

Multi-agent based Community Computing System Development with the Model Driven Architecture

Yuna Jung
Artificial Intelligence Laboratory
Ajou University, Suwon
Korea(South)
+82-31-219-2442
serazade@ajou.ac.kr

Jungtae Lee
Programming Language Laboratory
Ajou University, Suwon
Korea(South)
+82-31-219-2436
jungtae@ajou.ac.kr

Minkoo Kim
Artificial Intelligence Laboratory
Ajou University, Suwon
Korea(South)
+82-31-219-2437
minkoo@ajou.ac.kr

ABSTRACT

Ubiquitous system can be considered as a system, in which many kinds of component are connected by networks and the components cooperate with each other for achieving some goals. To develop a ubiquitous system, several researches have used multi-agent models. Even though current multi-agent models provide many benefits with ubiquitous system developments, they are not enough to develop ubiquitous systems. In this paper, we address the ubiquitous system which solves its problems with cooperative organization consisting of several heterogeneous computing components. We call such a ubiquitous system as a community computing system. In the community computing system, some goals are given to be achieved by the cooperative organization. In this paper, we propose a high-level abstraction model of multi-agent based community computing system. In addition, we propose the development process of the community computing system using MDA(Model Driven Architecture) approach.

Categories and Subject Descriptors

I.2.11 [Distributed Artificial Intelligence]: Distributed Artificial Intelligence – *Multi-agent systems, Coherence and Coordination.*

General Terms

Design, Languages

Keywords

Multi-agent systems, Ubiquitous, Community, MDA

1. INTRODUCTION

A ubiquitous system 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. General multi-agent models can not completely support specific features of ubiquitous systems. In this

paper, we focused on the agent-based ubiquitous system solving the problems with mission-oriented organizations and called such ubiquitous system as the community computing system. In the community computing system, a community consists of several agents and provides ubiquitous services can not provided by individual computing component. The community computing system has features such as mission-oriented organization, role, goal, cooperation between roles or organizations, and so on. Such specific features of the community computing system can not be completely modeled using existing multi-agent models because existing models deal with general and low-level concepts. Therefore, in this paper, we propose a new high-level abstraction model using the community metaphor. In addition, we concern the development of the community computing system as well as high-level abstraction. In deed, it is hard to develop a highly complex distributed system using only a design. Therefore, we try to formulate the development process from high-level abstraction to implementation of a multi-agent based community computing system with model transformation of MDA.

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 propose the community computing system and its development process. Section 4 is dedicated to the conclusion and future works.

2. RELATED WORK

Gaia introduced a methodology for agent-oriented analysis and design [1]. In Gaia, a multi-agent system is regarded as a collection of computational organizations consisting of various interacting roles. Such a view of Gaia is analogous to our view of a cooperation based ubiquitous system. However, it is not adequate to apply agent-oriented methodologies including Gaia to develop a multi-agent based cooperative ubiquitous system. One reason is that they can not capture characteristics of the cooperation based ubiquitous computing. In the agent models, even in Gaia, an organization is only implicitly defined such structures within the role and interaction models. In the cooperation based ubiquitous computing systems, it is important to explicitly define organizations, because the objective of a system is realized by achieving organization's goals. Therefore, we need a high-level abstraction model to describe organization's structures and interaction behavior within and between them. To do this, we employ community metaphor. Community concept is introduced by several agent cooperation models [2]. In ubiquitous domain, PICO (Pervasive Information Community Organization) project [3] used the community concept. The community of PICO provides a framework for collaboration and coordination among

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

AAMAS'06, May 8–12, 2006, Hakodate, Hokkaido, Japan.

Copyright 2006 ACM 1-59593-303-4/06/0005...\$5.00.

agents. The concept of community computing satisfies requirements of cooperation based 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 cooperative ubiquitous system, and then describe the interrelationships between communities. In this paper, we define that the ubiquitous system modeled by community concept as the community computing system.

Another reason is the lack of well defined and automated development process from design to implementation. Agent-oriented models only care about analysis and design of a system. This situation requires solid development process from analysis of system requirement to implementation. To define the development process of multi-agent based community computing systems, we employ MDA (Model Driven Architecture) [4]. MDA is an approach to system development, which increases the power of models in that work. 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. Then, the high-level abstraction model is transformed to the more detailed model until a frame of source code is obtained [5].

