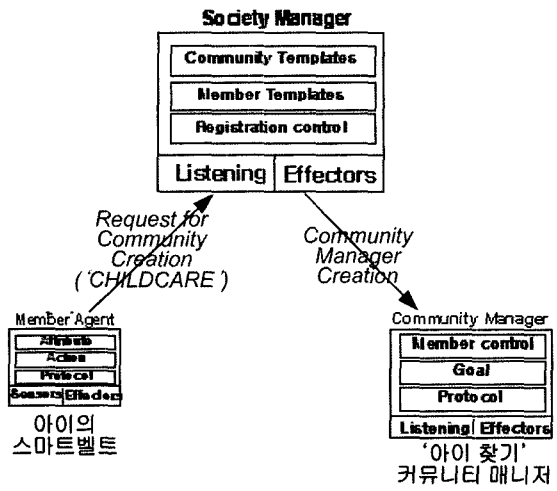
```
Society: GHODAMCITY {
  Member Society Member {
  Attribute :  LOCATION=GHODAMCITY;
               CITY_ADDRESS=GHODAM;
        SAFETY_LEVEL={SAFE| LEVEL-1 WARNING| LEVEL-
        2 DANGER};
Action :    Wait_for_Msg(MsgType="inform", FromWho,
        InformedData);  }
  Member Animate Object extends Society Member {
Attribute : SPECIES=STRING; GENUS=STRING;
               FAMILY=STRING; }
  Member Human extends Animate Object {
Attribute : SEX={MALE|FEMALE}; AGE={0~150) ;
    RELATIONSHIP=STRING; JOB=STRING;
Actions : Choose_the_nearest_family(Location, NearestFamily);
      Request_for_picture(RequestWho=WATCHER.id,
            Location, RequestedPicture);
          Take_to(Who, Where);
      Choose_the_nearest_person(Location,NearestPerson);
      Choose_responce(Choice1, Choice2, Choice); }
```

```
  Member Inanimate Object extends Society Member {
    Attribute : STATUS=STRING;  }
  Member Electronic Appliance extends Inanimate Object {
Attribute : ELECTRONIC_POWER=STRING ;
          WEIGHT=INTEGER; HIGHT=INTEGER;
          USAGE={HOME|INDUSTRY|RESEARCH};  }
  Member Home Appliance extends Electronic Appliance {
Attribute : USAGE=HOME;
ASSIGNED_ROOM={LIVINGROOM|KITCHEN|BEDROOM|BATH|
READING};
Actions : Display_Info(InformedData);  }
  Member Streetlamp extends Electronic Appliance {
Attribute : MONITORING SERVICE={AVAILABLE|NOT
            AVAILABLE};
          LIGHTENING={YES|NO};
Actions : Send_picture(ToWhom, RequestedPicture);}
 }
}
```

Scene#2	Community 생성 요청

Description

아파트 단지에서 놀고 있던 아이가 안전한 구역을 벗어나자, 아이가 차고 있는 스마트 벨트가 이를 감지하여, 아이를 안전하게 집으로 데려가기 위하여 '아이 보호' 커뮤니티 생성을 요청한다. 이 요청을 받은 사회는 아이 보호 커뮤니티를 생성한다.



Scene#3	Community 생성 및 Member Casting

Description

아이 보호 커뮤니티가 생성되면 커뮤니티는 아이 보호를 위해 필요한 멤버들을 커뮤니티로 캐스팅한다. 커뮤니티는 캐스팅된 멤버들에게 각자가 수행할 프로토콜을 전송하며, 프로토콜을 전송 받은 멤버들은 그에 따라 행동하며, 커뮤니티의 목표를 달성해 간다.

