Journal of Universal Computer Science, vol. 16, no. 15 (2010), 2139-2174
submitted: 31/1/10, accepted: 28/7/10, appeared: 1/8/10 © *J.UCS*

Situation-Aware Community Computing Model for Developing Dynamic Ubiquitous Computing Systems

Youna Jung
(LERSAIS, University of Pittsburgh, Pittsburgh, USA
yjung@pitt.edu)

Minsoo Kim
(LERSAIS, University of Pittsburgh, Pittsburgh, USA
minkim@pitt.edu)

Abstract: For many complex and dynamic ubiquitous services, context-aware cooperation can be a solution. However, the way is not yet clear to make individual objects cooperate with each other as situations change. In addition, in the present environment in which many smart agents are already deployed, we are able to quickly develop ubiquitous services by utilizing existing agents. In the case of urgent but unavailable services, such fast development is required but there is no existing work to provide a path. To meet such requirements, in this paper, we thus introduce community computing as a new paradigm in which ubiquitous services are provided through context-aware cooperation among existing agents. To design such systems intuitively, we propose an abstraction model, called the situation-aware community computing model which includes the community situation model and the situation-aware cooperation model. In addition, for fast and convenient system development, we propose a development process based on the MDA (Model-Driven Architecture) approach [OMG, 03]. Following the development steps of MDA, we propose three models each having different abstraction levels and the model transformation process from the high-level model, CCM, to the source code. To make such transformation semi-automatic, we develop a toolkit, called CDTK. By using CDTK, we are able to implement a community computing system conveniently and systematically. To verify the proposed work, we implemented two small systems based on motivated scenarios; CHILDCARE and COEX-Mall. Through the simulated results of those systems, we examined the possibility of community computing as a new development paradigm.

Keywords: Community Computing, Ubiquitous Computing System, Cooperation, Context-Awareness, Multi-agent System Development, Model Driven Architecture
Categories: H.5.3, I.2.11, K.6.3

1 Introduction

Since ubiquitous computing was articulated by Mark Weiser in 1991 [Weiser, 91], many researchers have attempted to realize the potential of a variety of ubiquitous services. In the present study, we surveyed existing research and found the unique characteristics of ubiquitous computing as follows [Weiser, 91][Kindberg, 02].

- Composition of highly heterogeneous computing objects
- Dynamic request for resources and services in a ubiquitous environment
- Dynamic interaction among heterogeneous computing objects

- Frequent environmental changes due to mobility of users and computing objects

First of all, a ubiquitous computing system is composed of highly heterogeneous computing objects. As computing objects have mobility and their status changes frequently, the environment of a ubiquitous system is continuously changing. In such a dynamic environment, predictable or unpredictable ubiquitous services are dynamically requested. Among all characteristics, we are especially concentrating on the complexity and dynamics of ubiquitous services.

Since many ubiquitous services require various tasks, most of them can be effectively provided through cooperation among heterogeneous computing objects rather than through the ability of an individual one. As the required ubiquitous services become larger and more complex, cooperation among ubiquitous objects becomes increasingly more important. Accordingly, it is necessary to raise concerns about the cooperation-based service providing scheme including the configuration of cooperative organizations and cooperative behaviors among ubiquitous objects.

Another crucial issue of ubiquitous computing systems is context-awareness, which is able to provide services properly while the computing environment is dynamically changing. To support dynamic cooperation, therefore, cooperation systems should possess context-awareness. Context-aware computing [Schilit, 94] [Dey, 01] has emerged as a promising way to build intelligent and dynamic systems. Several beneficial features such as dynamicity, adaptability and interoperability make context-awareness become one of the essential requirements of recent systems. In spite of such advantages coming from context-awareness, some researchers tackled the problem of limitation of representation power; the context is weak for giving comprehensive understanding of a phenomenon. In order to make up for the weakness, this situation was introduced. In this paper, we therefore exploit the concept of community situation to support dynamic cooperation.

Our ultimate goal is to design and develop a ubiquitous computing system which dynamically provides diverse services, even highly complex or unpredictable services. In order to achieve our purpose, we introduce the situation-aware community which consists of existing objects. In this paper, community is a high-level abstract concept for organizing, managing, operating, repairing groups of computing objects in ubiquitous environments. By using the situation-aware community concept, we are able to satisfy the requirements of ubiquitous computing as follows.

- Adaption to environmental changes
  - Dynamic goal-driven community creation
  - Dynamic binding of the community roles and computing objects
- Dynamic cooperation
  - Dynamic injection of cooperation into objects
  - Dynamic decision of cooperative behavior
- Proper separation of concerns
  - Separation between group concerns and individual element concerns
- Scalability of the environment
  - Dynamic merging of group organizations

To develop the ubiquitous systems providing the situation-aware community services, in this paper, we propose community computing as a development paradigm and introduce its abstraction model, the situation-aware community computing model.

In addition, we propose a development process to implement community computing systems fast and easily. In this process, we provide three models which represent a system in different abstraction levels. For verification of the proposed work, we implement two community computing systems by using the proposed models and the development process.

This paper is organized into 7 chapters including the present chapter. In Chapter 2, we show our motivation and requirements using two example scenarios. Chapter 3 provides background related to our requirements; ubiquitous system development work including the middleware-based approach and multi-agent based approach, cooperation system development work, and context-aware system development work. By the comparison of our work with existing research from the viewpoint of requirements, we emphasize our contribution. In the Chapter 4, we introduce community computing as a new development paradigm for the ubiquitous systems providing dynamic cooperative services. We first introduce basic terminology then focus on community by specifying the levels of communities and the lifecycle of a community. Furthermore, for systematic development, we propose the development process based on the MDA approach from CCM to source codes. In Chapter 5, for supporting dynamic cooperation among agents, we propose the situation-aware community computing model which includes the community situation model and the situation-aware cooperation model. Chapter 6 shows the simulation results of two small systems implemented for motivational scenarios, CHILDCARE and COEX-Mall. Finally, in Chapter 7, we conclude this paper by stating our contributions and future work.

2 Motivation

As mentioned above, our goal is to develop ubiquitous computing systems that dynamically provide complex and large-scale services, even unpredictable services. Towards this end, we need to fulfill the following requirements; context-awareness, cooperation, utilization of existing smart objects, model possession to design and develop a system.

- Context-awareness - To offer the best services according to the changing situation, first of all, context-awareness is required. By guaranteeing the context-awareness, a system can provide the most proper service based on the user's condition through the most available and performable computing object at the requested time. In order to do so, the contexts of users and computing objects are also significant as well as ambient contexts such as time or temperature.
- Cooperation - Cooperation is an efficient solution to provide services requiring large and diverse tasks. By using cooperation, we can increase the reusability of services more than by developing large-scale services. Furthermore, the quality of services can be improved by hiring the best objects for each task.
- Utilization of existing smart objects - If there are various smart objects in an environment, it is a good idea to utilize them actively. Compared to developing other services from scratch, it makes the process of service

development easy and fast. Let's assume that we attempt to develop ubiquitous services for buildings or specific areas. In such a scenario, there would be many heterogeneous smart devices, such as cell phones, PDAs, notebooks, surveillance cameras, smart television, or smart sensors performing their own tasks. If we use their various capabilities through cooperation among them, we are able to save a lot of time and effort.

- Possession of a model to design and develop a system - In order to deal with unpredictable services, we need the process of intuitive design and fast development for services. If we have a model to help such a process, we can quickly provide services, even if they have not been developed.

In order to achieve our goal, all these requirements should be satisfied. To solidify our requirements, we show various scenarios as following section.

2.1 Motivation Example

To explain our motivation effectively, we describe two scenarios that show the necessity for context-aware cooperation among smart objects in a ubiquitous computing environment; CHILDCARE and COEX-Mall.

CHILDCARE. Let's assume that we develop a ubiquitous computing system for an apartment. This system consists of various smart objects such as personalized smart devices; smart watches, smart phones like the iphone, or PDAs, smart home devices; smart television, audio, or bed and apartment monitoring devices like surveillance cameras, ambient sensors; temperature sensor or fire alarm sensor, and so on. Each ubiquitous object provides its own services to residents. For example, people can watch television shows or movies, listen to music, take phone calls, or check messages from other residents or from the apartment office. Bill's family lives in this apartment. In the morning around 11am, Bill went to work and Amy, his wife, is washing dishes while listening to the radio from the smart audio in the kitchen. Tom, their five year old son, is playing with toys in the living room. After a while, Tom sees his friend so he goes outside to play with the friend. However, Amy does not realize he has left the house because of the sound of running water. To find his friend, Tom goes too far from home and he is not safe now because the area where he is standing is adjacent to a road. At that time, surveillance cameras keep watching him and Tom's watch knows the exact location of Tom. Furthermore, there is Susan who lives next door to Tom but she cannot help Tom because she does not know this situation and what they need to do.

- Cooperation – To bring him back to home, diverse tasks are required. For example, we need to get the location of Tom, transmit his image and location information to his mother, search the nearest person to Tom, and ask someone for help. It is easy to make objects providing such services cooperate with each other than to develop a complex service with all the capabilities such as location acquisition, face recognition on a video stream, and transmission of multimedia messages.
- Utilization of existing smart objects – Many existing objects already have the required capabilities. For instance, many personal devices such as GPS watches or smart phones can know the location. Surveillance cameras are able to recognize faces by video processing. In addition, smart phones can display information about Tom with his image. If we utilize those existing

objects, we can reduce the time and cost to develop the service to bring a child home.

- Possession of a model to design and develop a system – If such service is unavailable, it should be developed and deployed as fast as possible. To provide unpredictable services, we need a model to make the service development convenient.
- Context-awareness – To provide the best service, participant objects and their tasks are dynamically decided according to the environmental context or objects' context. For example, the closest surveillance camera to Tom is in the CHILDCARE community to watch Tom. For the same reason, Susan is selected as a guardian of Tom since she is in the location closest to him. In addition, it is better to let Amy know Tom is out by a text message not a voice alarm since she may be not able to hear any voice due to the sound of running water. As you see, the contextual information such as the location of Tom and the status of Amy is significant to provide services effectively.

COEX-Mall. COEX-Mall is a huge shopping mall with a multiplex cinema, an amusement park, restaurants, and stores. To attract customers and offer several convenient services, several robots exist in this mall. The robots have various capabilities such as movement, alarm, face recognition, voice recognition, and information searching. Based on those capabilities, they provide many services such as the advertising service, the information display, the patrol service, and the terrorist detection service. When a robot patrols its area, a woman asks it to find her missing son.

- Context-awareness – Only robots and patrolmen who are on duty can help to find a missing child. In addition, the context information of the child is useful to find him such as his height, weight, image, and preference.
- Cooperation – To search for a child in a wide area, it is good to try to find him in every area simultaneously. In order to do this, robots patrolling the area need to cooperate with each other.
- Utilization of existing smart objects – If we use a robot with the capability of searching a specific person, we can immediately provide a service for a missing child. It is definitely more cost-effective than if we develop another similar service from scratch.
- Possession of a model to design and develop a system – if we have an abstract model to specify a service, we can quickly and easily deploy the missing child service by modifying the existing terrorist detection service.

