

Community based Ubiquitous System Development in Multi-agent Environment

Youna Jung¹, Jungtae Lee², Minkoo Kim¹

¹ Artificial Intelligence Laboratory, Ajou University,
Suwon, Korea(South)
{serazade, minkoo}@ajou.ac.kr

² Programming Language Laboratory, Ajou University,
Suwon, Korea(South)
jungtae@ajou.ac.kr

Abstract. Ubiquitous system can be considered as the intelligent space, in which many kinds of component are connected by networks and also each component recognizes their connection and cooperates with others for achieving some goals. In many case of ubiquitous computing environments, users and devices interact and cooperate each other to attain some goals. To develop such a ubiquitous system, we should model a ubiquitous system in a cooperation view. To do this, in this paper, we model a ubiquitous system using community metaphor consistently. In the proposed high-level abstraction model, we can represent mission-oriented organizations and interrelationships between members or organizations. In addition, so far, a developer of a ubiquitous system should start from scratch and think about the action as an individual component and also the action as a member of cooperative organization at same time. Therefore, we also introduce the development process, which make it possible concerning the cooperative action and the individual action separately. To examine the proposed community computing model and the development process, we trying to implement a small system in JADE Platform.

1 Introduction

Agents are often used by software developers to more naturally understand, model, and develop a complex distributed system [1]. A ubiquitous system is a highly complex distributed system [2]. Therefore, it can be regarded as a multi-agent application which target support for daily-life activities. Multi-agent based ubiquitous systems are more than just an extension of multi-agent system to ubiquitous domain because ubiquitous computing systems have unique needs. In many ubiquitous environments, it is important that cooperation between computing components to offer ubiquitous services, so we need to represent cooperative organizations and relationships between them in a conceptual model of a ubiquitous system. In addition, to implement such a ubiquitous system, developers should concern about the cooperative actions and also own actions at same time. However, it is hard to develop a highly complex distributed system in such wise. Therefore, we try to formulate the development process in point

of separation of concern. In the high-level model, developers concern about only cooperative relationship in a ubiquitous system. Then, in the detailed model, individual actions of each computing component are concerned. Such separation of concern can be made development of a ubiquitous system easy and systematical.

In this paper, we propose a high-level abstraction model of a multi-agent based ubiquitous system in which cooperative relationships are important. In this model, we employ community metaphor to describe mission-oriented organization. Furthermore, we also propose the development process with model transformation of MDA. Each model helps in separation of concern.

The rest of the paper is organized as follows. We first introduce some related works and motivation in section 2. Then, in section 3, we give a brief description of the multi-agent based ubiquitous system and its development process. Section 4 presents each models used in development process, and transformations between the proposed models. We present a computational model for system implementation in section 5. Finally, section 6 is dedicated to the conclusion and future works.

2 Related Works and Motivation

Gaia introduced a methodology for agent-oriented analysis and design because existing approaches fail to adequately capture an agent's flexible, autonomous problem solving behavior, the richness of an agent's interactions, and the complexity of an agent system's organizational structures [3][4]. In Gaia, a multi-agent system is regarded as a collection of computational organizations consisting of various interacting roles. Gaia allows an analyst to go systematically from requirement statements to a design through a process of developing increasingly detailed models of the system to be constructed.

However, for developing a multi-agent based ubiquitous system, it is not adequate to use agent-oriented methodologies including Gaia as it stands. One reason is that they can not perfectly capture needs of ubiquitous computing. In many ubiquitous systems, it is much important that structure of organizations to attain goals and relationship between organizations. However, even in Gaia, an organization is only implicitly defined such structures within the role and interaction models [3]. In some ubiquitous computing systems, it is important to explicitly define organizations, because the objective of ubiquitous system is realized by achieving organization's goals. Therefore, for developing such a ubiquitous system, we need a high-level abstraction model to describe organization's structures and interactive behavior within and between them.