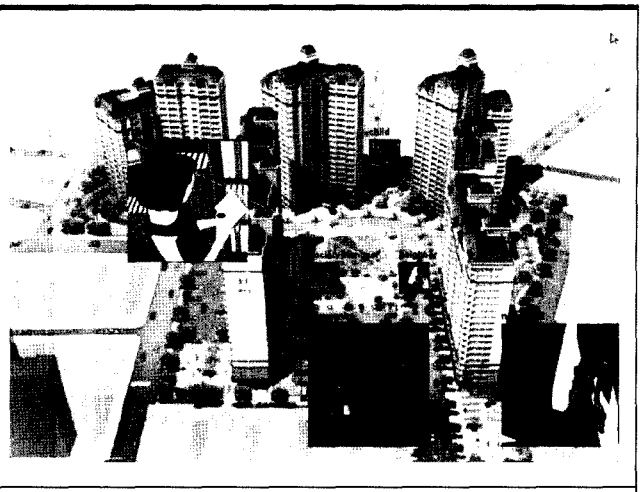
CIM-PI Script

```
Community CHILDCARE {
  Role CHILD:1 {
    Attribute: SAFETY_LEVEL={SAFE|
        LEVEL-1 WARNING| LEVEL-2 DANGER};
    Cast:SAFETY_LEVEL={ LEVEL-1 WARNING|
        LEVEL-2 DANGER};  }
  Role  FAMILY:1~50 {
    Attribute : RELATIONSHIP;
    Cast:RELATIONSHIP={CHILD.MOTHER|
      CHILD.FATHER|CHILD.SISTER|CHILD.BROTHER|
      CHILD.GRANDMOTHER|CHILD.GRANDFATHER};
  Role  NEIGHTBOR:1~20 {
    Attribute : RELATIONSHIP;
    Cast : RELATIONSHIP=CHILD.NEIGHTBOR;  }
  Role  WATCHER:0~2 {
    Attribute : MONITORING SERVICE=
      {AVAILABLE|NOT AVAILABLE}; LOCATION;
    Cast : MONITORING SERVICE=AVAILABLE;
      LOCATION=CHILD.LOCATION; }
  CHILD:Human,Smartbelt ; FAMILY:Human ; NEIGHBOR:Human ;
  WATCHER:Camcoder,Camera,Streetlamp;
  ..... }
```

Scene#4	이웃2의 아이 보호 요청 거절

Description

프로토콜 수행 과정에서 아이의 어머니는 이웃2에게 아이를 집으로 데려다 줄 수 있냐고 요청하지만, 이웃2는 사정상 요청을 거절한다. 이 과정은 커뮤니티 모델에 기술된 내용에 기반한다.



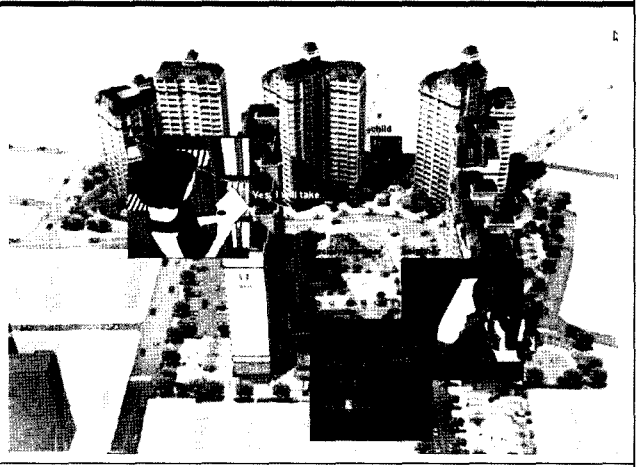
CIM-PI Script

```
Communication of NEIGHBOR{
IF(RECEIVE(MsgType="proposal",FromWho=FAMILY,
      InformedData="CHILDCARE", ChildLocation=child.location))
   Choose_responce(Choice1="reject", Choice2="accept", Choice);
   ALT((Choice="reject",1); (Choice="accept",2);)
     SEND(MsgType="reject-propose", ToWhom=FAMILY,
        InformedData="CHILDCARE", ChildLocation=child.location);
     SEQ
       SEND(MsgType="accept-propose", ToWhom=FAMILY,
            InformedData="CHILDCARE", ChildLocation=child.location);
          Take_to(Who=CHILD, Where=HOME_STUFF.ADDRESS);
          SEND(MsgType="inform", ToWhom=FAMILY,
                InformedData="CHILD_AT_HOME");
        END SEQ
      END ALT
   END IF
}
```