As you see in the two scenarios, we can dynamically provide large-scale services which require diverse tasks including unprepared urgent services by satisfying four requirements. In order to do so, we have surveyed a lot of previous work to meet such requirements. In the next section, we describe related work then compare it to our work in detail.

3 Related Work

As mentioned above, our goal is to develop a ubiquitous system that dynamically provides useful services, even unpredictable and highly complex services. Looking

forward, we need to find a way to fulfill four requirements in a ubiquitous system; context-awareness, cooperation, utilization of deployed objects, and model possession. First of all, we surveyed existing research on ubiquitous system development and found an earlier work which simultaneously satisfies our all requirements. However, we were not able to find anything that exactly met our objectives. As a second step, we tried to find research on context-aware systems and cooperation systems. However, there was nothing adequate for our needs since they usually have a bias toward one requirement, either context-awareness or cooperation. In this section, we introduce each element and declare our contribution by a comparison with existing work.

3.1 Ubiquitous System Development

For the past few decades, many researchers have paid attention to the development of ubiquitous systems. As a result, various approaches were proposed but we concentrated on the most popular approaches; the middleware approach and the multi-agent based approach. However, most existing work on these approaches does not fully consider cooperation among distributed agents or components connected to middleware. They only mention cooperation, and fail to provide a way to design and enforce cooperation. In this section, we introduce some outstanding work in each approach.

3.1.1 Middleware based Ubiquitous System Development

In the area of the ubiquitous system development, the objective of middleware approaches is to offer an infrastructure to manage resources, sense context information, and assist in the development and execution of ubiquitous applications. Although most existing middleware does not support ad-hoc communications, an important feature of ubiquitous computing environment [Ejigu, 08], this approach is still a good way to combine distributed components and provide integrated services. Hence, for middleware, we introduce Gaia in the Super Spaces project and PICO.

Super Spaces. In the Active Spaces Project [Roman, 00], an experimental middleware infrastructure, called Gaia, was introduced to coordinate ubiquitous software objects and heterogeneous networked devices contained in a physical space. The major contribution was to present active spaces as a programmable environment instead of a collection of individual and disconnected heterogeneous devices. In 2004, an extended version, Super Space, was proposed to manage and orchestrate groups of Active Spaces [Al-Muhtadi, 04]. However, they did not suggest an abstraction model to conceptualize ubiquitous objects constructing the space or cooperative relationships between objects in their space.

PICO. PICO (Pervasive Information Community Organization) [Kumar, 03][Sung, 02] is a middleware framework for dynamically creating mission-oriented communities of autonomous ubiquitous software objects offering ubiquitous services. In several agent cooperation models, organizational concepts have already been introduced [Jennings, 03][Wooldridge, 02], but PICO has applied such concepts to ubiquitous domains. In this project, a community was defined as a ubiquitous object consisting of one or more agents working towards a common goal. In addition, they introduce community computing as a framework for collaboration among agents.

Their fundamental concept satisfies requirements of ubiquitous computing, such as proactive real-time collaborations for automated and continuous services provided in a heterogeneous environment.

3.1.2 Multi-Agent based Ubiquitous System Development

The Multi-agent based approaches are frequently used to develop a ubiquitous system because of agents' features such as flexible and autonomous problem solving behavior and the richness of interactions. This approach focuses on the way to seek out necessary agents while meeting the requirements of a ubiquitous system. When the requirements are given first, it starts to find a way to design and implement necessary agents to offer the required services.

However, in the case of a ubiquitous system intending to provide services using existing objects, participant agents in a system are already defined. In that case, it is more important to consider how to meet the system requirements using existing agents rather than what agents are required. Additionally, most multi-agent based ubiquitous system development approaches do not deeply concentrate on cooperation. To achieve our goal, however, cooperation is the most important aspect. Therefore, we need to consider cooperation in more detail than others do. In this section, we briefly introduce previous work relating to multi-agent based ubiquitous systems.

Gaia. Gaia [Wooldridge, 00][Jennings, 03] introduced a methodology for analysis and design of a multi-agent system. 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 design through a process of developing increasingly detailed models of the system to be constructed.

AALADIN. AALADIN [Ferber, 98] is a meta-model of a multi-agent system based on organizational concepts. It allows for describing any kind of organization using only the core concepts of groups, agents, and roles. In the extended version [Ferber, 03], the model was improved into an AGR model (Agent/Group/Role model). In this model, the dynamic aspect is added by specifying the creation of a group, the entering and exiting mechanism of an agent within a group, and the role acquisition mechanism.

BRAIN. BRAIN [Cabri, 03] is a framework for supporting the different phases of the development of interaction in MASs by modeling the interactions between agents based on the concept of roles and describing such roles using an XML-based notation known as XRole. The authors implemented Rolesystem as an interaction infrastructure of BRAIN, but they did not concern cooperation.

3.2 Cooperation System Development

As you see, the existing work in the area of ubiquitous system development handles the cooperation issue in the naïve manner. However, cooperation has been a good way to solve a problem requiring diverse resource and capabilities and perform a highly resource-consuming and time-consuming task [Wooldridge, 99]. The ubiquitous service is one of domains requiring such tasks. That is the reason that cooperation is an essential aspect to achieve our goal.

To support cooperation among computing objects in a ubiquitous computing system, we surveyed the existing work for the development of cooperation systems, such as CSCW (Computer-Supported Cooperative Work) [Wilson, 91]. In fact, researchers have used cooperation, but there is a slight difference between existing work in the meaning and style of cooperation. In this section, we investigate previous work in terms of the cooperative group and cooperation style and we compared similar research within our computing community from the perspective of cooperation as shown in Table 1.

Team in Computer Supported Cooperative Work (CSCW). The major objective of CSCW is to develop a groupware that effectively performs a common task using information sharing among all users [Borghoff, 00]. Typically, a group in CSCW is a small project-oriented team and a team is defined as a set of predefined people. Team members are human users and their cooperative work is tightly coupled by sharing information about team membership as well as the skills or roles of the other members [Johansen, 98]. In the group protocol component, the ways to cooperate and communicate are described. Typical groupware of CSCW are the video conferencing system and the joint document editing system.

Organization in Multi-Agent System. To provide services requiring complex interactions such as ubiquitous services, multi-agent systems are frequently developed [Wooldridge, 00]. To cooperate with other agents, an organization is constructed and each role of the organization is dynamically assigned to a member agent for performing a cooperation protocol/procedure. Such cooperation procedure is able to be predefined or dynamically determined depending on the agent's intelligence but it is usually fixed. In most multi-agent systems, the cooperation procedures are first decided to provide the requested services and then agents that perform such procedures are designed. For example, in Gaia methodology [Zambonelli, 03], the cooperative procedure is predefined in the protocol description. Since the existing multi-agent systems do not have any cooperation model, it is not easy to design or modify the cooperation.

Community in PICO. PICO is a middleware framework for dynamically creating mission-oriented communities of ubiquitous objects using the community concept to represent the structure of cooperating organizations. However, it has no cooperation model and leaves room in its framework. For example, there are no explanations about how to create communities and members and manage them, dynamically assign software objects, called delegent, into the real objects, and make members cooperate with each other.

Community Computing in Digital Kyoto Project. This project introduced community computing in 1998 in order to support the process of organizing diverse and amorphous people who are willing to share knowledge and experiences [Ishida, 98]. The objective of their work was to make a city-scale support system to assist humans in their everyday lives. In the Digital Kyoto project [Ishida, 02][Besselaar, 02], a community is a digitalized representation of real human communities. All human members in a community share their preference and knowledge and reach consensus. This project aims to support such a process in the aspects of hardware and network because all decisions about cooperation are made by human users.

Community Computing in Microsoft. In 2005, Microsoft introduced its vision of community computing [Blau, 05]. It defines community computing as an emerging

technological environment where users sharing other's computing capacity and user id's and objects are spread all over these devices. Microsoft insisted that system engineering focus on managing massive connections among powerful devices in a community computing environment [Microsoft, 05]. Looking ahead, one of the Microsoft research groups is trying to systematically support its community computing by developing various tools capable of making people interact with each other when gathering and exchanging services. In its research, a community is group of devices sharing information and capacity, concentrating on the systematical support for communities to facilitate communications among humans. It seems that they are.

	Member		Cooperation			Decision Maker
	Type	Selection	Goal	Process	Style	
CSCW	Human	Static	Video conferencing, Document authoring	Static	Tightly coupled	Human
Multiagent	Agent	Dynamic	Unlimited	Static /Dynamic	Tightly/ Loosely coupled	Agent
PICO	Agent	Dynamic	Unlimited	Static /Dynamic	Tightly/ Loosely coupled	Agent
Digital Kyoto	Human	Dynamic	communication among humans	Static /Dynamic	Tightly coupled	Human
Microsoft	Device	Static /Dynamic	communication among humans	Static / Dynamic	Tightly coupled	Human
Our Community Computing	Agent	Dynamic	Unlimited	Static /Dynamic	Tightly/ Loosely coupled	Agent

Table 1: Comparison of related work with our work in the viewpoint of cooperation

3.3 Context-Aware System Development

Context-awareness is one of the most critical issues in the ubiquitous computing research area because most ubiquitous computing systems are required to have the advantages of dynamism, adaptability and interoperability. In this section, we introduce recent work related with the context aware system.

As the computing environment changes from distributed computing to mobile computing, and again from mobile computing to a ubiquitous/pervasive computing [Strang, 04], context-awareness plays a key role in designing and developing applications. In the literature, the term 'context' has been defined in several ways. Schilit referred context as where you are, who you are with, and what resources are nearby [Schilit, 94]. Information about location and identification of people or objects can be regarded as context information. These kinds of context information are very useful for modern systems, but it is difficult to describe a situation based on location and identification. Schmidt et al. pointed out the problem and argued there is more to

context than location [Schmidt, 99]. They categorized context information into several types and introduced a hierarchically organized context model. Dey et al. [Dey, 01] pointed out the same problem in which these definitions are too specific and consequently they proposed a more general definition: Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves. This definition makes it easier for an application developer to enumerate the context for a given application scenario. The system developer or designer may use any kind of information as context to express situations.

In order to take advantage of context-awareness, several techniques such as context modeling and context acquisition are required [Baldauf, 07]. In the literature, a number of context frameworks, including these technologies, were proposed. In particular, several works aim for context-awareness in ubiquitous computing systems. Among them, we introduce a few outstanding works as follows.

Context ToolKit. Dey et al [Dey, 01][Salber, 99] have proposed the Context-Toolkit, which provides a framework for the development and execution of sensor-based context-aware applications and provides a number of reusable components. The Context Tool-Kit supports rapid prototyping of certain types of context-aware applications, but it is too general to engineer context for cooperation systems.

CoCA. Ejigu et al [Ejigu, 07] have proposed a collaborative context-aware service platform, called CoCA, for pervasive environments. CoCA provides context modeling method with relational databases and ontology. In particular, CoCA proposes a fast context reasoning algorithm by pruning non-relevant context data (i.e. reducing data set for reasoning). However, the collaboration concept for context-awareness is too weak and hence it is hard to apply to cooperation systems.