To abstract structures of mission-oriented organizations, we employ community metaphor. Community concept is introduced by several agent cooperation models [5][6]. In ubiquitous domain, PICO (Pervasive Information Community Organization) project [7] used community concept in 2003. PICO's objective is to meet the demands of automated, continual, unobtrusive services and proactive real-time collaborations among devices and software agents in dynamic, heterogeneous ubiquitous environments. To do this, PICO dynamically generates mission-oriented communities that perform tasks for users and devices. The major contribution of the PICO project

is the introduction of a concept, called community computing, and using it as a framework for collaboration among autonomous agents, called as deagent. The community computing concept provides the necessary platform to enable effective communication and collaboration among heterogeneous hardware and agents. A community of PICO is defined as an entity consisting of one or more agents working towards achieving a common goal. It provides a framework for collaboration and coordination among agents. The concept of community computing satisfies requirements of ubiquitous computing such as proactive real-time collaborations for automated, continuous services providing in heterogeneous environment. Therefore, we choose the community as a metaphor describing organization's structure in a multi-agent based ubiquitous system, and then describe the interrelationships between communities. In this paper, we define that the ubiquitous computing system modeled by community concept as the community computing system.

Another reason is the need of well defined and automated development process supporting separation of concern between cooperative concern and individual concern. Agent-oriented models care about two of them in a model at same time. After, developers should implement a system using the design in which organizational view and individual view are mixed. This situation requires solid development process supporting separation of concern. To do this, we employ MDA (Model Driven Architecture) [8]. MDA is an approach to system development, which increases the power of models in that work. So far, system development only uses models as a guide for implementing. However, it is hard that implementing a highly complex distributed system using only guide. Accordingly, MDA describes the whole development process from design to implementation. In order to do this, MDA proposes to start the process by building high-level abstraction models obtained by requirement analysis, and then refine them until obtain models that directly represents the final system. Using MDA approach provides benefit to the development of community computing systems. It make the ubiquitous systems developers focus on describing the system using high-level abstraction primitives, community, in the cooperation view. Then, the high-level abstraction model is transformed to the more detailed model until a frame of source code is obtained. In the detailed model, an individual view of each computing component is concerned. Though the model transformation, a portion describing cooperation of the source code is generated at least.

3 Development Process of Community Computing System

In this paper, we propose a high-level abstraction model and development process for multi-agent based community computing systems. In the proposed model, using community concept, we represent proactive, mission-oriented organization and interrelationship between them, explicitly. Additionally, to implement a system, we propose a development process based on MDA approach. The frame of source code can be generated from a design through the model transformation process of MDA. In MDA, several models reflecting different abstraction level are needed. For applying MDA approach to development of a multi-agent based community computing system, we also define models representing different abstraction level and modeling languages

for each model. The development process with models and runtime environment of a community computing system is shown in Fig. 1.

First of all, CCM (Community Computing Model), the most high-level abstraction model, describes a system with community metaphor. It models how communities make up a system and fulfill requirements of a system. To describe the CCM, we defined a modeling language, CML (Community computing Modeling Language). A CIM-PI (Platform Independent Community Computing Implementation Model) considers implementation of the system without concern about specific platforms. It shows the system in detail with computing components and interoperation between them. The frame of CIM-PI can be derived from a CCM, and the rest should be filled by developers. As a modeling language of CIM-PI, we define CIL-PI. To describe system implementation depending on a specific platform, we also define a CIM-PS (Platform Specific Community computing Implementation Model). The skeleton of CIM-PS can be also derived from a CIM-PI, but the rest should be filled manually. A CIM-PS combines the specifications in the CIM-PI with the details that specify how that system uses a particular type of agent platform. As a modeling language of CIM-PS, CIL-PS is used. Using the model transformation process from a CCM to a CIM-PS, the frame of the source code and a portion related cooperation of the code are generated. This process allows community computing system development to be ease and systematic, because the developer can advance his thought from cooperation view over a system to individual view of each component. Furthermore, we can guarantee consistency in the entire development process, because we can manage the process with a coherent metaphor, community.