Scene#5	이웃1의 아이 보호 요청 수락

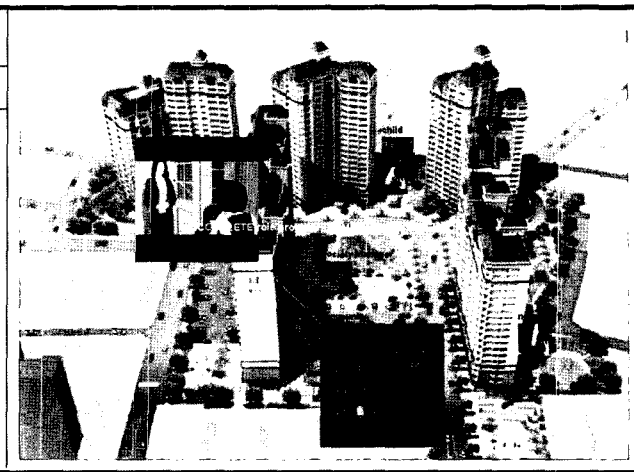
Description

아이의 어머니는 다시 이웃1에게 아이를 데려와 줄 것을 요청하고, 이 요청에 이웃1은 당연하다 생각하며 이를 수락한다.



CIM-PI Script

```
Communication of NEIGHBOR{
IF(RECEIVE(MsgType="proposal",FromWho=FAMILY,
      InformedData="CHILDCARE", ChildLocation=child.location))
   Choose_responce(Choice1="reject", Choice2="accept", Choice);
   ALT((Choice="reject",1); (Choice="accept",2);)
     SEND(MsgType="reject-propose", ToWhom=FAMILY,
        InformedData="CHILDCARE", ChildLocation=child.location);
     SEQ
       SEND(MsgType="accept-propose", ToWhom=FAMILY,
            InformedData="CHILDCARE", ChildLocation=child.location);
          Take_to(Who=CHILD, Where=HOME_STUFF.ADDRESS);
          SEND(MsgType="inform", ToWhom=FAMILY,
                InformedData="CHILD_AT_HOME");
        END SEQ
      END ALT
   END IF
}
```

Scene#6	커뮤니티 목표 달성
Description	
어머니의 요청을 받은 이웃은 아이의 위치로 이동해 아이를 데리고 아이의 집으로 아이를 데려간다. 아이가 어머니를 만나자 커뮤니티는 목표를 달성하게 되고, 커뮤니티는 소멸된다.	

CIM-PI Script

```
Community CHILDCARE {
Communication of FAMILY {
 IF(RECEIVE(MsgType="inform",FromWho=CHILD,
      ChildLocation=child.location, PictureOfChild=child.picture) )
   Choose_responce(Choice1="IGNORE",
         Choice2 = "CHILDCARE", Choice);
   ALT {(Choice="IGNORE",1); (Choice="CHILDCARE",2);}
     SEQ
      SEND(MsgType="inform",
        ToWhom="CHILD", InformedData="IGNORE");
          EXIT;
     END SEQ
     SEQ
       Choose_the_nearest_person(Location=CHILD.location,
           NearestPerson);
       SEND(MsgType="propose", ToWhom=NearestPerson,
        InformedData="CHILDCARE",ChildLocation=child.location);
       END SEQ
     END ALT
   END IF
   IF(RECEIVE(MsgType="reject-proposal", FromWho=CHILD,
      ChildLocation=child.location, PictureOfChild=child.picture) )
     SEQ
       Choose_the_nearest_person(Location=CHILD.location,
           NearestPerson);
       SEND(MsgType="propose", ToWhom=NearestPerson,
        InformedData="CHILDCARE",ChildLocation=child.location);
       END SEQ
     END IF
       ....
}}
```