AMUSE. AMUSE [Takahashi, 05] proposed by Takahashi et al. is an agent-based middleware for context-aware ubiquitous services. The main target of this work is to maintain consistency of resource context data and multiple context coordination in pervasive environments. To do this, AMUSE introduces the concept of 'agentification', which makes each resource an agent. Each agent possesses context management ability and cooperation ability to resolve context conflict.

UbiCoMo. In 2009, Bortenschlager et al. [Bortenschlager, 09] proposed a ubiquitous coordination model called UbiCoMo. It provides comprehensive concepts for context-sensitive coordination of agents in ubiquitous environments. To do this, UbiCoMo has a multi-layered architecture representing real world agents, ubiquitous entities, and services. UbiCoMo shows efficiency and usability of its coordination method by testing on MAS, but it is not concerned about cooperation, the main feature of MAS.

In addition, there are many context-related works. However, it seems that context-awareness for cooperation systems is still a challenging issue. In cooperation systems, it is important to know context which related with a cooperative group or inferred context from cooperating members rather than context of individuals. However, there is no existing work dealing with cooperation factors, such as participant members, the goal of cooperation, and the status of cooperation.

3.4 Our Contribution

Although we surveyed many areas, we were not able to find previous research to adequately satisfy our four requirements; context-awareness, cooperation, utilization of existing objects, and model possession. In Table 2, we present an evaluation of related research along with our work in terms of these four requirements.

			Context-awareness	Cooperation	Existing objects utilization	Model possession
Ubiquitous System	Middleware	Super Space	O	X	O	X
		PICO	X	X	O	Low
	Multi-agent based	Gaia	X	Medium	X	O
		AALADIN	X	Low	X	O
		BRAIN	X	Medium	X	O
Cooperation System		CSCW	X	Low	X	X
		Digital Kyoto	X	Low	X	X
		Microsoft	X	Low	X	X
		Multi-agent	O	Medium	X	O
Context-aware System		Context ToolKit	High	X	X	X
		CoCA	High	X	X	X
		AMUSE	High	X	X	X
		UbiCoMo	High	Low	X	X
Our Community Computing			O	High	O	O

Table 2: Evaluation of our work and other related work in terms of our requirements

Gaia middleware in Super Space project has its own context model based on predicates [Ranganathan, 03] to make applications context-aware, but it does not consider cooperation. PICO middleware provides a description of devices, agents that represent users or applications, and communities in there model. However, it is not complete because many things are not specified such as the creation of community and the cooperation process of the community. Authors have explained how to provide pervasive services by using the communities of agents but have not presented the way to design and enforce cooperation in a community. In addition, they have left development of the context-awareness service to the future.

The Gaia methodology has several models to develop a multi-agent system, but context-awareness is not considered in those models. In Gaia's models, cooperation among agents is represented as a protocol, which consists of the protocol name, the initiator and partner, input and output, and the description of the cooperation procedure. There is no cooperation model and a developer must implement such cooperation in the agent system according to the protocol description. In AALADIN, the organizational concepts, such as groups and roles, were introduced but there is no way to describe cooperation in a group. In the BRAIN framework, interactions between roles can be described by events and actions of each role but cooperation among roles is not present. Since these works aim to develop agents, they are far from using existing agents.

Existing cooperation systems, such as Digital Kyoto, MS's Community Computing, and groupware in CSCW, aim to systematically support cooperation among human users. They focus on how to help users communicate with each other and perform common tasks without conflicts, rather than how to achieve a complex goal by cooperation. Furthermore, they do not take context-awareness seriously. Since the objectives of this work are to develop a new system, they do not utilize existing objects.

In existing context-aware systems, cooperation is not considered so they do not handle context related to cooperation. To represent context, they have formal models but not to develop services or systems. UbiCoMo provides the coordination method for services but it is not for developing services. In addition, it is out of its scope to use deployed objects. In fact, there is no existing work in this area, which fulfills our requirements.

As existing work is not suitable for achieving our goals, we previously proposed a new development paradigm that we call community computing for ubiquitous computing systems that meet our all of our requirements [Jung, 06]. In this early version, a community computing system has an abstraction model for intuitive design and the development process based on the MDA (Model-Driven Architecture) approach for systematic implementation. In addition, existing agents in an environment can be specified in the abstraction model and then used to achieve the community goals. The context information about members and environment was also represented in the model. However, context-awareness is only used for assigning agents to roles and not for cooperation. In the early version, the cooperation procedure was fixed and described like a pseudo code in the model without any cooperation model. That means this model is not able to support dynamic cooperation.

To fully satisfy our requirements, we propose here an improved model, which has the situation-aware cooperation model. To use the community's situation, we first propose the community situation model. This model provides the way to represent and use the context of members and the community situation. The cooperation model, employing the community situation model, allows members to cooperate with each other according to the community situation. Members' tasks are dynamically decided depending on the change in the community's situation. By using these two models, we can finally guarantee the context-awareness and the dynamic cooperation in ubiquitous computing systems. This paper's contribution to community computing is as follows:

- Guaranteeing the context-awareness in a ubiquitous computing system
  - To represent the context information of agents, we use key-value pairs. According to the agent's context, many things are decided, such as the community's creation, proper member selection, and cooperation procedure.
- Supporting dynamic cooperation among agents to provide complex and large-scale services
  - By using the situation-aware cooperation model, we can provide dynamic services according to the community's situation
- Developing a ubiquitous system by using agents that have already been deployed

  - In our community computing, we propose a way to use existing smart agents to quickly provide services, even unpredictable services. All existing agents in an environment are represented as members in a community computing system.
- Using an abstraction model to intuitively design a system and a development process
  - To design a system, we propose an abstraction model, named the community computing model. Then, we develop a system according to the MDA-based development process from the high-level model to implementation.

4 Community Computing

In this paper, we propose community computing as a development paradigm for ubiquitous computing systems where ubiquitous services are provided by cooperation among existing agents. Community computing focuses on how to satisfy the requirements of a ubiquitous system by cooperation among deployed agents, while multi-agent based approaches focus on what agents are needed to satisfy the requirements. In this chapter, we introduce our community computing in detail. We define the terms and then concentrate on the community by specifying the level of communities and the lifecycle of a community.

4.1 Terminology

For better understanding, we introduce in this section the terminology for community computing.

- Ubiquitous Space – A ubiquitous space (U-Space) is a dynamically connected and coordinated set of heterogeneous ubiquitous computing objects. Its boundary is flexible and extendible due to the mobile objects. A ubiquitous object, a smart object in a U-Space, is able to represent various kinds of software and hardware devices and human users.
- Community Computing System – a kind of ubiquitous computing system providing ubiquitous services through communities.
- Society – a metaphor for a community computing system, which is constructed by members and communities.
- Community – a metaphor for a proactive organization comprised of members cooperating with others to achieve particular goals. A community has goals, necessary roles, and information about cooperation and role-member binding. A community is able to have multiple goals and those goals can be issued in parallel. To abstract the types of communities, we describe community templates. A community instance is dynamically created according to the associated community template in execution time.
- Role – a well-defined position in a community with an associated set of expected behaviors [Ferber, 03]. A role represents a particular capability

necessary to achieve community goals. The capability of a role is presented by actions of the role.

- Cooperation – a cooperative interaction among members who take a particular role in a community.
- Member – a metaphor for an agent who belongs to a community computing system. In our community computing, the members are restricted to agents having their own context, capability, and intelligence. If necessary, they can play a role within a community and we call such agents community members. Sometimes, they can take several roles in more than one community simultaneously.
- Role-member Binding – In order to create a community instance, we need to find the best members for each role. We call this process role-member binding.

4.2 Community

Community is the most essential concept of our community computing. For better understanding of the community concept, it is worth introducing the levels and the life cycle of the communities.

4.2.1 Levels of the communities

The required ubiquitous services are able to have different levels of dynamics. Some services need to be dynamically provided according to user's requests, while others need to be continuously offered such as public security services. Besides, some urgent services have to be provided even though they have not been developed, requiring the immediate creation of such services. According to the dynamics of services, we distinguish the levels of communities as static, dynamic, and evolving as shown in Table 3. According to the style of necessary services, a designer can decide the level of a community to create.

Static Community. A static community is the simplest level of communities. In this community, all members and their cooperation are predefined and not changed. A static community is used to provide permanent services without replacing providers. Here are examples of the static community.

- Community of temperature sensors in a building
- Community for residential security including door lock system, monitoring cameras, alarm systems, and the server in the security company.

Dynamic Community. A dynamic community is a more advanced community. In a dynamic community, members who take a community role are changed according to the member's context whenever an instance of community is created. A community can require a role to be performed by the most recent one, the nearest one, or the most preferred one. When a community instance is initiated, the role-member binding is done and the corresponding members can vary depending on the member's context, such as the last execution time, location, or preference. In addition, cooperation among members can be dynamically decided depending on the member's context or the community's situation. A template of a dynamic community can be reused and there will be various instances of it. A dynamic community satisfies the expectation of the community computing partially from the viewpoint of the

dynamism of members and the cooperation process. Still, the template is not adaptable since the goal and roles of a community are predefined before an instance is created like in static communities. Therefore, a more advanced and adaptive community model is required. Here come the examples of the dynamic community.

- A community for finding a missing child, which consists of nearby neighbors, policemen who are on duty, the child's parents, surveillance cameras, and so on. To find the child effectively, the closest member to the child should be selected in the execution time and then that member cooperates with the others in the best way they can.
- A dynamic community for residential security service, which includes door lock systems, house monitoring cameras, alarm systems, the current workers at the security company, and policemen in the vicinity. The security company workers and the policemen will be decided dynamically.

Evolving Community. An evolving community is the most dynamic model of a community. The template of an evolving community is created on demand depending on the situation and the available agents. In addition, the existing template of an evolving community can be adapted and a new template for a specific instance is generated. We anticipate that the community's templates for general service exist and then a template for unexpected service can follow from the existing one by adapting the template to the available agents and situation. In urgent cases, evolving community services would be useful since they can solve the emergency problems even though a system was not prepared for such services at the time of the request. Here are some examples of evolving community.

- The community that handles a traffic accident may include various types of members such as the car involved in the accident, the human user, the emergency service of a hospital to care for the human users, a tow truck, and the insurance company workers to negotiate the cost. Depending on the severity of the accident, the required members will be different. In addition, participating members will be dynamically decided according to the location and time of the accident. Therefore, the required members and the required tasks should be determined on the spot.

	Static Community	**Dynamic Community**	**Evolving Community**
Community Type Definition	Static	Static	On demand
- Role	Static	Static	Dynamic & Adaptable
- Goal	Static	Static	Dynamic
- Cooperation	Static	Dynamic	Dynamic & Adaptable
Community Instance Creation	Static	Dynamic	Dynamic
-Participant member	Static	Dynamic	Dynamic & Adaptable

Table 3: The levels of communities

4.2.2 Lifecycle of community

In our community computing, each community instance has its own lifecycle from initiation to termination. The lifecycle consists of three stages and one optional stage as shown in Figure 1.

