Telecommun Syst (2012) 51:137–146
DOI 10.1007/s11235-011-9423-6

Mobile agent based framework for mobile ubiquitous application development

Seungkeun Lee · Kuinam Kim

Published online: 25 March 2011

Abstract Ubiquitous computing which enabled by the availability of mobile, heterogeneous devices that supply context information, is currently not matured by the lack of programming support for the design and development of context-aware applications. Especially, ubiquitous computing environment is not static which can be compensable dynamically according to need of environment. Mobile agent is a very efficient framework applications can cooperative in heterogeneous environment. Therefore, we have developed a mobile agent based framework that significantly eases the development of mobile, context-aware applications. The framework allows developers to fuse data from disparate sensors, represent application context, and reason efficiently about context, without the need to write complex code. An event based communication paradigm designed specifically for ad-hoc wireless environments is incorporated, which supports loose coupling between sensors, actuators and application components.

Keywords Mobile agent · Mobile ubiquitous computing

1 Introduction

This Context awareness and mobility are core concepts in the vision of ubiquitous computing where networks of small computing devices are dispersed in the physical environment, operating autonomously and independently of centralized control. Context-aware applications are a large and important subset of the overall set of ubiquitous computing applications, and have already demonstrated the advantages gained from the ability to perceive the surrounding environment. Such applications however remain difficult to develop and deploy, with no widely accepted programming model available. Programmers are often required actuator devices at a low level in order to develop relatively simple applications [1–3].

The mobility of devices in the ubiquitous computing environment also raises challenges in the areas of communication and interaction due to factors such as dynamically changing network addresses and system configurations, susceptibility to disconnection and low bandwidth [4].

The main components of a context-aware application are a set of sensors for capture of context data, a set of rules governing behavior according to context and a set of actuators for generating responses. We have developed the sentient object model for the development of context-aware applications in an ad-hoc mobile environment, which defines software abstractions for sensors and actuators, and provides a framework for the specification of production rule driven behavior. Sentient objects have a number of characteristics that are important in ubiquitous computing environments:

- Sentience—the ability to perceive the state of the environment via sensors
- Autonomy—the ability to operate independently of human control in a decentralised manner
- Proactiveness—the ability to act in anticipation of future goals or problems.

The framework fulfills the two major goals identified by Dey and Sohn [5] that are necessary for the successful development of ubiquitous, context-aware applications, namely:

S. Lee (✉)
Joah Cooperation AG, Industriestrasse 30, 8302, Kloten, Switzerland
e-mail: lee@joah.ch

K. Kim
Kyonggi University, 94-6, Yiuidong, Yeongtonggu, Suwon, Kyonggido 443-760, Republic of Korea
e-mail: harap123@daum.net

- Applications are easier to design, prototype and test, supporting a faster iterative development process
- Designers and end-users are empowered to build their own applications A number of other middleware proposals address the challenges of effectively developing context-aware applications.

Seminal work by Dey and Sohn [5] provided a toolkit which enabled the integration of context data into applications, but did not provide mechanisms for performing sensor fusion, reasoning about context, or dealing with mobility. Context acquisition and use was often tightly integrated into a single application [2], and could not easily be incorporated into other applications. Other work provided mechanisms for reasoning about context [1, 6], but still did not provide a well-defined programming model and did not address the challenges of mobility. Recent and ongoing work [7–9] provides programmer support for the development of context-aware applications, but does not provide the ability to systematically specify and manage event filtering, sensor fusion and rule-based inference in a mobile ad-hoc environment, as our framework does.

2 Common requirements of context aware frameworks

From the functionalities of the frameworks studied above, we came up with the common set of requirements that any context aware framework satisfies:

- Sensor technology to capture the contextual information: Acquire raw contextual information
- Support for event based programming model so as to have the ability to trigger events when certain context change is observed.
- A way to communicate the sensed contextual data to other elements in the environment and a way to interpret the collected data: provide interpreted context to application.

Support to build context aware applications is provided by various infrastructures indicated above among which the two promising supporting infrastructures for our thesis are: The Context Toolkit and the Java Context Aware Framework. JCAF would have been a better choice for our thesis, since it supports an Application Programming Interface (API), which can be used directly to handle contextual changes and since it is based on deployment in an organization. However, it is relatively new and is under active development. Being a relatively new proposal, JCAF has no proper documentation as of now. Context toolkit is relatively well established and the first version is now publicly available. Hence, modified context toolkit is a better choice for use in the framework proposed in this thesis.