그림 11 CDTK를 이용하여 개발한 아이보호 커뮤니티 컴퓨팅 시스템의 시뮬레이션

커뮤니티는 아이와 아이의 부모, 그리고 친한 동네 이웃들로 구성되므로 이들을 캐스팅하여 하나의 아이보호 커뮤니티를 생성한다. 일단, 커뮤니티가 생성되면 각 역할로 캐스팅 된 멤버에게 '아이보호' 목적을 달성하기 위한 프로토콜이 전송된다. 전송된 프로토콜을 끝까지 수행함으로써목적이 달성되는 것이다. 아이보호 목적을 달성하기 위한 프로토콜은 다음과 같이 작성하였다. 먼저 스마트벨트는 아이가 위험 범위에 있다는사실을 가족들 중 가장 가까운 사람에게 알리고, 아이를 데려올것인지를 묻는다. 이때 집에 있는 아이의 엄마가 아이를 데려오고자 하면, 커뮤니티 관리자는 자신에 속한 가족과 친한 이웃들 중에서현 시점에서 아이와 가장 가까이에 있는 사람을 선택하여 아이를 집으로 데려올 수 있는지를 묻는다. 요청을 받은 이웃이 이를 수락하여 아이를 안전하게 집으로데려오면 아이를 인계 받은 엄마는 목적이 달성되었다는 메시지를 커뮤니티 관리자에게 보낸다(이러한 과정은 엄마가 가지고 있는 개인 디바이스에서GUI로 이루어진다). 목적이 달성되었음을 전달받은 커뮤니티 관리자는 모든 멤버들과 사회 관리자에게 커뮤니티 해체를 알리는 메시지를 보낸다. 메시지를 받은 사회 관리자가 해당 커뮤니티 관리자를 삭제하면 아이보호 커뮤니티는 소멸된다.

6. 평가와 토론

본 논문에서 제안한 모델은 유비쿼터스 시스템에서 제공하는 서비스의 대부분이 협업에 의해서 제공되는 경우에 의미가 있다. 대부분의 서비스가 개별 요소에 의해서 제공된다면 유비쿼터스 시스템에서 협업을 위한

조직이 필요하지 않으며 이를 위한 추상화 모델 역시 필요하지 않다. 그러나 단순한 서비스는 개별 요소가제 공할 수 있지만 좀 더 복잡하고 지능적인 서비스를 제공하기 위해서는 요소들 간의 협업이 많은 부분에서 필요할 것으로 생각된다. 그러나 개별 서비스로도 충분한 작은 응용 시스템에서는 본 논문에서 제안하는 커뮤니티 컴퓨팅모델은 효과적이지 않을 것이다.

또한 현재 제안된 모델은 커뮤니티 컴퓨팅 모델의 초기 버전으로서 커뮤니티의 구성과 문제 해결 방식이 미리 정해져 있다. 따라서, 복잡하게 커뮤니티의 구성이 바뀌어야 하거나 커뮤니티가 문제를 해결하는 방식이 실시간으로 결정되어야 하는 문제는 해결할 수 없다. 그러나, 현재의 수준으로도 해결할 수 있는 유비쿼터스 시스템들은 상당히 존재한다고 여겨지며 이러한 시스템들에 대해서는 추가적으로 여러 시나리오를 만들어 검증하고자 한다. 현재 코엑스 몰과 같은 대형 쇼핑몰에서 한 개인의 요구 사항을 동적으로 파악하여 커뮤니티를 형성하여 제공하는 시나리오와 군사 작전에서 단순한 적의 공격에 대해서는 즉시에 커뮤니티를 만들어 적을 제압하는 군 시나리오를 고려하고 있으나 아직 구체화되지 않았으며 추후의 연구를 통해 지속적 발표할 예정이다.

마지막으로 CDTK에 대하여 생각하여 보자면, 이러한 개발 도구를 사용함으로써 각 모델에서 중요시 하는 부분만을 고려하여 개발을 진행하다가 개발과정의 마지막에서는 그러한 분리된 고려들이 모두 통합되어 코드로 표현되어 나오는 방식이 최근 소프트웨어 개발에서 중요시 하는 Separation of Concern의 개념을 따르고 있다고 할 수 있다. 이러한 장점 외에도 CDTK를 이용하여 개발자의 프로그래밍 작업을 어느 정도 줄일 수 있는지를 표 2에서 나타내어 보았다. 아래의 표에서 보면, 결과 가)에서는 CIM-PI의 커뮤니티 정보를 이용하