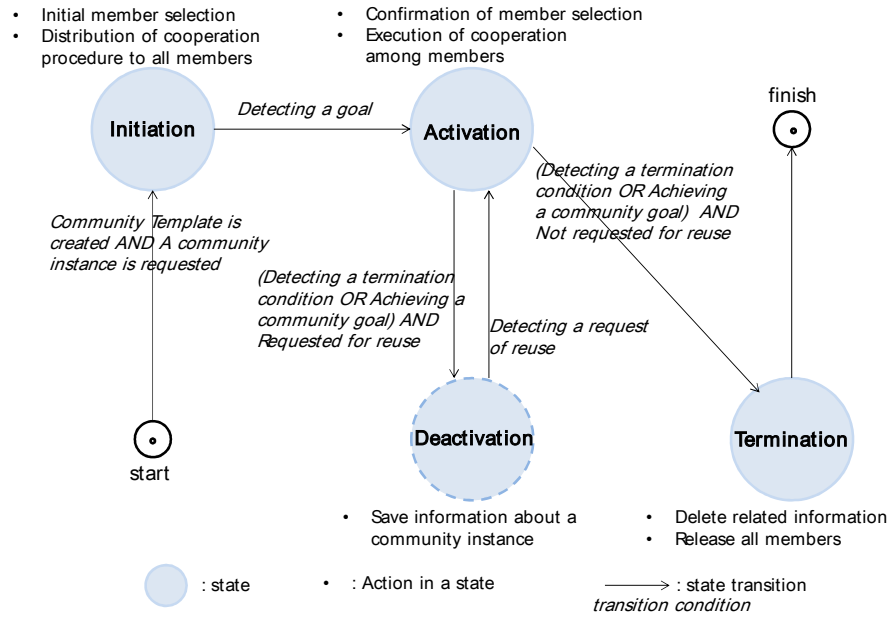

Figure 1: The lifecycle of a community instance

1) Initiation – When a member or a community recognizes a community goal, they can initiate the creation of a community instance. Before the initiation of a community instance, its community template needs to be created. In the initiation stage, members are initially selected for each role and the cooperation process to achieve a community goal is distributed to all members. For a static community, members cannot be changed until the community is terminated. On the other hand, for the dynamic community and evolving community, candidate members are evaluated for be part of a community instance. After the selected candidate accepts a community role, the cooperation process will be deployed to the member. We can say that a community instance is created when the distribution of the cooperation process to all members is completed.
2) Activation – After a community instance is created, it can be activated by detecting a request for a community goal. The activation of a community instance means that all members start to cooperate with each other to achieve their goal. Usually, most communities are activated as soon as they are created. Sometimes a community instance can be re-activated to reuse the instance. In such cases, for a dynamic community, the member selection should be performed again since the condition of members may have changed.
3) Deactivation (optional) – A community instance is deactivated when a cooperation process needs to be paused for a while or if a community instance is expected to be required again. In this state, information about the instance, including selected members and the cooperation process, is saved. Then, this

information will be reused when the community instance is required. Static communities may prefer to be deactivated rather than terminated.

4) Termination – If a community instance needs to be terminated and the community is not expected to reuse it, then the community instance is terminated. The termination of an instance means that information about the instance is deleted from the repository and the cooperation processes of all members are deleted.

4.3 Development Process for Community Computing Systems

To develop a community computing system quickly and conveniently, we need a systematic development process. To accomplish this, we employ the MDA approach. MDA (Model-Driven Architecture) is one of the promising development approaches for very huge and complex application systems. It starts by obtaining the requirement analysis and builds the most high-level model for an application system. Then it refines the model until the model directly specifies the system. In order to follow the MDA approach, we propose three models, CCM, CIM-PI, and CIM-PS, each having different abstraction levels and the model refining process from CCM to CIM-PS.

CCM (Community Computing Model), the most high-level abstraction model, describes how a community computing system satisfies its requirements with communities. For a more detailed specification, CCM is transformed to CIM-PI (Platform Independent Community computing Implementation Model). CIM-PI considers implementation of a system without concern for specific platforms. It describes types of members in a system and refines the community description with member types in detail. Concerning specific platforms, CIM-PI is transformed to CIM-PS (Platform Specific Community computing Implementation Model), which specifies how a system runs on a particular platform. By the model transformation process from CCM to CIM-PS, some portions of the source code are automatically generated but the remaining portions are manually filled by developers. Through the proposed process, we are able to develop a community computing system quickly and systematically. Furthermore, developers can guarantee consistency throughout the entire development process by using a coherent metaphor, that is, community.

5 Situation-Aware Community Computing Model

To design and develop a community computing system, we need a well-defined abstraction model. As we mentioned, however, existing models do not fulfill our need to specify communities and dynamic cooperation in detail. Therefore, in the early stage, we proposed an abstraction model for community computing systems, called the simple community computing model [Jung, 06].

This model supports dynamic role-member binding but cooperation is still static. It uses members‘ contexts to dynamically assign users to community roles. However, cooperation among members is predefined as pseudo codes and it cannot be changed. It means that the simple model did not support dynamic cooperation. Furthermore, all members should know other members’s tasks as well as their assigned tasks to cooperate with each other during cooperation. In case of the huge and complex

cooperation, it is not easy to design such cooperation and it can be a burden for members to know entire cooperation procedure.

In this paper, we therefore propose the situation-aware cooperation model for intuitive design and dynamic execution of cooperation. Using the cooperation model, we propose an improved model, the situation-aware community computing model. Before we describe this model in detail, it is worthy to introduce previous work related to situation and the community situation model.

5.1 Situation Related Work

Context has been broadly researched in various areas of computer engineering. As computing environment has been evolving from distributed computing to mobile computing, and then to ubiquitous/pervasive environment, context has become critical for designing and developing emerging applications [Strang, 04].

Recently, research on context focusing on representing or reasoning dynamic world itself beyond context. As an effort to do so, some researcher have been concentrating on situation. In the early stage, the concept of situation was introduced for situation calculus and used to implement dynamic systems [McCarthy, 69] [Levesque, 97]. Recently, however, situation has been used in pervative computing systems. Yau et al. [Yau, 06] defined situation as an expression of device-action record and/or a set of context information relevant to a system over a period of time. In this work, situation was used to trigger further devices‘ actions. His definition provides more comprehensive understanding of situation by: (i) incorporating the concept of context; (ii) referring to the purpose of situation as a trigger for further actions. That is, it shows the relationship between context and situation, and provides a direction for situation-aware computing. In this paper, we share his viewpoint on situation in ubiquitous computing systems and use the concept of community situation to trigger cooperative tasks of member.

5.2 Community Situation Model

To use community situation, we define the community situation as follows.

> ***Definition (Community Situation).*** *Community situation is a state which needs cooperation among members for resolving problems within a community by capturing the members‘ contexts.*

Diverse information can be used to characterize community situation. Aggregation of members‘ contexts, relational context between members, and configurable and environmental context information are used to define community situation. The member context is a set of knowledge built by capturing information about members within a community. Each context, basically, is represented by name and value pair, for example ‘*time, 3p.m.*’ and ‘*location, 3rd floor*’. Since context modeling and representing issues are not a major contribution of this paper, we assume that a system can match semantic distance between context information of individuals (*i.e. 3 p.m. and 15:00*) by using ontology, semantic matcher, and other related technologies. The aggregation of member context is a Boolean constraint expression which aggregates between member context using three logical connectives (i.e. $\wedge$, $\vee$ and –) and six standard (binary) comparison operators (i.e. $=$, $\neq$, $<$, $>$, $\leq$, and $\geq$). The relational context is (first-order) ontological relationbetween two individuals,

for example "M_a *is father of* M_b" and "M_b *is son of* M_a". The configurable and environmental context is a set of properties about the community itself, for example number of members or community-dependent public knowledge which is shared among members.

5.3 Situation-Aware Cooperation Model

Before we propose the community situation-based cooperation model, we briefly introduce existing cooperation models [Cockburn, 96], [Brazier, 97], [Hua, 03], [Perez, 04], [Guo, 06]. In most cooperation models, cooperation is described as a predefined static program called as recipe, plan, or skill. Therefore, it is hard to make their cooperation dynamic. In addition, the means for designing cooperation itself was not discussed while the means of realizing cooperation are introduced. Therefore, we arrived at a decision that a new cooperation model is needed for intuitive design of dynamic cooperation.

In this paper, we propose the situation-aware cooperation model for community computing. The idea is that cooperation is dynamically executed by recognizing changes of community situations. If a community's situation is changed, then tasks that each member should perform are determined accordingly. Let $S = \{s_0, s_1 \ldots, s_n \mid s_0 = \textit{start situation}, s_n = \textit{goal situation}\}$ be a set of situations. We note that the numbering of situation does not mean a sequence of situation. Changes of situation vary from one to another and it sorely depends on member context. On a given situation $s_{i(0<i<n)}$, each member performs actions specified in the s_i as per their assigned roles. The result of member actions and changes of member context will transit the community situation to s_j, and then members perform actions according to the tasks specified in s_j. The community goals are finally achieved through such transitions in community situation.

This model assumes the awareness of community situation and members' tasks in certain community situation. Prior to defining our cooperation model, we introduce several promises for it.

Assumptions. The situation-aware cooperation model is founded on the following promises.

- Awareness of the community situations – All members in a community should be aware of community situation
- Awareness of tasks of each member in a given community situation – All members should know that which tasks need to perform at a certain community situation
- Ability of multiple tasks execution of a member – In a community situation, a member can perform more than one task in sequential order
- Independent situation change of completion in a previous situation – Although tasks are not completely finished in a previous situation, the situation can be changed into next situation
- Completion of cooperation – Community situations are dynamically changed, but are capable of reaching a situation of community termination

Each cooperation consists of cooperation blocks. A cooperation block describes one piece of entire cooperation in a certain community situation with the definition of the community situation and tasks of roles in the situation. The BNF definition of the situation-aware cooperation model is shown in Figure 2.

<Community Situaton> ::=<Situaiton_Name> <Context_Expresseion>$^{1+}$
<Situation_name> ::=<identifier>
<Context_Expression> ::= <Context> | <Aggregated_Context> | <Relational_Context>
<Context> ::= (<Context_Name> = <Value>)
<Aggregated_Context>::=<Conjunctive_Context>|<Disjuctive_Context>|<Negative_Context>
<Conjunctive_Context> ::= <Context> AND <Context>
<Disjunctive_Context> ::= <Context> OR <Context>
<Negative_Context> ::= NOT <Context>
<Relatioinal_Context> :: <Context> <Relation> <Context>
<Relation> ::= <String>

(a) Definition of member context and community situation

<Situation_Aware_Cooperation_Model>
::= ***Community*** <Community_Type_Name> **{** <Community_Goals_Description> **}**
<Community_Type_Name>::=<Identifier>,
<Community_Goals_Description>::= **Goals** <Goal_Description>$^{1+}$
<Goal_Description>::= <Goal_Name>**(**<Participant_Roles>**) {** <Community_Coopertion> **}**
<Goal_Name>::=<Identifier>, <Participant_Roles>::=<Role_Name>$^{1+}$, <Role_Name>::=<Identifier>
<Community_Cooperation>::=<Cooperation_Block>$^{1+}$
<Cooperation_Block>::=<Community_Situation _Name> **=>** <Role_Task>$^{1+}$
<Role_Task>::=<Role_Name> **:** { <Role_Action_Name> {**(**<Parameter>$^{0+}$**)**}^{opt} }$^{1+}$ **;**
<Community_Situation_Name >::=<Identifier>, <Role_Name>::=<Identifier>
<Role_Action_Name>::=<Identifier>, <Parameter>::=<String>

(b) Situation-aware cooperation model

Figure 2: BNF definition of the situation-aware cooperation model

When a member performs their own actions in a certain community situation, conflicts can occur. In the view of a member, actions that are launched in current situation can conflict with actions that the member already executed when the member plays another role for different community or when actions in previous situation is not finished yet. In the view of a community, an action of a member can conflict with actions of other members. For both cases, the conflicts should be resolved for achieving community goals.