3 The sentient agent programming model

We provide a programming model, based on the sentient object model and incorporating the STEAM event service,

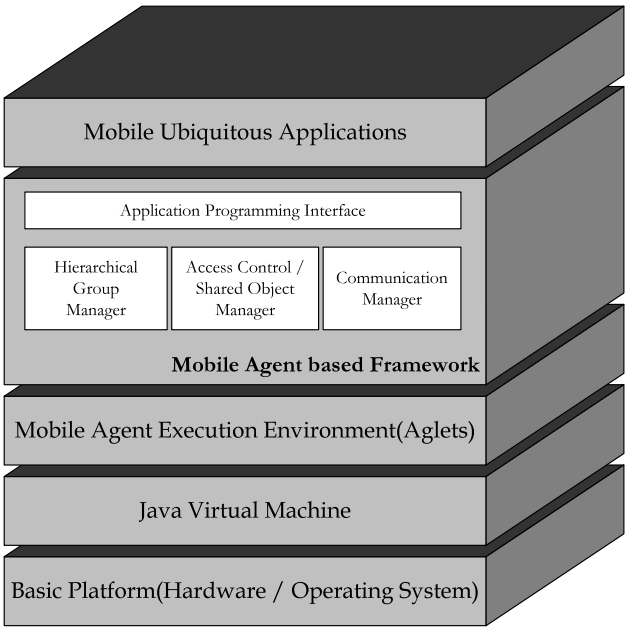
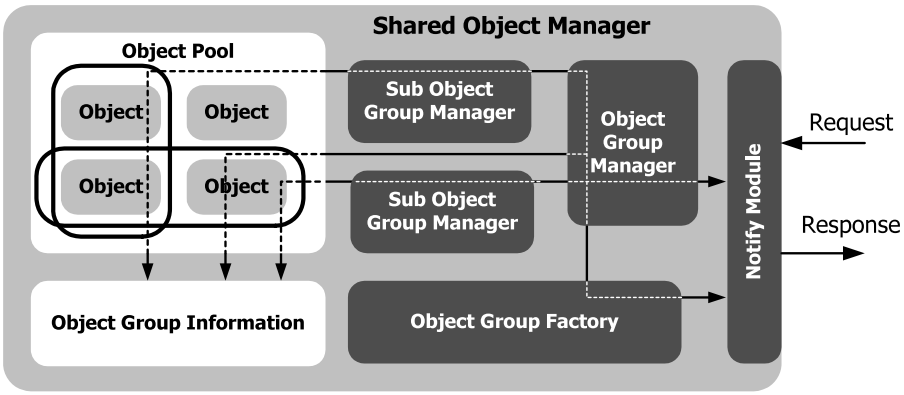


Fig. 1 Overview of Framework

Fig. 2 Shared object manager



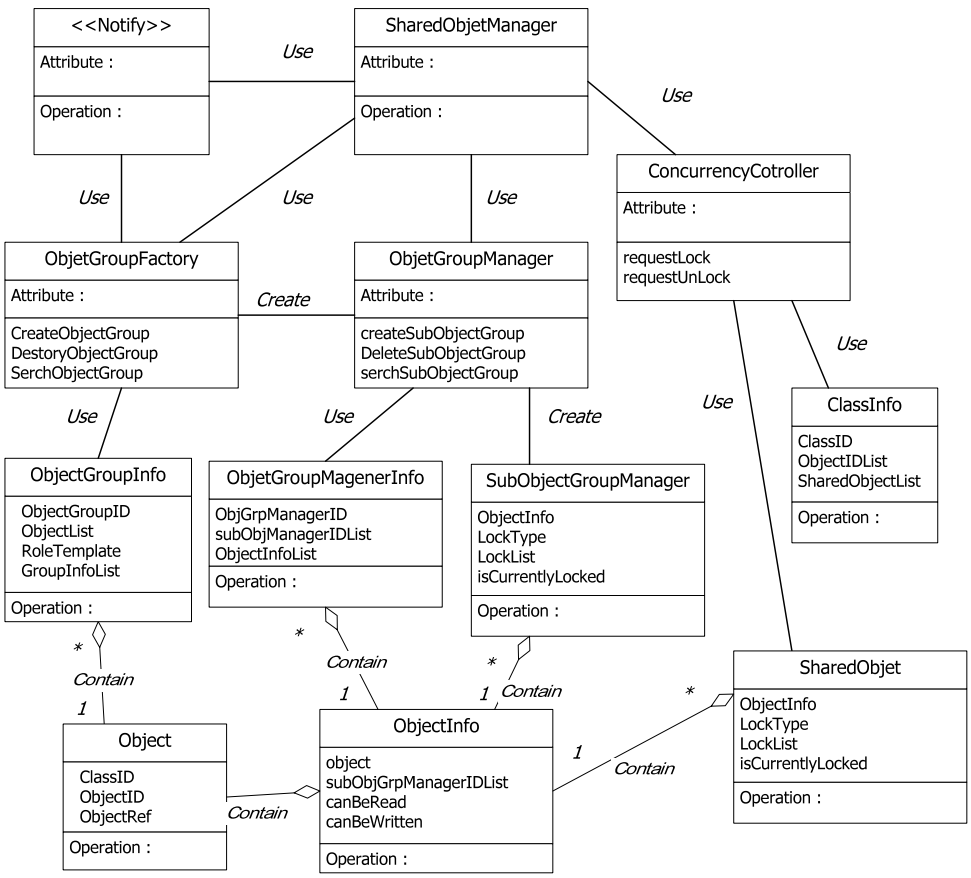


Fig. 3 Class diagram of Shared Object Manager

which provides abstractions for the development of mobile, context-aware applications.