표 2 '아이찾기' 시스템 구현에 있어서 CDTK에 의한 코드 자동 생성 비율

Java 파일 이름		CDTK로부터 자동 생성된 코드 라인 수 (①	개발자가 추가 작성한 코드 라인 수 (자동생성코드의 수정 및 멤버 에이전트의 고유 기능 작성)	완성된 코드 라인 수 (②	CDTK에 의한 코드 자동 생성 비율 (①②
Society	GHODAMCITY.java	74	0	74	100
Community	Home.java	102	1	103	99.0
	CHILDCARE.java	104	5	109	95.4
Member	HomeApplication.java	78	0	78	100
	Human.java	173	35	208	83.2
	InanimateObject.java	85	8	93	91.4
	SocietyMember.java	104	9	113	92.0
	Streetlamp.java	103	5	108	95.4
	AnimateObject.java	87	2	89	97.8
	ElectronicAppliance.java	67	0	67	100
Protocol	CommunicationofNEIGHBOR.java	101	3	104	97.1
	CommunicationofInitiator.java	73	0	73	100
	CommunicationofParticipant.java	76	10	86	88.4
	CommunicationofWATCHER.java	75	18	93	80.6
	CommunicationofCHILD.java	78	13	101	77.2
	CommunicationofFAMILY.java	100	26	126	79.4
가)	합계 (16개 java 파일)	1,480	135	1,625	92.31
기본 protocol	Registration.java	80	0	80	100
	WaitingRegistration.java	48	0	48	100
	CommunityCreation.java	95	0	95	100
	MemberCasting.java	210	0	210	100
	ResponseToCasting.java	84	0	84	100
	WaitingCommunityCreation.java	81	0	81	100
	SendProtocol.java	72	0	72	100
	ReceiveProtocol.java	44	0	44	100
Utility	RoleBinding.java	105	0	105	100
	SelectiveValue.java	62	0	62	100
	Value.java	153	0	153	100
	RangeValue.java	42	0	42	100
	Resource.java	35	0	35	100
나)	합계 (29개 java 파일)	2,591	135	2,736	94.7

여 CDTK가 생성한 자바 파일에 대한 코드 자동생성 비율을 보여준다. 하나의 사회, 커뮤니티 템플릿, 멤버 템플릿, 그리고 커뮤니티 목적을 달성하는 프로토콜들이 각각 하나의 자바 파일로서 구현되며 CDTK는 CIM-PI의 내용으로 자바 파일들에서 협업에 관련된 부분들을 어느 정도 자동 생성해 준다. 이렇게 자동 생성된 부분에 개발자가 추가적으로 작성하여 완성된 자바 파일을 생성한다. 아이보호 커뮤니티 컴퓨팅 시스템의 경우 CDTK를 이용함으로써 개발자는 전체 코드를 작성하는 것이 아니라 자동 생성된 코드에 일부분만을 추가 작업하는 것으로 응용 시스템을 개발할 수 있다. 결과 나)에서는 어떤 응용 시스템을 개발하더라도 필수적으로 필요한 코드들을 포함한 비율을 보여준다. 커뮤니티 컴퓨팅을 위한 기본 프로토콜과 유틸리티 클래스들이 이에 속한다. 이러한 코드는 한번 개발되어 CDTK에 포함되면 모든 커뮤니티 컴퓨팅 응용 시스템에서 사용할 수 있으므로 이 역시 CDTK에 의하여 개발자가 자동으로 제공받을 수 있는 코드라고 할 수 있다. 이러한 것들을 포함하여 자동생성 비율을 계산하면 개발자가 얻는 편리함은 더욱 크다. 본 논문에서 보여주고 있는 '아이 보호' 시나리오의 경우 각 멤버의 기능을 개발하는 부분, 즉 멤버의 자체 behavior에 대한 코드를 제외하고 멤버들이 커뮤니티를 만들어 협업하는 부분에 대한 코드에 있어서는 약 5%에 해당하는 코드만을 작업하여 아이보호 시스템의 커뮤니티 협업 기능을 개발하였음을 보여주고 있다. 이때, 각 멤버 종류의 수행 기능은 개발자가 구현한다. CDTK가 각 멤버 종류에서 협업에 사용되는 멤버의 고유 기능에 대한 함수 선언을 주면, 이에 대한 구현은 개발자가 맡는다. 따라서 각 멤버의 고유 기능이 복잡하고 많아 프로그래밍 양이 많다면, CDTK가 자동 생성하는 코드의 양이 상대적으로 많은 도움이 되지 못할 수도 있다. 하지만, 전체 시스템을 CDTK를 이용하지 않고 개발하는 것보다는 이용하는 편이 훨씬 쉽고 편리하게 시스템을 개발할 수 있을 것이다. 본 논문에서 예를 들고 있는 '아이 찾기' 시스템은 각 멤버의 고유 기능이 매우 단순하고 이에 대한 코드 작성 량도 매우 적다. 따라서, CDTK로 자동 생성한 코드의 량이 전체 코드 량에 있어 매우 높아져서 아래의 표와 같이 매우 드라마틱한 코드 자동생성 비율이 나타나게 된 것이다. 실제 시스템에 있어서는 이와 같지는 않을 것이나 CDTK를 이용하여 얻을 수 있는 편리함은 반드시 존재한다고 볼 수 있다.