3. COMMUNITY COMPUTING SYSTEM AND ITS DEVELOPMENT PROCESS

In this paper, we propose the high-level abstraction model and development process for multi-agent based community computing systems through the model transformation process of MDA. For applying MDA approach, we need several models representing different abstraction level and modeling languages for each model.

Before presenting each model 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. At ordinary time, it performs its own action. Then, triggering a goal, it forms a community, and interact each other to achieve the goal. In this paper, we assume that all templates 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. Offering a condition of joining into a community, a community can control entry and secession of each member. As a goal dynamically arises, a community is dynamically generated to achieve the goal. Then, the community is disorganized when the goal is attained. In this paper, we assume that community templates and the structure of each community are predefined. That is, a community, which is not predefined, is can not created in a system. As well as templates of community, goals and protocols of a community are also fixed.
- *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. It defined as a sequence of messages and member's actions. While a community performs a protocol, another goal can be issued.
- *Society*: It is a metaphor to represent U-space, the environment of a system. It is constructed by agent members and communities. At the first time, only members exist in society. After the start of the society, a community is generated if an agent recognizes a goal. When it attains its objective, a community is terminated, but the society is alive until the community computing system is exhausted.

First of all, as a most high-level abstraction model, CCM (Community Computing Model) describes the multi-agent based community computing system with community concept. Its objective is modeling the requirement and environment of a system. At runtime, a community computing system makes the 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 agents and communities in a CCM. To describe the CCM, we defined a modeling language, CML (Community computing Modeling Language). A CIM-PI considering implementation of Community computing system without concern about specific platforms. It shows the system in detail with computing elements and interoperation between them. A CIM-PI extends community descriptions and society descriptions of CCM in detail. In community description of CIM-PI, it is represented that mapping relationships between roles and member templates and interactions of protocols. Each role of the community is matched with certain member templates can playing the role. In addition, protocols of the community are detailed. The frame of CIM-PI (Platform Independent Community Computing Implementation Model) can be derived from CCM, and the rest should be filled by developer. To express a CIM-PI, we define CIL-PI as a modeling language. To describe implementation of a system 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 is represented as program codes in specific agent platform such as JAM or JADE.

Using the model transformation process from a CCM to a CIM-PS, we can obtain the frame of source code. This process allows community computing system development to be ease and systematic, because the gap between design and implementation is filled. Furthermore, we can guarantee consistency in the entire development process, because we can manage the process with coherent philosophies.

```
Space GHODAM_U-Space {
Community TRAFFICACCIDENT {
  Roles PATIENT : 1 ~ 500 {
    Attribute : BODYCONDITION={NORMAL|ABNORMAL};
                CALLAMBULANCE={YES|NO};
    STATUS={MOVE|STAY|CLASHED};
    Cast : STATUS= CLASHED; }
  Roles CAR : 1~ 500 {
    Attribute : STATUS={NORMAL|CLASHED};
                DRIVING MODE={PARKING|REVERSE|DRIVE|NEUTRAL|STOP};
    Cast : DRIVINGMODE=STOP; STATUS= CLASHED; }
```

```

Goals TRAFFICACCIDENTHANDLING {
    Protocol request_for_ambulance_service :
        (Initiator=PATIENT or CAR, Participant= PATIENT or CAR) {
            Subprotocol : AMBULANCE-ASSOCIATION.AMBULANCE SERVICE,
                patient profile; } }
    Ontology : TRAFFICONTOLOGY; }
Community AMBULANCE-ASSOCIATION { ..... }
Community EMERGENCYAMBULANCE { .....}
Society GHODAMCITY{
    Member GHODAMCITIZEN {
        Attribute : ADDRESS-CITY= STRING; Cast : ADDRESS-CITY= GOTHAM; } } }