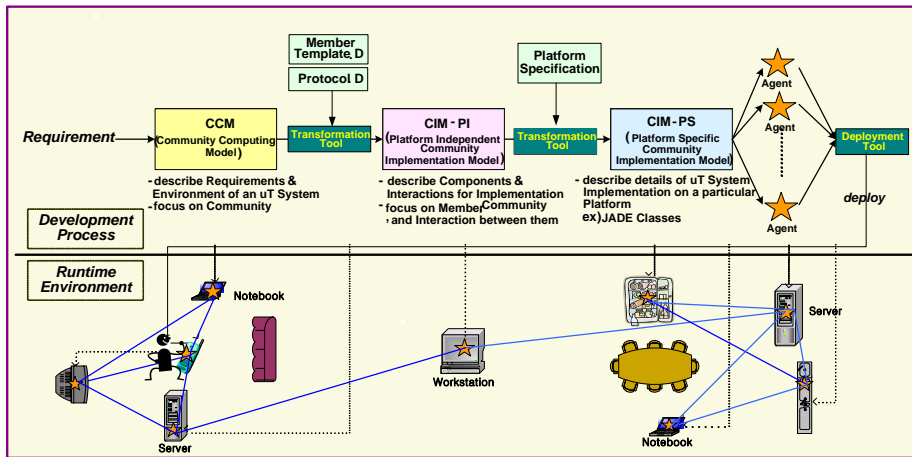


Fig. 1. The development process and runtime environment of a community computing system

4 Community Computing Models

4.1 Community Computing Model (CCM)

A CCM is the most high-level abstraction model of a community computing system. The objective of a CCM is to describe requirements and environment of a ubiquitous system using the community concept. At runtime, a community computing system makes the intelligent space combined virtual environment and physical environment. We called this space as *U-space* (Ubiquitous space). *U-space* is a logical and physical area where ability of a ubiquitous system can influence. *U-space* can be defined as a collection of individual agents and communities. At this time, each agent represents an individual computing component constituting a ubiquitous environment. In addition, a set of communities represent ability of the ubiquitous system for meeting requirements. Accordingly, to describe the requirement and environment of a community computing system, a CCM describes communities and agents can be existed in the system. In a CCM, the life cycle of agents and communities is described. Using the life cycle description, a system can manage the creation and termination of communities and agents members (see Section 5, at runtime, a society manager control the creation and termination of a community). The ability of a system is represented as goals of communities. The way to achieve community's goals is presented as protocols of the community, but is not described in detail. In short, CCM is a sketch of a multi-agent based community computing system, which shows how the system fulfills its requirements in a *U-space*.

Before describing the structure of the CCM in detail, it is worth introducing the basic concepts constructing the core of community computing.

- *Member*: it is the building blocks of a community computing system, and be implemented as an agent. It can represent a computing device, physical hardware, human, and software. Its ability is represented through its attributes and actions. At ordinary time, it performs its own action. Then, triggering a goal, it forms a community and interact others to achieve the goal. In this paper, we assume that all types of member, which can be existed in a system, are predefined.
- *Community*: It is a metaphor to describe a proactive organization and made up with agent members collaborating with others to achieve particular missions. A community has goals and protocols. A community can manage the life cycle of its own members. In detail, a community can control entry and secession of each member by offering joining conditions into a community. As a goal arises dynamically, a community is dynamically generated to achieve the goal. At this time, actually, a society manager representing a society creates a community manager representing a community. Therefore, the community manager constructs a community by casting individual agents. After then, the community is disorganized when the goal is attained. In this paper, we assume that community types and the structure of each community are predefined. It means that all community types and a community's goals and protocols should be predefined in a system.

- *Goal*: It is a particular condition that a community should satisfy. In short, it is the objective of a community.
- *Protocol*: It is a description of interaction between community's members to achieve a goal. That is, a protocol shows how each role performs its job (cooperative action and individual action) to obtain its objective. It defined as a sequence of communicative messages and member agent's actions. While a community performs a protocol, another goal can be issued to achieve the goal of the community.
- *Society*: It is a metaphor to represent *U-space*, a system. It is constructed by agent members and communities. At the first time, only members exist in society. In detail, when a ubiquitous system starts to operate, a society manager is created. Therefore, a society manager broadcast and then each agent recognizes and a society and registers. After the formation of a society, a community can be generated when an agent recognizes a goal. Then, if the community attains its objective, a community is terminated but the society is alive until the community computing system is exhausted.

A CCM has two part, community description and society description. In the community description, we defined structure, goals, and protocols of each community. To describe a CCM, we create a modeling language, CML (Community computing Modeling Language). To prove our idea, we apply the proposed development process to the CHILDCARE scenario. In the CHILDCARE scenario, when a child goes out of home, a smartbelt of the child requests community generation to the society manager representing a society. Then, a community manager of CHILDCARE community is created by a society manager, and then casts agent members for each role. For safety of the child, the CHILDCARE community informs child's family of the situation of the child, and then the child's mother searches a nearest person who can help the child to go home. Finally, when the child arrives home, a goal of the CHILDCARE community is achieved and then the community is disorganized.