First of all, we assumed that the tasks of a member in a certain community situation are executed sequentially by one thread, and thus we do not need to worry about conflicts on a thread. Conflicts happen when a member tries to execute conflicting own actions or when more than two members try to execute conflicting actions simultaneously. To handle such conflicts, we classify conflicting actions into two categories, namely, *mutually exclusive conflict* and *time dependent conflict*. In the case of the mutually exclusive conflict, when a conflict occurs, then one action among conflicting actions should be terminated. In case of the time dependent conflict, one action among conflicting actions should be executed first, with execution of another action to follow. For handling conflicts in runtime, a community manager and each member have an action-conflicts list about conflicts. In the list, the types of conflicts

are represented. At this time, conflicts between same actions can be included in the list. For example, assume that a member performs action a_2 in community situation S_1. After a few seconds, the situation is changed to S_2, although a_2 is not finished. After that, the situation would be changed again to S_3, and the member should perform a_2 again in a situation S_3. However, a_2, which was executed in the previous situation S_1, would still be operating.

5.4 Situation-Aware Community Computing System Development

In order to make up the drawback of the simple community computing model, we proposed the situation-aware cooperation model and improve the previous model by employing the cooperation model. To develop dynamic community computing systems, in this section, we introduce the family of the situation-aware community computing models including CCM, CIM-PI, and CIM-PS (See 4.3 if you want to know each model). By the model transformation from CCM to CIM-PS as shown in Figure 3, we can develop a community computing system systematically.

In CCM, cooperation of a community is represented by community situations and roles' tasks in each situation. In CIM-PI, each community situation is defined and role-member binding is described. Cooperation process is more detailed with members' actual actions. Furthermore, to manage memberships and resolve conflicts among members' actions, community policy is added, which includes member casting policy, member's secession policy, and action conflicts list. Member casting policy represents a criterion of member selection such as distant dependent casting or response-time dependent casting. Member secession policy specifies that how to handle members' secession. For example, when a member disappears, then the corresponding cooperation can be initialized with a new member, continue with a new member, or be terminated. In member type description of CIM-PI, all member types and their hierarchyare defined. A member type is specified by member's attributes, actions, cast condition, contexts, and policies. When a member simultaneously plays more than one role, conflicts among tasks which required from different communities may happen. To resolve such conflicts, we define an action conflict list in member policy. In addition, we may need to know a priority of community creation if two or more community instances are required at the same time. To deal with such problems, we define the precedence of communities and exclusive communities in society policy.

CIM-PS is generated by combining CIM-PI with platform-specific information such as attribute acquisition, action mapping, and member configuration. In attribute acquisition, we describe the source of member attributes, for example, sensors or output of member action. In action mapping description, we describe that how to execute a member's actions. If a system utilizes existing agents, a developer should make a connection between members' actions in model and real actions of deployed agents. In member configuration, a member's components are additionally described such as its sensor drivers, operating systems, and communication channels. We show the definition of CIM-PS and an example based on COEX-Mall scenario in Appendix A and B.

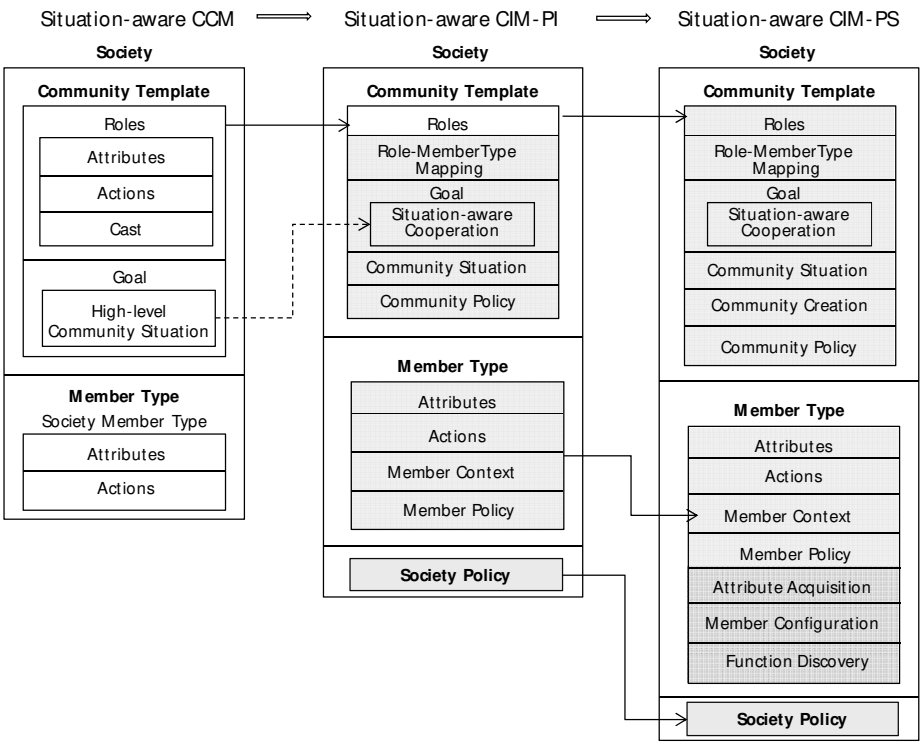

Figure 3: Development process of situation-aware community computing systems by model transformation from CCM to CIM-PS

5.5 Case Study

To examine the proposed model, we introduce two case studies. Let's remind the COEX-Mall scenario in section 2.1 Motivation Examples. As we described, several robots exist in the mall. These robots have various capabilities and each robot offers own services such as guide service or information display at ordinary time. When a community goal arises, robots compose a community instance to achieve the goal. In this section, we present two community's cooperation.

5.5.1 Level-1 Cooperation

When COEX-Mall is opened, an instance of Patrol_COEX community is initiated by casting all robots and guides who are on duty. After the initiation, community situation is entered to the start situation of cooperation, PATROL_RANGE_ASSIGN, as a patrol manager commands to start patrol service. The community instance is activated, as soon as it enters the start situation. In PATROL_RANGE_ASSIGN situation, all members perform Area_Assign(COEX) to negotiate patrolling areas. When the assignment is done, PATROL_BEGIN context is made by a patrol manager and then all robots and guides perform patrolling service. If a robot or a guide cannot patrol any more due to heavy workload or a sudden interruption, they are able to request

reassignment. At this time, the robot can issue PATROL_RANGE_ASSIGN_REQ context and then the situation can be changed again to PATROL_RANGE_ASSIGN situation. If a manager issues PATROL_END context, the community situation is changed to PATROL_END and all members finish their tasks. If the community instance enters to the finish situation, all members are released and the instance is terminated. In Figure 4, we show the situation-aware cooperation for Patrol_COEX community.

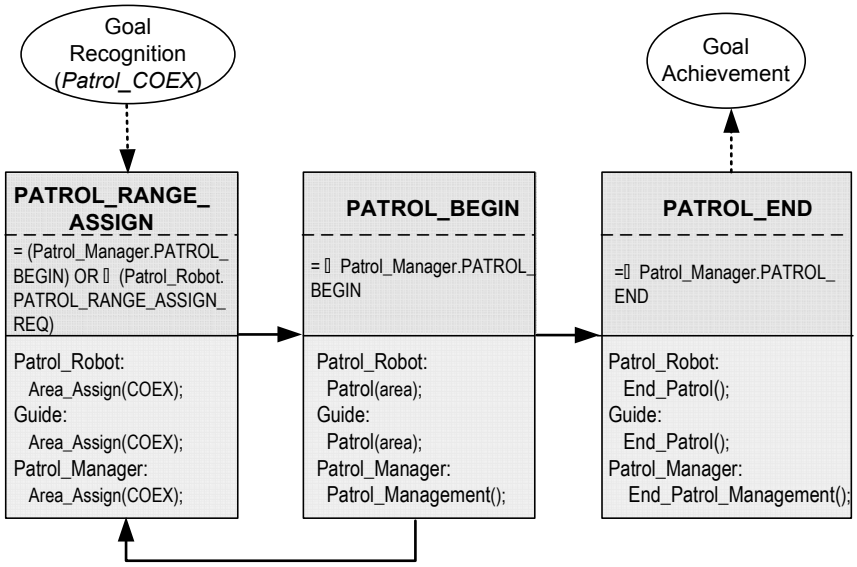

Figure 4: Situation-aware cooperation for Patrol_COEX community

5.5.2 Level-2 Cooperation

When a robot is on patrol as a member of Patrol_COEX community, a mother who lose her son ask to find the missing child. To immediately provide such service, the robot issues TAKE_REQUEST_FIND_PERSON context, and then an instance of corresponding community, Find_Person, is initiated and then activated by entering the start situation, FIND_PERSON_REQUEST. In this situation, all robots share the profile of missing child and then issue FIND_PERSON context. Accordingly, the situation is changed to FIND_PERSON and all members search the child while patrolling. At this time, each robot takes at least two community roles; one role is for Patrol_COEX community and another is for Find_Person community. If there is any member who find the child, PERSON_FOUNDED context is issued and then the community goal is successfully achieved. In the PERSON_FOUNDED situation, all members finish their tasks and then the community is terminated. However, if they fail to find the child for a period of time, they issue PERSON_NOT_FOUNDED. In this situation, a guide reports the present situation to policemen and then a community instance is terminated. In Figure 5, we show the situation-aware cooperation of Find_Person community.

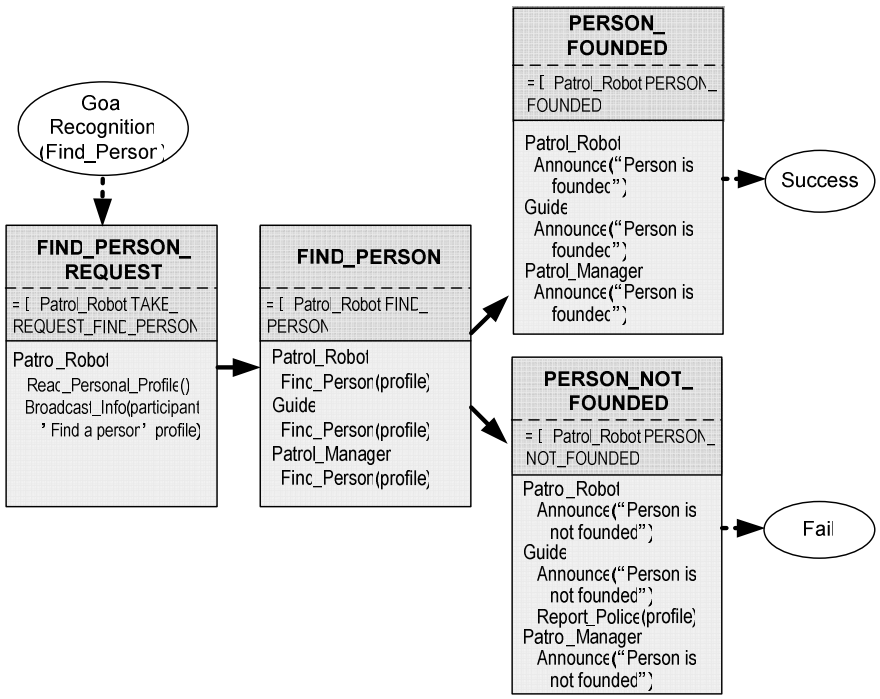

Figure 5: Situation-aware cooperation for Find_Person community

6 Implementation

Since community computing is a development paradigm for ubiquitous computing systems, it is important to examine our work by implementing a system. To help the proposed development process in semi-automatic manner, we implement a developing toolkit, called CDTK. By using CDTK, we conviniently developed two community computing systems based on motivation scenarios presented in chapter 2.