```

Figure 1. An example of CCM using CML

```

Platform Independent Community Implementation Description {
Space GHODAM_U-Space {
    Community TRAFFICACCIDENT {
        PATIENT:Human; CAR:Car ;
        Protocol request_for_ambulance_service {
            Communication of Initiator {
                SEQ
                make_profile_of_Patient;
                Send_Msg(request, Participant:1, profile_of_Patient);
                CASE(failure, Participant:1, All ambulance is busy)
                Send_Msg(request, Participant:1, profile_of_Patient);
                CASE(inform, Participant:1, Ambulance_ID)
                Waiting_for_Ambulance(Ambulance_ID); }
            Communication of Participant {
                CASE(request,Initiator:1, profile_of_Patient))
                SEQ
                Check_Idle_Instance(Ambulance,Driver,Paramedic);
                Select_Instance(Ambulance,Driver,Paramedic);
                ALT Send_Msg(failure, Initiator:1, All ambulance is busy);
                Send_Msg(inform, Initiator:1,Ambulance_ID); } } }
        .....
    }
Society: GHODAMCITY {
    Member Society Member {
        Attribute : LOCATION=GHODAMCITY; CITY_ADDRESS=GHODAM;}
    Member Animate Object extends Society Member {
        Attribute : SPECIES=STRING; GENUS=STRING; FAMILY=STRING;}
    Member Human extends Animate Object {.....}
    Member Inanimate Object extends Society Member {.....}
    Member Transportation Object extends Inanimate Object {.....}
    Member Car extends Transportation Object { .....}
    Member Ambulance extends Car {.....} } }

```

Figure 2. An example of CIM-PI using CIL-PI

```

GOALS: ACHIEVE trafficaccident_handling "patient";
FACTS:
Plan: { NAME: "request for ambulance service"
    GOAL: ACHIEVE trafficaccident_handling $PATIENT;
    ASSERT patient $PATIENT;
    BODY: }
Plan: { NAME: "communication of initiator"
    GOAL: PERFORM initiator;
    BODY: RETRIEVE patient $PATIENT;
    ASSIGN $SUCCESS_FLAG "Failure";
    WHILE : TEST (== $SUCCESS_FLAG "Failure") {
        EXECUTE connectToAgentAsClient $PORT $HOST $IN $OUT;
        EXECUTE sendMessage $OUT $PATIENT;
        EXECUTE receiveMessage $IN $ANSWER;
        WHEN : TEST (== $ANSWER "Inform") {
            EXECUTE receiveMessage $IN $AMBULANCE_ID;
            PERFORM waiting_for_ambulance $AMBULANCE_ID;
            ASSIGN $SUCCESS_FLAG "True"; }; } }

```

```

Plan: { NAME: "communication of participant"
    GOAL: PERFORM participant;
    BODY: EXECUTE connectToAgentAsServer $PORT $HOST $IN $OUT;
        EXECUTE receiveMessage $IN $PATIENT;
        ASSIGN $IDLE_FLAG "Failure";
        PERFORM check_idle_instance;
        WHEN : TEST (== $IDLE_FLAG "Inform") {
            PERFORM select_instance;
            EXECUTE snedMessage $OUT "Inform";
            EXECUTE snedMessage $OUT $AMBLUANCE_ID; };
        WHEN : TEST (== $IDLE_FLAG "Failure") {
            EXECUTE snedMessage $OUT "Failure"; }; }
//...

```

JAM file of TRAFFICACCIDENT community template, "TRAFFICACCIDENT.jam "

Figure 3. An example of CIM-PS based on JAM[6] platform

4. 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 an implementation model. However, there are several issues remaining for future work.

- Capability verification of community computing model
- Policy of community computing model
- Dynamic structuring of community
- Dynamic cooperation of member agents
- Test on variety platforms

5. 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

6. REFERENCES

- [1] Jacques Ferber, *Multi-agent Systems*, Addison-Wesley, Reading, MA, 1999.
- [2] 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.
- [3] Mohan Kumar, Behrooz A. Shirazi, Sajal K. Das, Byung Y. Sung, et. al. PICO: A Middleware Framework for Pervasive Computing, *Pervasive Computing*, 1536-1268, 2003, 72-79
- [4] Object Management Group. Model Driven Architecture Guide, 2003
- [5] Javier Munoz. Pervasive Systems Development with Model Driven Architecture, In *Proceedings of the 7th International Conference on the Unified Modeling Language (UML2004)* (Lisbon, Portugal, October 10, 2004).
- [6] Huber M. j. JAM Agent in a Nutshell, version 0.65+0.76i, November 1, 2000