<pre> 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.RESIDENT, Participant=HOME_STUFF.RESIDENT) } } 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; </pre>	<pre> 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; } } } </pre>
--	--

Fig. 2. An example of CCM using CML

As the first step of the development process, a CCM of the childcare scenario is shown in Fig. 2. As you see, two community types can be existed in the system, HOME and CHILDCARE. A community type describes roles, goals, and ontology. A role is defined by attributes and conditions to cast member agent performing the role. An attribute is represented by an attribute name and a set of possible values. If the attribute value not be defined, data type can take the place of particular values of the attribute. Among attributes, cardinality is a mandatory attribute, so it represented in next of role name. A goal is defined by a goal name and protocol for achieving the goal. A protocol presents initiator, participant, and sub-protocol can be issued on the way to perform the protocol. Optionally, an ontology name can be described if the community uses. In the society description, society name (in Fig.2, GHODAMCITY) and the condition to join into the society are defined. In our example, the value of ADDRESS.CITY attribute should be “GHODAM” to belong to the GHODAMCITY society.

4.2 Platform Independent Community Implementation Model (CIM-PI)

The objective of CIM-PI is to describe the architecture of a system concerning implementation. This model exhibits a specified degree of platform independence so as to be suitable for use with a number of different platforms of similar type. A CIM-PI extends community and society descriptions of a CCM in detail. In the community description of CIM-PI, it is represented that mapping relationships between roles and member types and cooperative interaction s to achieve a goal. Each role of the community is matched with certain member types, and the member type should be able to play the role. In addition, protocols of the community are detailed. A protocol description consists of the initiator’s communicative action and the participant’s that. The language for protocol description is based on Occam [9]. Occam enables the programmer to express a program in terms of concurrent processes which communicate by sending messages through communication channels. We apply constructs of Occam to our protocol description for concurrency handling. Constructs of the communicative action is as follows: *SEQ*, *PAR*, *ALT*, *IF* and *EXIT*. The communicative action can be member’s actions or primitive actions such as *SEND* and *RECIEVE*. We import message types of FIPA (Foundation for Intelligent Physical Agents) [10]. In the Society description of CIM-PI, all member types are defined. In the member description, the hierarchy of member types, and member’s attributes and actions are defined. Using the keyword, *extends*, we represent the hierarchy of member types. For examples, *Streetlamp extends Electronic Appliance* means that Streetlamp type is a child type of Electronic Appliance type, so Streetlamp type is inherited attributes of Electronic Appliance. We defined CIL-PI to model a CIM-PI. An example of CIM-PI is shown in Fig. 3.

<pre>Platform Independent Community Implementation Description { Community Home { Role HOME_STUFF:1-100000 {</pre>	<pre>Protocol take_a_child_home { communication of child { </pre>
--	--

<pre> 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; RESIDENT: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:Camcorder,Camera,Streetlamp; </pre>	<pre> Society GHODAMCITY { Member Society Member { Attribute : LOCATION=GHODAMCITY; CITY_ADDRESS=GHODAM; SAFETY_LEVEL={SAFE LEVEL-1 WARNING LEVEL-2 DANGER}; Actions : 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; HEIGHT=INTEGER; USAGE={HOME INDUSTRY RESEARCH}; } Member Home Appliance extends Electronic Appliance { Attribute : USAGE=HOME; ASSIGNED_ROOM={LIVINGROOM KITCHEN BED ROOM 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); } } </pre>
---	--

Fig. 3. An example of CIM-PI using CIL-PI

4.3 Platform Specific Community Implementation Model (CIM-PS)

A CIM-PS combines the specifications in the CIM-PI with the details that specify how that system uses a particular type of platform. At this moment, platform means that agent platforms such as JAM [11] or JADE [12]. A CIM-PS is represented by agent codes for a particular platform. Frame of a CIM-PS can be derived from a CIM-PI. In detail, the cooperation portion and the frame as a programming guide is generated. In this paper, we implement a multi-agent based community computing system on JADE platform so our CIM-PS describes java files for each member type, community type, and GHODAM society in JADE environment. A CIM-PS of our scenario is shown in Fig. 4.