6.1 Community Developing Tool-Kit: CDTK

For systematic and convenient development of community computing systems, we implement Community Developing Tool-Kit (in short, CDTK) shown in Figure 6. To develop CDTK, we use Eclipse platform in order to take the beneficial features of it such as plug-in architecture, perspective views, and bundles. CDTK can give aid to system designers as well as developers by providing intuitive interfaces for community design. We should note that difference in community's level does not lead to a change in CDTK modules themselves. CDTK produces different codes depending on design of communities, but model transformation process in CDTK is same regardless of a community design. Furthermore, it is out of its scope to decide and design the level of required community. Here are main functionalities of CDTK.

- Project Management – A community computing system is managed as a project in CDTK. A user can create a community computing project to design a system, and then save and open it. In addition, a project can be directly imported or exported from/to CCM or CIM-PI file.
- User Interface (UI) for system development – CDTK has a number of UI components: tree, table, text editor, and etc. Entities in community computing systems such as society, community, member, and resources are designed by using such UIs. Such UI helps to reduce overall time for system development as well as provide an efficient overview of system.
- Model Transformation – As described in Section 4.3, we proposed a development process based on model transformation. CDTK supports the model transformation from CCM to CIM-PI, from CIM-PI to CIM-PS, and from CIM-PS to source codes. Each model is converted to more specific model and then developers need to fill the rest part with newly required information. Model transformation function maintains the consistency between different models as well as reduces development time.
- (Semi) Automatic Runtime Code Generation – CDTK semi-automatically generates run-time codes through the model transformation from CIM-PI to source codes. In the present version, CDTK generates java codes for Jade[1] platform. The code generation can be extended to other agent platforms such as Cougaar[2], Jack[3], and AgentBuilder[4] by developing generation module or plug-in on CDTK. Currently, the rate of automatic code generation is around 60%.

6.2 Simulation Result

By using CDTK, we develop two community computing systems based on two scenarios presented in section 2.1; CHILDCARE and COEX-Mall.

CHILDCARE. When Tom goes far, his smart watch requests for a community instantiation to the society manager agent which maintains a system. Then, the society manager creates a community manager to initiate an instance of CHILDCARE community by selecting best members who take community roles. After the initiation, the smart watch informs child's location to family, and searches for the nearest person who can help. While a neighbour goes to the child, the closest surveillance camera sends the image of child to the selected neighbour and family. Finally, when the child arrives at home, the community goal is achieved and then the community is disorganized. In Figure 7, the simulation result of the CHILDCARE community computing system is shown. The presented screenshot shows member casting process to gather best members for each community role which defined in the corresponding community template. As you can see, a community manger is communicating with each member to comprise a community instance and achieve its goal.

---

[1] http://jade.tilab.com
[2] http://www.cougaar.org/
[3] http://www.agent-software.com.au/
[4] http://www.agentbuilder.com/

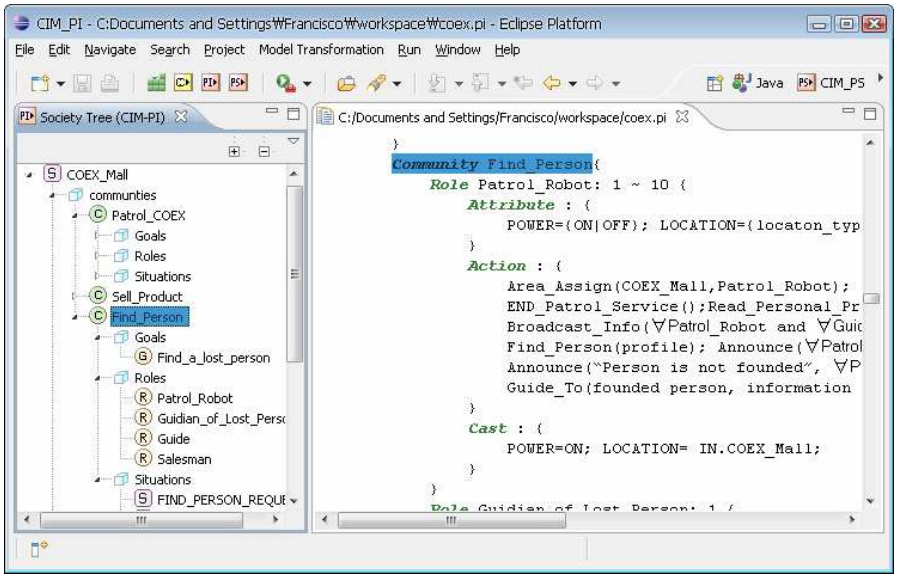

Figure 6: CDTK, Community Developing Tool-Kit. A developer is developing Find_Person community by using Society Tree and Script Editor.

COEX-Mall. To examine the proposed situation-aware community computing model, we developed COEX-Mall system to offer patrolling service and missing child care service. Situation-aware cooperation process for the Patrol_COEX community and Find_Person community are already shown in Figure 4 and 5. In this system, ARGUS is a patrol robot in COEX-Mall. For providing situation-aware services, we assume that ARGUS is able to recognize changes of community situation and knows required tasks in each community situation. Moreover, we assume that ARGUS performs required actions in sequence only. Figure 8 shows the simulation result of COEX-Mall system in which four ARGUSs cooperate with each other to find a missing child during patrolling an assigned area in COEX-Mall.

Description
#1 Member registration 1. Child, family of the child, neighbors, and other objects should register with GHODAMCITY' society. 2. To provide a service for protecting children, a 'CHILDCARE' community is designed
#2 Request for community creation 1. A child is in dangerous area. 2. Child's smart watch senses danger and then requests to create an instance of 'CHILDCARE' community to society manager 3. A society manager creates a community manager of 'CHILDCARE' community
#3 Initiation and activation of community instance 1. 'CHILDCARE' community manager initiate a 'CHILDCARE' community instance by selecting best member for all community roles. 2. After initiation, community situation immediately enters to start situation, then the community is activated and its cooperation begins.
#4 Termination of 'CHILDCARE' community instance 1. Susan, who takes a community role of neighbor 2, brings Tom to his mother. 2. By achieving a community goal, the community instance is terminated.
Screenshot

Figure 7: Simulation of CHILDCARE community computing system

Description
#1 COEX_Mall Society Creation 1. COEX_Mall system is started, and COEX-mall society is created. 2. All members in the system are registered with COEX-mall society.
#2 Initiation of an instance of 'Patrol_COEX' community 1. As ARGUS are turned on, it is requested to initiate a 'Patrol_COEX' community instance. 2. A society manager of the COEX-mall generates a community manager of 'Patrol_COEX' community, then the manager selects all members for each role to initiate 'Patrol_ COEX' community 3. By gathering all best members, an instance of 'Patrol_COEX' community is initiated
#3 Activation of 'Patrol_COEX' community instance. As community situation enters to start situation, the instance is activaed and all ARGUS and guides begin patrolling service
#4 Initiation of 'Find_ Person' community instance (Figure 8-(a)) While patrolling, an ARGUS receives a request for finding a missing child, so it requests to initiate a community instance of 'Find_Person' community.
#5 Activation of an instance of 'Find_ Person' community All ARGUS are involved in an instance of 'Find_Person', then they start to find the child. (Figure 8-(b)) At this time, all ARGUS are taking at least two roles from 'Patrol_ COEX' community and 'Find_Person' community
#6 Termination of 'Find_ Person' community instance.(Figure 8-(c)) When an ARGUS finds the missing child using its vision sensing, the 'Find_Person' community instance is disorganized. By the way, 'Patrol_COEX' community instance keeps running. (Figure 8-(d))
Screenshot   (a) (b) (c) (d)

Figure 8: Simulation of COEX-Mall community computing system

7 Conclusion

Our goal is to quickly and conveniently develop ubiquitous computing systems, which dynamically provide useful services, even complex, large-scale, and unpredictable services. To achieve our goal, we present four requirements that should be satisfied in a ubiquitous system; context-awareness, cooperation, utilization of existing objects, and possession of a model to design and develop a system. Since existing work in various fields does not achieve these goals, in this paper, we propose community computing as a new development paradigm for ubiquitous computing systems where services are provided through cooperation among existing smart agents.

Our community computing fulfills the four requirements successfully. In the community computing models, a member's context is represented and utilized when role-member binding and cooperation are performed. By using communities among agents, a system can support cooperation to provide services, which require diverse and large-scale capabilities. In the simple community computing model, there is no cooperation model, so cooperation is described as a pseudo code. However, in the situation-aware model, we are able to guarantee dynamic cooperation as well as to quickly and intuitively design cooperation by using the situation-aware cooperation model. To define this cooperation model, we also propose the community situation model and security policies to resolve conflicts. Furthermore, we can easily use existing agents to develop ubiquitous services, since all of them can be represented as members in the model. In addition, to make designing and developing community computing systems convenient and fast, we propose two abstraction models, the simple model and the situation-aware model, along with the development process based on MDA approach. For systematic implementation of a community computing system, we also develop a development toolkit, called CDTK. Using this toolkit, we developed two small systems and presented the simulation results to verify the feasibility of two community computing models; the simple model and the situation-aware model.

Despite our progress, our proposal suggests several avenues for future work.

1) Study on evolving communities – The situation-aware community computing model only can support the static community and the dynamic community. Therefore, we need to find a way to develop evolving communities and use them.
2) Improvement of the situation-aware cooperation model and the situation model - This version of the model is based on strong assumptions so it requires a model which can deal with situation-awareness and cooperation simultaneously without strong assumptions.
3) The security of a community computing system – To use our community computing in practice, security should be guaranteed. Research on security should be required, such as authentication, access control, authorization, privacy, and trust.
4) Various case studies – To expand the area of applications, we continuously apply our research to various problem domains.

References

[Al-Muhtadi, 04] Al-Muhtadi, J., Chetan, S., Ranganathan, A., Campbell, R. H.: Super Spaces: A Middleware for Large-Scale Pervasive Computing Environments, Perware '04, IEEE Press, Orlando, 198-202, 2004.