7. 결론 및 향후 연구

본 논문의 기여 사항은 다음과 같다.

- 협업 조직 기반의 유비쿼터스 시스템을 위한 추상화 모델로서 커뮤니티 컴퓨팅 모델을 제안하였다. 제안된 모델을 통해 기존의 에이전트 모델로는 완벽하게 표현되지 못했던 목적지향적인 협업 조직의 구조와 그들 간의 동적인 상호작용, 그리고 협업조직의 동적인 생성과 소멸을 커뮤니티 개념을 이용하여 모델로서 기술할 수 있다.
- 위와 같은 시스템을 다중 에이전트 환경에서 개발하기 위한 개발 과정을 제안하였다. 본 논문에서는 추상화 모델과 실제 개발 시스템간의 격차를줄이기 위해 MDA 방식을 적용한 커뮤니티 컴퓨팅 개발 과정을 제안하였으며, 이를 위하여 서로 다른 추상화 수준을 가지는 모델들과 기술 언어들을 정의하였다.
- 제안된 개발 방식을 좀 더 편리하게 진행하도록 하기 위해 커뮤니티 컴퓨팅 시스템개발 도구(CDTK)를 개발하였다. CDTK를 이용하여 아이보호 시스템을 구현하여 그 실현성을 검증하여 보았다.

그러나, 커뮤니티 컴퓨팅에 대하여 아직 완벽히 해결하지 못한 문제들이 많이 남아있으며, 이는 앞으로도 지속적으로 연구를 진행해야 할 것이다.

- 커뮤니티 모델의 기술 범위와 효용성 검증 - 제안된 커뮤니티 모델이 유비쿼터스 시스템에서 발생할 수 있는 문제 패턴들 중에서 어떠한 것을 해결할 수 있으며, 그 효율성은 어떠한지를 좀 더 구체적으로 실험하여 검증해야 하겠다.
- 커뮤니티 컴퓨팅 시스템에서의 Policy - 제안된 모델이 실제적으로 적용될 수 있으려면 존재 가능한 충돌(Conflict)를 해결할 수 있어야 한다. 이를 커뮤니티 모델에서 Policy로 기술하여 시스템에 적용하고자 한다.
- 동적인 커뮤니티 구조 형성 - 본 논문에서는 커뮤니티를 구성하는 역할들이 미리 정해져 있다고 가정하였다. 따라서 커뮤니티를 구성하는 멤버 에이전트의 종류와 해결할 수 있는 문제가 미리 정해져 있는 것이다. 그러나, 문제가 생기면 시스템에서 해결할 수 있는 문제인지를 판단하여 스스로 커뮤니티를 구성할 수 있으면 좀 더 다양한 문제를 해결할 수 있을 것으로 기대한다.
- 멤버들 간의 동적 협업 - 현재 커뮤니티의 목적(Goal)을 달성하기 위한 협업 방법은 미리 프로토콜에 의해 정의되어 있다. 즉, 프로토콜로 정의되지 않은 방법으로는 문제를 해결할 수가 없는 것이다. 설사 사회(Society)에 좀 더 효율이 좋은 멤버 타입이 새로 등록되었다고 해도 커뮤니티 프로토콜을 수정하지 않는 한 이를 이용할 수 없다. 따라서 앞으로는 커뮤니티에 필요한 능력을 가지고 동적으로 멤버를 찾아서 그들 간의 협업 패턴을 동적으로 만들 수 있으면 변화되는 상황에 맞춘 효율적인 방식으로 목적을 달성할 수 있

을 것이다.

- 다양한 플랫폼에서의 검증 - 현재 본 논문에서는 JADE 에이전트 플랫폼에서만 검증하여 보았으나, 여러 에이전트 플랫폼에서 또는 더 나아가 에이전트가 아닌 다른 플랫폼 환경에서도 가능한 모델과 개발 과정을 연구해야 하겠다.