<pre> package GHODAMCITY.agent; import jade.core.Agent; import jade.core.behaviours.*; </pre>	<pre> } catch (FIPAException fe) { fe.printStackTrace(); } addBehaviour(new WaitingRegistration(this)); </pre>
---	--

<pre>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);</pre>	<pre> 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"); } }</pre>
---	--

a) Generated GHODAMCITY.java from CIM-PI(GHODAMCITY society)

<pre>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 = (SimpleBehaviour) Class.forName(protocolName).newInstance(); addBehaviour(behaviour); } catch (ClassNotFoundException e) { System.out.println("Behaviour Not Found : " + protocolName); } catch (Exception e) {</pre>	<pre> 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("RESIDENT", "Human", "CommunicationofParticipant"); } protected void takeDown() { try { DFService.deregister(this); } catch (FIPAException fe) { fe.printStackTrace(); } } public void updateMemberList(AID member) { members.add(member);</pre>
--	---

```
e.printStackTrace(); } }
b) Generated HOME.java from CIM-PI (HOME community)
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) Generated HomeAppliance.java from CIM-PI (HomeAppliance member type)

Fig. 4. An example of CIM-PS

4.4 Model Transformation

Model transformation is the process of converting one model to another model of the same system. To develop a system, MDA performs model transformation from a high-level abstraction model to a low-level abstraction model representing the final implementation. In our proposal, the model transformation process starts the process by building a CCM and refine them until obtain the source code. The first step of the model transformation is conversion of a CCM into a CIM-PI. Skeleton of a CIM-PI is derived from a CCM, and the rest is filled using additional information. To complete such interpretation process, information concerning implementation is needed such as member type description, hierarchy of member type, mapping between roles and member types, and protocol description. In the second step, we convert a CIM-PI into