[Baldauf, 07] Baldauf, M,. Dustdar, S., Rosenberg, F.: A Survey on Context-aware Systems, Int. Journal of Ad Hoc and Ubiquitous Computing, Vol. 2 No. 4, 263-277, 2007

[Besselaar, 02] Besselaar, P., Tanabe, M., Ishida, T.: Introduction: Digital Cities Research and Open Issues, Lecture Notes in Computer Science Vol. 2362, Springer-Verlag, 1-9, 2002.

[Blau, 05] Blau, J.: Microsoft: Community computing is on the way; InfoWorld Magazine, 2005, http://www. infoworld.com/article/05/11/22/HNcommunitycomputing_1.html

[Borghoff, 00] Borghoff, U. M, Schlichter, J. H.: Computer-Supported Cooperative Work: Introduction to Distributed Applications, Springer-Verlag Berlin, 2000.

[Bortenschlager, 09] Bortenschlager, M., Castelli, G., Rosi, A., Zambonelli, F.: A Context-Sensitive Infrasturcture for Coordinating Agents in Ubiquitous Environments, Journal of Multiagent and Grid Systems, Vol. 5 No. 1, 1-18, 2009

[Brazier, 97] Brazier, F. M. T., Jonker, l., et al.: Formalization of a cooperation model based on joint intentions, Proc. of the Third International Workhop on Agent Theories, Architectures and Languages(ATAL'96), Lecture Notes in Artificial Intelligence, Vol. 1193, 141-155, 1997.

[Cabri, 03] Cabri, G., Leonardi, L., Zambonelli, F.: A Framework for Flexible Role-based Interactions in Multi-agent System, Proc. Conf. on Cooperative Information Systems (CoopIS), Italy 2003.

[Cockburn, 96] Cockburn, D., Jennings, N. R.: ARCHON: A Distributed Artificial Intelligence System for Industrial Applications, Foundation of Distributed Artificial Intelligence, Wiley, 319-344, 1996.

[Dey, 01] Dey, A.K.: Understanding and Using Context, Personal and Ubiquitous Computing - Special Issue on Situated Interaction and Ubiquitous Computing, Vol. 5, No. 1, 4-7, 2001.

[Ejigu, 07] Ejigu, D., Scuturici, M., Brunie, L.: CoCA: A Collaborative Context-Aware Service Platform for Pervasive Computing, In Proc. Third Int. Conf. Information Technology: New Generations (ITNG'07), 297-302, Las Vegas, USA, April 2007.

[Ejigu, 08] Ejigu, D., Scuturici, M, Brunie, L.: Hybrid Approach to Collaborative Context-Aware Service Platform for Pervasive Computing, Journal of Computers, Vol. 3, No. 1, 40-50, 2008

[Ferber, 98] Ferber, J., Gutknecht, O.: A meta-model for the analysis and design of organization in multi-agent systems, Proc. 3rd Int. Conf. on Multi-agent Systems (ICMAS'98), 1998.

[Ferber, 03] Ferber, J., Gutknecht, O., Michel, F.: From Agents to Organizations: An Organizational View of Multi-agent Systems, Proc. AOSE 2003, Australia 2003.

[Guo, 06] Guo, H, Gao, J., et al.: Recipe, Policy and Self-Organizing: A Hybrid Collaboration Approach for Agent-based Cooperative Design, Proc. of the 10th Int. Conf. on Computer Supported Cooperative Work in Design, 2006

[Hua, 03] Hua, C, Gao, J., et al.: AGDRSCOM: A complicated Dynamic Real-time Strong Cooperation System Model, Proc. of the Second Int. Conf. on Machine Learning and Cybernetics, 318-323, 2003

[Ishida, 98] Ishida, T.: Community Computing and Support Systems, Lecture Notes in Computer Science Vol. 1519, Springer-Verlag, 1998

[Ishida, 02] Ishida, T.: Digital city Kyoto, Communication of the ACM, Vol. 45, No. 7, 76-81, 2002.

[Jennings, 03] Jennings, N. R., et al.: Developing Multiagent Systems: The Gaia Methodology, ACM Transactions on Software Engineering and Methodology, Vol. 12, No. 3, 317-370, 2003.

[Johansen, 98] Johansen, R.: Groupware: Computer Support for Business Teams, New York 1998

[Jung, 06] Jung, Y., Lee, J,. Kim, M.: Multi-agent based Community Computing System Development, Proc. Int. Joint Conf. Autonomous Agents and Multiagents Systems (AAMAS'06), 1329-1331, Hakodate, Japan, 2006

[Kindberg, 02] Kindberg, T., Fox, A.: System Software for Ubiquitous computing, IEEE Pervasive computing, Vol. 1, No.1, 70-81, 2002.

[Kumar, 03] Kumar, M., Shirazi, B., Das, S. K., Singhal, M., Sung, B., Levine D.: Pervasive Information Communities Organization PICO: A Middleware Framework for Pervasive Computing, IEEE Pervasive Computing, July-September, 72-79 2003

[McCarthy, 69] McCarthy, J., Hayes, P. J.: Some Philosophical Problems from the Standpoint of Artificial Intelligence, Machine Intelligence, Vol. 4, 463-502, 1969.

[Microsoft, 05] Microsoft community technologies research group, 2005, http://research.microsoft.com/community/

[OMG, 03] Object Management Group, Model Driven Architecture (MDA) Guide 2003

[Perez, 04] Perez, M. S., Sanchez, A., et al.: Cooperation Model of a Multiagent Parallel File System for Clusters, Proc. IEEE Int. Symp. Cluster Computing and the Grid, 595-601, 2004

[Reiter, 97] Reiter, R.: The situation Calculus Ontology, Electronic News Journal on Reasoning about Actions and Changes, 1997, (http://www.ida.liu.se/ext/etai/rac/notes/1997/09/note.html)

[Roman, 00] Roman, M., Campbell, R. H.: GAIA: Enabling Active Spaces, Proc. 9th ACM SIGOPS European Workshop, Kolding, Denmark, 229-234, 2000.

[Salber, 99] Salber, D., Dey, A.K., Abowd, G.D.: The Context Toolkit: Aiding the Development of Context-enabled Applications, In Proc. SIGCHI Conf. Human Factors in Computing Systems: the CHI is the Limit, 434-441, 1999.

[Schilit, 94] Schilit, B., Adams, N., Want, R.: Context-aware Computing Applications, In Proc. Workshop on Mobile Computing Systems and Applications, 85-90, 1994.

[Schmidt, 99] Schmidt, A., Beigl, M., Gellersen, H.W.: There is more to Context than Location, Computers and Graphics, Vol. 23 No.6, 893-901, 1999

[Strang, 04] Strang, T., LinnhoffPopien, C.: A Context Modeling Survey, In Proc. Int. Workshop on Advanced Context Modeling, Reasoning and Management, in conjunction with UbiComp 2004, Nottingham, UK, September 2004

[Sung, 02] Sung, B., Shirazi, B., Kumar, M.: Pervasive Community Organization, Eurasia2002, Tehran, November 2002.

[Takahashi, 09] Takahashi, H., Suganuma, T., Shiratori, N.: AMUSE: An Agent-based Middleware for Context-aware Ubiquitous Services, In Proc. Int. Conf. Parallel and Distributed Systems (ICPADS'05), 743-749, Fukuoka, Japan, July 2005.

[Weiser, 91] Weiser, M.: The Computer for the Twenty-First Century, Scientific American, 1991.

[Wilson, 91] Wilson, P., et al.: Computer Supported Cooperative Work, Oxford, UK, Intellect Books, 1991.

[Wooldridge, 00] Wooldridge, M., Jennings, N. R.: The Gaia Methodology for Agent-oriented Analysis and Design, Autonomous Agents and Multi-Agent Systems, 285-312, 2000.

[Wooldridge, 02] Wooldridge, M.: An Introduction to Multiagent Systems, John Wiley & Sons, 2002.

[Wooldridge, 99] Wooldridge, M., Jennings, N. R.: The Cooperative Problem-Solving Process, Journal of Logic computation, Oxford University Press, Vol. 9 No. 4, 563-592, 1999.

[Yau, 06] Yau, S.S, Liu, J.: Hierarchical Situation Modelling and Reasoning for Pervasive Computing, Proc. Int. Workshop on Software Technologies for Future Embedded and Ubiquitous Systems and Int. Workshop on Collaborative Computing, Integration, and Assurance (SEUS-WCCIA'06), 5-10, 2006.

[Zambonelli, 00] Zambonelli, F., Jennings, N. R., Wooldridge, M.: Developing Multiagent Systems: The Gaia methodology. ACM Transactions on Software Engineering and Methodology (TOSEM), Vol. 12, No.3, 317-370, July 2003.

Appendix

A. BNF definition of situation-aware community computing model: CIM-PS

<Static_Community_Situation_based_Community_Computing_Model_Description>
::= ***Society*** <Society_Name> **{**<Community_Type_Description><Member_Type_Description>
<Society_Policy_Description> **}**
<Society_Name>::=<Identifier>
<Community_Type_Description>::= ***Community Template Description {*** <Community_Type>$^{1+}$***}***
<Community_Type>::= ***Community*** <Community_Type_Name>**{** <Role_Description>$^{1+}$
<Role_Member_Binding><Community_Goals_Description><Community_Situations_Description>
<Community_Creation_Description><Community_Policy_Description><Ontology_Description>^{opt} **}**
<Community_Type_Name>::=<Identifier>
<Role_Description>::= ***Role*** <Role_Name>**:** <Role_Cardinality> **{** <Attributes_Description>
<Actions_Description><Cast_Conditions_Description>**}**
<Role_Name>::= <Identifier>, <Role_Cardinality>::=<Min_Cardianality> **~** <Max_Cardinality>
<Min_Cardianality>::=<digit>$^{1+}$, <Max_Cardinality>::=<digit>$^{1+}$
<Attributes_Description>::= ***Attribute*** **:** {<Attribute_Name>**=**
{<Attribute_Value>|<Attribute_Value_Type_Name>};}$^{1+}$
<Attribute_Name>::=<Identifier>, <Attribute_Value_Type_Name> ::=<Identifier>
<Attribute_Value>::=<String>|<Range_Value>|<Selective_Value>
<Range_Value>::=**(**<MinValue>**,**<MaxValue>**)** , <MinValue>::=<digit> , <MaxValue>::=<digit>