참 고 문 헌

[1]. Nicholas R. J. On agent-based software engineering, Elsevier, Artificial Intelligence 117, 277-296 (2000).

[2] Weiser M. Ubiquitous Computing. Nikkei Electronics, December 6, 137-143, (1993).

[3] M. Wooldridge, Nicholas R. J. The Gaia Methodology for Agent-oriented Analysis and Design, Autonomous Agents and Multi-Agent Systems, 3, 285-312, (2000).

[4] R. Jennings, et. al. Developing Multiagent Systems: The Gaia Methodology, ACM Transactions on Software Engineering and Methodology, Vol.12, No.3, July, 317-370 (2003).

[5] J. Ferber and O. Gutknecht, "A meta-model for the analysis and design of organization in multi-agent systems", In Proceedings of 3rd International Conference on Multi-agent Systems (ICMAS '98), 1998.

[6] G. Cabri, L. Leonardi, F. Zambonelli, "A Framework for Flexible Role-based Interactions in Multi-agent System," In Proceedings of the 2003 Conference on Cooperative Information Systems (CoopIS), Italy, 2003.

[7] Youna Jung, Jungtae Lee, Minkoo Kim, "Multi-agent based Community Computing System Development with the Model Driven Architecture," Fifth International Joint conference on Autonomous Agents and Multiagent Systems, pp.1329-1331, May 12th 2006.

[8] Youna Jung, Jungtae Lee, Minkoo Kim, "Community based Ubiquitous System Development in Multi-agent Environment," 4th International Workshop on Ubiquitous Mobile Information and Collaboration Systems, pp.984-998,June 6th 2006.

[9] Youna Jung, Jungtae Lee, Minkoo Kim, "A Development Approach for Ubiquitous System using Community Metaphor," International Symposium on Ubiquitous Computing Systems, pp.129-138, Oct 10th 2006.

[10] Jennings, N. R., et. al. Transforming Standalone Expert Systems into a Community of Cooperating Agents, Int. Journal of Engineering Applications of Artificial Intelligence 6 (4), 317-331 (2003).

[11] Wooldridge M. An Introduction to Multiagent Systems, John Wiley & Sons, Reading, (2002).

[12] Mohan Kumar, et. al. PICO: A Middleware Framework for Pervasive Computing, Pervasive Computing, 1536-1268, 72-79 (2003).

[13] Object Management Group. Model Driven Architecture Guide, (2003).

[14] INMOS Ltd, OCCAM: Programming Manual. Prentice-Hall, Englewood Cliffs, NJ, (1984).

[15] FIPA Standard, SC00037J, FIPA Communicative Act Library Specification, (2002).

[16] Huber M. j. JAM Agent in a Nutshell, version 0.65+0.76i, November 1, (2001).

[17] F. Bellifemine, A. Poggi, and G. Rimassa. JADE - A FIFA-compliant Agent Framework. In Proc. of Practical Application of Intelligent Agents and MultiAgents (PAAM'99), London, UK, April, pp. 97-10 (1999).

정 유 나

2000년 2월 아주대학교 정보및컴퓨터공학부 공학사. 2002년 2월 아주대학교 정보통신전문대학원 공학 석사. 2005년 2월 아주대학교 정보통신전문대학원 공학 박사 수료. 2005년~현재 유비쿼터스 프로젝트(커뮤니티 컴퓨팅 모델). 관심분야는 멀티 에이전트 시스템, 커뮤니티 컴퓨팅

이 정 태

1979년 2월 서울대학교 농과대학 원예과 농학사. 1981년 2월 서울대학교 자연과학대학 계산학 이학석사. 1981년~1983년 서울대학교 자연과학대학 계산학 이학박사. 1983년~1988년 울산대학교 전산학과 전임강사. 1988년 2월 아주대학교 정보통신대학 전임강사. 1988년~현재 아주대학교 정보통신대학 정교수. 관심분야는 컴포넌트 베이스 시스템, 미들웨어, 객체지향 응용 프레임웍

김 민 구

정보과학회논문지 : 소프트웨어 및 응용 제 33 권 제 2 호 참조