a CIM-PS using specification of particular agent platform. A CIM-PS is a collection of agent classes representing each member type, each community type, and a society. The configuration of agent classes differs with agent platforms. That is, a CIM-PS should describe the java files, if the system is basis on JADE platform. After we gain a CIM-PS, we can generate the source code, for examples, java files. In each model of our development process, a developer can focus on the model's view. In model description, only cooperative behavior of the system is concerned. After the model transformation, a developer programs the individual action portion of source code additionally. Finally, the finished code is deployed into a system. That is the end of development process for a multi-agent based community computing system. After deployment, at runtime, deployed agents build up a U-space, and then organize communities as occasion demands until the system is destroyed. The conversion relationships between models and additional information used for model transformation are shown in Fig. 5.

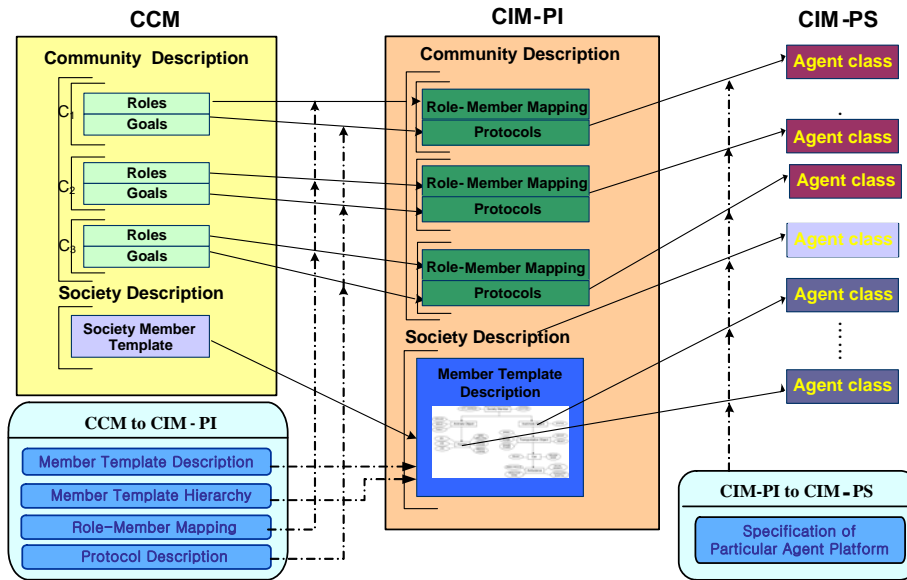


Fig. 5. Model transformation from CCM to CIM-PS with additional information

5 Implementation

To examine the proposed community computing model and the development process, we develop a small community computing system to serve CHILDCARE in JADE platform. JADE (Java Agent Development Framework) is a software development framework aimed at developing multi-agent systems and applications conforming to FIPA standards for intelligent agents [12]. To run a multi-agent based community system, we propose a computational model and it is shown in Fig. 6.

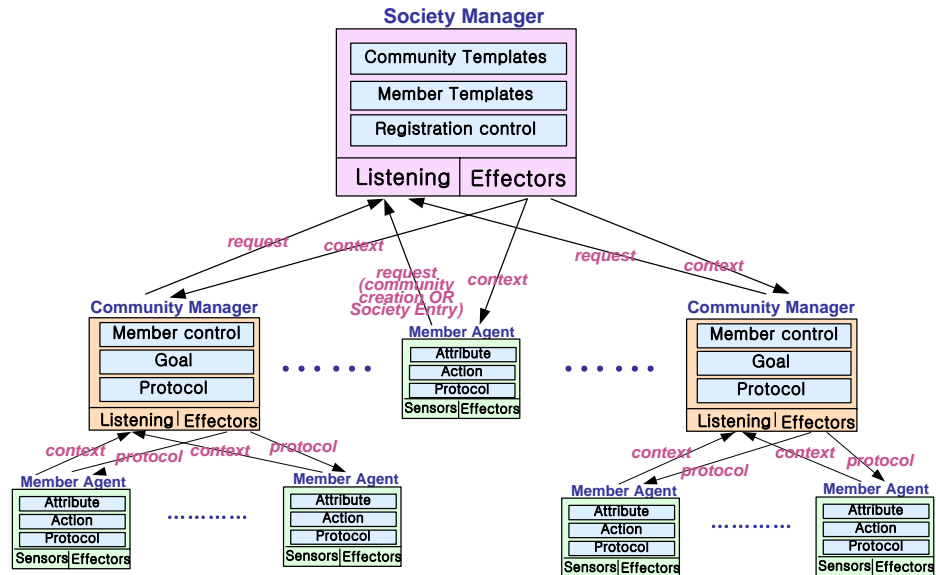
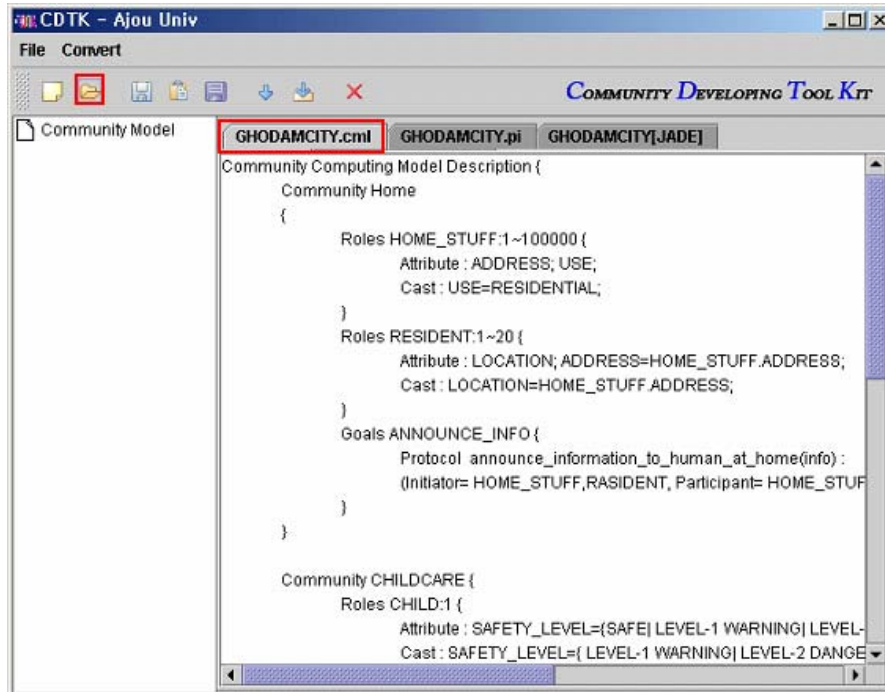


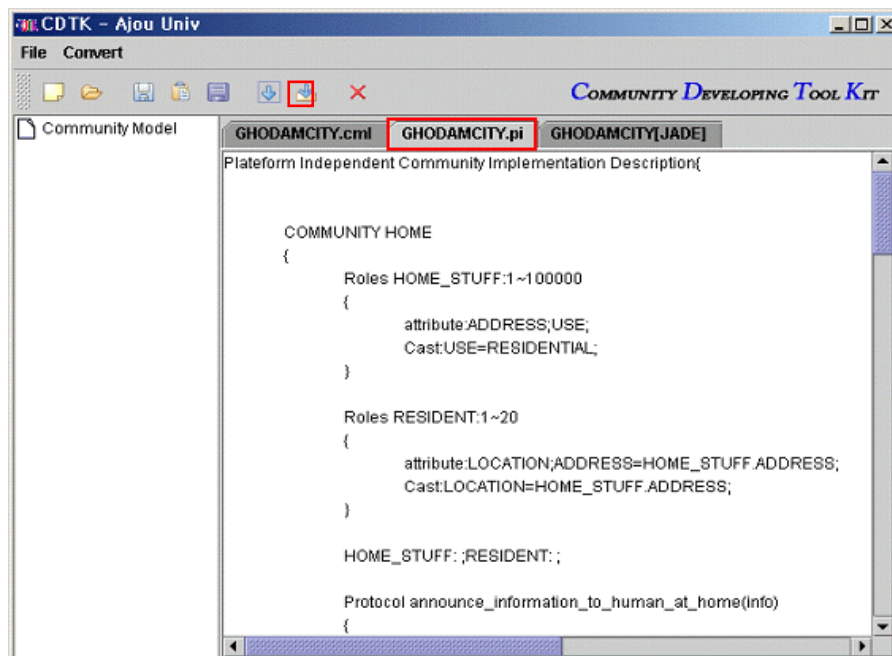
Fig. 6. Computational model of a multi-agent based community computing system

Only agent members exist in a society when a system starts to operate. At this time, to represent a society, a society manager agent is generated. It contains information about member types and community types, and society registration protocols of agent and community manager creation and termination protocols. Each agent should be registered with a society. Then, the society manager generates a community manager to achieve a goal when a member agent set particular