<Selective_Value>::= **{**<String> {**|**<String>}$^{1+}$**}**
<Actions_Description>::= ***Action*** **:** {<Action_Description>}$^{1+}$
<Action_Description>::=<Action_Name>**(**<Action_Parameters>^{opt}**);**
<Action_Parameters>::=<Parameter>{,<Parameter>}$^{0+}$, <Parameter>::=<String>
<Cast_Conditions_Description>::=***Cast*** **:**{<Attribute_Name>**=**<Attribute_Value>**;**}$^{1+}$
<Role_Member_Binding>
::= ***Role-MemberType Mapping*** ***{***{<Role_Name>**:**<Member_Type_Name>$^{1+}$**;**}$^{1+}$ **}**
<Community_Goals_Description>::= **Goals** <Goal_Description>$^{1+}$
<Goal_Description>::= <Goal_Name>**(**<Participant_Roles>**) {**<Community_Coopertion>**}**
<Goal_Name>::=<Identifier>, <Participant_Roles>::=<Role_Name>$^{1+}$, <Role_Name>::=<Identifier>
<Community_Cooperation>::=<Cooperation_Block>$^{1+}$
<Cooperation_Block>::=<Community_Situation_Name> **=>** <Role_Task>$^{1+}$
<Role_Task>::=<Role_Name> **:** { <Action_Name> **(**<Parameter>$^{0+}$**);** }$^{1+}$
<Community_Situation_Name>::=<Identifier>
<Community_Situations_Description>
::=***Community Situation*** ***{***<Community_Situation_Description>$^{1+}$***}***
<Community_Situation_Description>::= <Community_Situation_Name> **=**
{ {<Role_Situation> | <Community_Situation_Expression> }**};**
<Role_Situation>::=<Unary_Logical_Operator>^{opt} <Role_Name>**.**<Member_Situation_Name>
<Unary_Logical_Operator>::=<NOT_Operator>, <NOT_Operator>::= **NOT**
<Member_Situation_Name>::=<Identifier>
<Community_Situation_Expression>
::=**{**<Community_Situation_Item><Binary_Logical_Operator><Community_Situation_Item>**}**
<Community_Situation_Item>::= <Role_Situation> | <Community_Situation_Expression>
<Binary_Logical_Operator>::= **AND** | **OR**
<Community_Creation_Description>::=***Community Creation{*** <Community_Creation_by_Member>
<Community_Creation_by_Community> ***}***
<Community_Creation_by_Member>::=***By Member:***
{<Member_Type_Name><Member_Situation_Name> }^{opt}
<Community_Creation_by_Community>::=***By Community:***
{<Community_Type_Name><Community_Situation_Name> }^{opt}
<Community_Policy_Description>::= ***Community Policy {*** <Member_Casting_Policy_Description>
<Member_Secession_Policy_Description><Action_Conflict_in_Community> ***}***
<Member_Casting_Policy_Description>::= ***Member Casting Policy {*** <Member_Casting_Policy>$^{1+}$ **}**
<Member_Casting_Policy>::=<Role_Name>**:**<Casting_Policy>**;**
<Casting_Policy>::= **distance_dependent** | **response_time_dependent**
<Member_Secession_Policy_Description>
::= ***Member Secession Policy{***<Member_Secession_Policy>$^{1+}$ **}**
<Member_ Secession_Policy>::=<Role_Name>**:**< Secession_Policy>**;**
< Secession_Policy>::= **continue with a new** | **initialize with a new** | **community fail**
<Action_Conflict_in_Community>::=***Action Conflicts List*** **=**
{<Mutual_Exclusive_Conflicts>|<Time_Dependent_Conflicts>**}**
<Mutual_Exclusive_Conflicts>::=**MEC**
(<Role_Name>.<Action_Name>**,**<Role_Name>.<Action_Name>**)**
<Time_Dependent_Conflicts>::=**TDC (**<Role_Name>.<Action_Name>**,**<Role_Name>.<Action_Name>**)**

```
<Ontology_Description>::=<Ontology_Name>, <Ontology_Name>::=<Identifier>
Member_Type_Description>::= Member Type Description { <Member_Type>1+}
<Member_Type>::=Member <Member_Type_Name>{extends <Parent_Member_Type_Name>}opt
        { <Attributes_Description><Actions_Desciption>{<Cast_Conditions_Desciption>}opt
        <Member_Contexts_Description><Member_Configuration><Attribute_Acquisition>
        <Action_Mapping><Member_Policy_Description> }
<Parent_Member_Type_Name>::=<Identifier>
<Member_Contexts_Description>::=Member Context {<Member_Context_Description>1+}
<Member_Context_Description>::= <Member_Context_Name> :
   {<Singular_Member_ Context >|< Member_ Context _Expression>};
<Singular_Member_ Context >::=<Unary_Logical_Operator> opt <Attribute_Name>=<Attribute_Value>
< Member_ Context _Expression>
   ::={< Member_ Context _Item><Binary_Logical_Operator>< Member_ Context _Item>}
< Member_ Context _Item>::=<Singular_Member_ Context > | < Member_ Context _Expression>
<Binary_Logical_Operator>::= AND | OR
<Member_Configuration>::= Member Configuration =
   { <Sensor_Driver_Name>0+<OS><Communication_Channel>0+ }
<Attribute_Acquisition>::= Attribute Acquisition
   {<Attribute_Name> : {<Sensor_Name>|<Action_Name>} }
<Sensor_Name>::=<Identifier>
<Action_Mapping>::= Action Mapping {<Action_Name_in_Model>:<Action_Name_in_Member> }
<Action_Name_in_Model>::=<Identifier>, <Action_Name_in_Member>::=<Identifier>
<Member_Policy_Description>::= Member Policy { <Action_Conflict_in_Member> }
<Action_Conflict_in_Member>::=Exclusive Actions =
   {<Mutual_Exclusive_Conflicts>|<Time_Dependent_Conflicts>}
<Society_Policy_Description>::= Society Policy
   { <Community_Precedence_Description><Exclusive_Community_Description>}
<Community_Precedence_Description>::= Community Precedence
     { High_Priority : <Community_Name>0+; Medium_Priority : <Community_Name>0+;
       Low_Priority : <Community_Name>0+; }
<Exclusive_Community_Description>::= Exclusive Community
{ {(<Community_Name>,<Community_Name>)}0+ }
```

B. Example of situation-aware community computing model: CIM-PS

```
Society    COEX_Mall {
Community Template Description {
 Community Patrol_COEX{.....}
 Community Find_Person{
  Role Patrol_Robot: 1 ~ 10 {
     Attribute:POWER={ON|OFF}; LOCATION={location_type}; MODE={ BUSY|ORDINARY};
     Action:Area_Assign();Patrol();
     Cast : POWER=ON; LOCATION= IN.COEX_Mall;}
  Role Patrol_Manager : 1~ 2{
     Attribute:STATUS={ON DUTY| OFF DUTY}; LOCATION={ location_type};
     Action:Patrol_Management();
     Cast : STATUS=ON DUTY; LOCATION=IN.COEX_Mall;}
```

Role Guide **:** 1**~** 5**{ ... }*.... }***
***Role-MemberType Mapping* {**
Patrol_Robot:ARGUS**; Guidian_of_Lost_Person:**Human**; Guide:**Guide**;Salesman:**Human**; }**
Goals Find_a_lost_person(Patrol_Robot, Guidian_of_Lost_Person, Guide, Resident)
{*FIND_PERSON_REQUEST*=>
Patrol_Robot **:** Read_Personal_Profile(); Broadcast_Info(∀Patrol_Robot and ∀Guide and ∀Resident, "Find a person", profile);;
***FIND_PERSON*=>**
Patrol_Robot **:** Find_Person(profile);
Guide**:**Find_Person(profile); Salesman **:** Find_Person(profile);
***PERSON_FOUNDED*=>**
Patrol_Robot and Guide and Salesman **:** Announce(∀Patrol_Robot and ∀Guide and ∀Resident**,** "Person is founded", location);
Guide_To(founded person, information office);;
***PERSON_NOT_FOUNDED*=>**
Patrol_Robot and Guide and Resident **:** Announce("Person isn't founded",∀Patrol_Robot);;
Guide**:** Report_Police(" lost person", profile);;**}**
***Community Situation* {**
FIND_PERSON_REQUEST={ Patrol_Robot.TAKE_REQUEST_FIND_PERSON};
FIND_PERSON={Patrol_Robot.FIND_PERSON};
PERSON_FOUNDED={ Patrol_Robot.PERSON_FOUNDED OR Guide.PERSON_FOUNDED OR Resident.PERSON_FOUNDED};
PERSON_NOT_FOUNDED={ Patrol_Robot.PERSON_NOT_FOUNDED AND Guide.PERSON_NOT_FOUNDED AND Resident.PERSON_NOT_FOUNDED }**;}**
Community Creation {
By Member: ARGUS.TAKE_REQUEST_FIND_PERSON**;**
***By Community:* }**
***Community Policy* {**
***Member Casting Policy* {**
Patrol_Robot: distance-dependent**; Salesman:**distance-dependent**;**
Guide: distance-dependent; **}**
***Sudden Secession of Member* {**
Patrol _Robot: continue with a new**; Salesman:** continue with a new**;**
Guidian_of_Lost_Person: initialize with a new**; Guide: c**ontinue with a new**; }**
***Action Conflicts List*={ MEC(**Report_Police(" lost person", profile),Find_Person(profile))**; }}**
***Ontology* :** Patrol_COEX_Ontology**; } }**
***Member Type Description* {**
Member COEX_MallTIZEN **{**
Attribute : LOCATION=IN.COEX_Mall**; }**
Member Animate Object ***extends*** COEX_MallTIZEN
{ ……..}
Member ARGUS ***extends*** Robot **{**
Attribute *:* MODEL=STRING**;**FIND_PERSON={YES|NO}; PERSON_FOUNDED={YES|NO}; TAKE_REQUEST_FIND_PERSON={YES|NO};
Actions Area_Assign(COEX_Mall, Patrol_Robot); Patrol(COEX_Mall); END_Patrol(); Read_Personal_Profile(); Broadcast_Info(∀Patrol_Robot and ∀Guide and ∀Resident,"Find a person", profile); Find_Person(profile); Announce(∀Patrol_Robot and ∀Guide and ∀Resident**,** "Person is founded", location);Guide_To(founded person, information office);Announce("Person is not founded", ∀Patrol_Robot);
***Member Situation* {**
TAKE_REQUEST_FIND_PERSON:TAKE_REQUEST_FIND_PERSON=YES;
FIND_PERSON:FIND_PERSON=YES; PERSON_FOUNDED:PERSON_FOUNDED=YES;
PERSON_NOT_FOUNDED:PERSON_NOT_FOUNDED=YES;**}**
***Member Configuration*={** Vision_Sonsor_v3; Samsung_Location_Sensor_v1;**}**

```
    Attribute Acquisition {TAKE_REQUEST_FIND_PERSON:Vision_Sonsor_v3;}
    Action Mapping {
      Area_Assign(COEX_Mall,Patrol_Robot):Set_patrol_range(location);
      Patrol(COEX_Mall):CyberCap(patrol);END_Patrol_Service():CyberCap(patrolstop);
      Read_Personal_Profile():Read_RFID(person_RFID);
      Broadcast_Info(∀Patrol_Robot and ∀Guide and∀Resident, "Find a person", profile):
      BroadCasttowhom, msg); find_Person(profile):Search_Obj(Info);
      Announce(∀Patrol_Robot and ∀Guide and ∀Resident, "Person is founded", location):
      Notify(towhom,msg);Guide_To(founded person, information office):GuideServie(who,where);
      Announce("Person is not founded", ∀Patrol_Robot):Notify(msg,towhom);}
    Member Policy {
      Exclusive Actions={
    MEC(Patrol(COEX_Mall),END_Patrol_Service()); } } }        ...     }
Society Policy {
    Community Precedence {
      High_Priority: Find_Person;
      Medium_Priority: Patrol_COEX, Sell_Product'
      Low_Priority:;}
    Exclusive Community = { } }   }
```