a goal. A community manager representing certain community has condition to cast member agent and protocols for achieving goals. During a community manager performs a protocol, another community can be generated to achieve the goal. Such relationships between communities are described in the protocol description. After the community attains its goal, a society manager removes the community instance and announces disorganization to agent members. Each member agent has its attributes, actions, and default protocols for society registration and requesting a community creation.

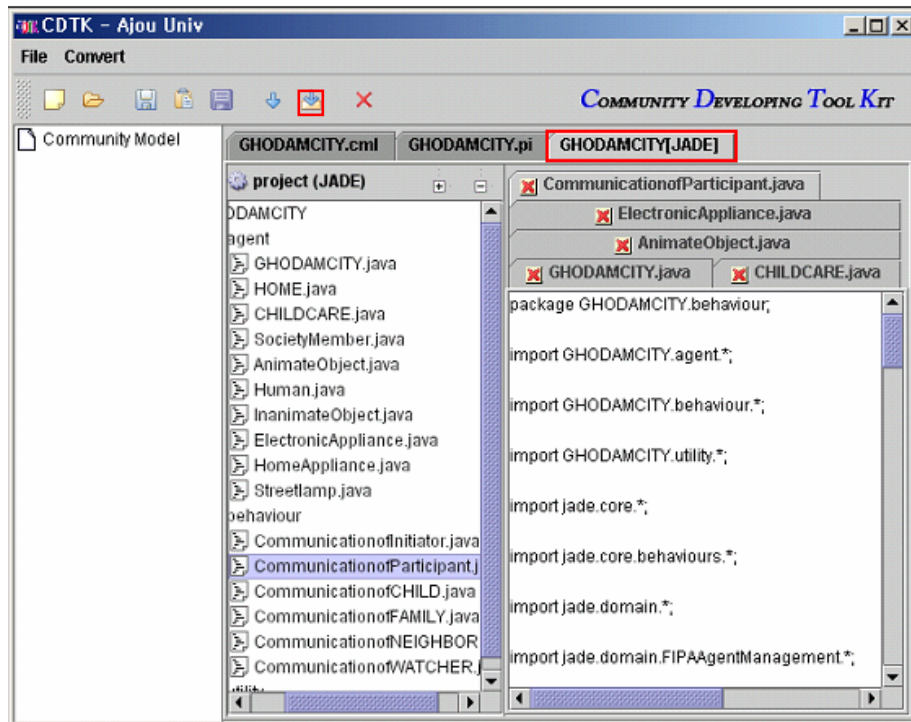
To develop a community computing system with MDA, we implement the Community computing system Development ToolKit (CDTK). In the CDTK, a developer can describe a system as a CCM file, and then CDTK transforms the CCM file to its CIM-PI file. Of course, the developer fills the particular portion of CIM-PI. Similarly, the CIM-PI file is transformed java files. At this time, an amount of java code is automatically derived from the CIM-PI, and then the developer just programs own behavior of each member. Our CDTK reduce an amount of programming, and it is valuable in development of complex and huge ubiquitous systems. Implemented toolkit is shown in Fig. 7.



a) The GHODAMCITY CCM



b) Transformed the GHODAM CIM-PI



c) Transformed the GHODAM CIM-PI

Fig. 7. Community computing system development using CDTK

6 Conclusion

In this article, we proposed a high-level conceptual model using community metaphor to support dynamic management of goal-oriented organization and cooperation in multi-agent based ubiquitous systems. Then, to fill a gap between a design and implementation, we propose the entire development process through model transformation from a high-level conceptual model to implementation. To do this, we define models representing different abstraction level and modeling languages for each model. For operation of a community computing system, we also propose a computational model.

However, there are several issues remaining for future work.

- Dynamic structuring of community – In this paper, we assume that structure of a community and community types are predefined. However, this assumption set a limit to ability of a system. That is, the system can handle only predefined goals. In future, we will make the system to support dynamic formation of community structure

- Dynamic cooperation of member agents – In this model, the cooperation is static in a system, because the community's goal is fixed. However, it makes a community to achieve its goals in always same way. Although another way can offer the more efficient methods to accomplish a goal, a community should use a predefined mean until such another way is defined as a new protocol.
- Test on variety platforms – we implemented a system on only JADE platform, so we are going to verify our idea on other platforms.

7 Acknowledgement

This research is supported by the ubiquitous Autonomic Computing and Network Project, the Ministry of Information and Communication (MIC) 21st Century Frontier R&D Program in Korea

References

1. Nicholas R. J. On agent-based software engineering, Elsevier, Artificial Intelligence 117 (2000), 277-296
2. Weiser M. Ubiquitous Computing. Nikkei Electronics, December 6, 1993, 137-143.
3. M. Wooldridge, Nicholas R. J. The Gaia Methodology for Agent-oriented Analysis and Design, Autonomous Agents and Multi-Agent Systems, 3, 2000, 285-312
4. R. Jennings, et. al. Developing Multiagent Systems: The Gaia Methodology, ACM Transactions on Software Engineering and Methodology, Vol.12, No.3, July 2003, 317-370
5. 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), 2003, 317-331.
6. Wooldridge M. An Introduction to Multiagent Systems, John Wiley & Sons, Reading, 2002
7. Mohan Kumar, et. al. PICO: A Middleware Framework for Pervasive Computing, Pervasive Computing, 1536-1268, 2003, 72-79
8. Object Management Group. Model Driven Architecture Guide, 2003
9. INMOS Ltd, OCCAM: Programming Manual. Prentice-Hall, Englewood Cliffs, NJ, 1984
10. FIPA Standard, SC00037J, FIPA Communicative Act Library Specification, 2002
11. Huber M. j. JAM Agent in a Nutshell, version 0.65+0.76i, November 1, 2001
12. F. Bellifemine, A. Poggi, and G. Rimassa. JADE – A FIPA-compliant Agent Framework. In Proc. of Practical Application of Intelligent Agents and Multi-Agents (PAAM' 99), London, UK, April 1999, p.97-10