3.1 Programming sensors and actuators

Sensors are developed as software abstractions that produce STEAM events, whilst actuators are developed as software abstractions that consume STEAM events. These software components encapsulate and act as wrappers for hardware and software sensors and provide mappings between specific sensor protocols and proprietary data formats, and STEAM events. Sensors provide a uniform interface to sensory information through STEAM events, and hide details of the underlying sensing technologies. Actuators extend from STEAM consumer entities that export an API for subscription management. Actuators consume STEAM events according to active subscriptions and filters, and transform the information contained within these events, to specific hardware or software commands. Actuators subscribe to event types of interest based on a set of event filters. In addition to the class file, each sensor and actuator also contains an XML description file, which contains information about the events produced or consumed, as well as probabilistic information regarding the uncertainty of each event.

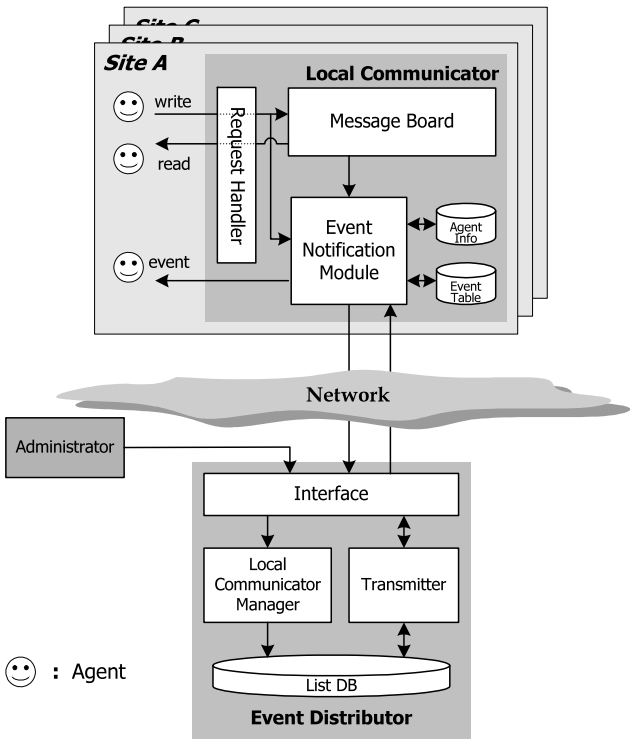


Fig. 4 Communication Manager

3.2 Programming sentient objects

Sentient objects are developed using a graphical development tool that allows developers to specify relevant sensors and actuators, define fusion networks, specify context hierarchies and production rules, without the need to write any code.

3.2.1 Specifying inputs and outputs

Specification of inputs to the object is done simply through 'drag and drop' from libraries which contain XML descriptions of sensors and sentient objects. Each descriptor describes the name and type of each STEAM event produced by the sensor or object. Outputs are specified in the same way, with descriptors available describing the events consumed by the actuator or object. Much of the programming effort is concentrated in the specification of the logic of the object.

3.2.2 Specifying contexts

Specification of the context hierarchy is the next step in the development of an object. The hierarchy is built up by specifying the attributes of each individual context. The most important attributes of a context are: (1) The set of events which are of interest in this context; (2) The set of rules which are active in the context; (3) The set of other contexts to which this context is related (i.e. child contexts) and may transition to; (4) The conditions under which transition to another active context occurs.

In effect, a context acts as a composite event filter whilst it is active, only those events which are defined as being of interest in that context are delivered to the object. A context defines the behaviours that are appropriate within the context by specifying the set of production rules that are active, and consequently evaluated, in each context. The fact that only a subset of sensor inputs and production rules needs to be considered for each individual context aids in reducing the complexity of developing context aware applications.

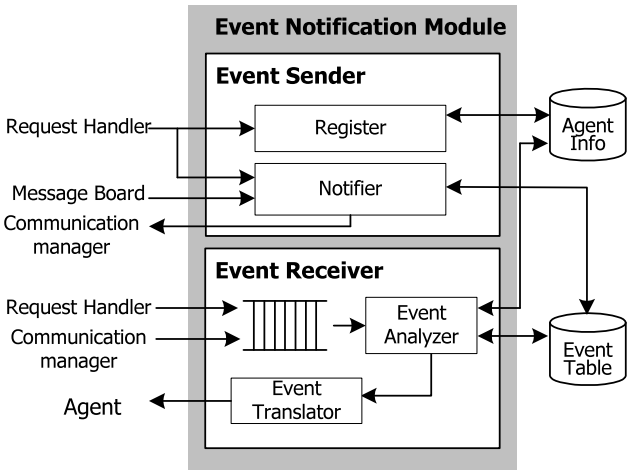


Fig. 5 Event Notification Module

3.3 Specifying fusion services

Fusion services within sentient objects are defined at the level of a context, using Bayesian networks. The tool allows the user to specify a Bayesian network for fusing fragments of context data on a per-context basis, via an intuitive graphical network builder.

The network is constructed by defining the events of interest and their relationships by adding nodes and arcs in an interactive manner. The probabilities of all root nodes are then specified and a conditional probability table (CPT) is

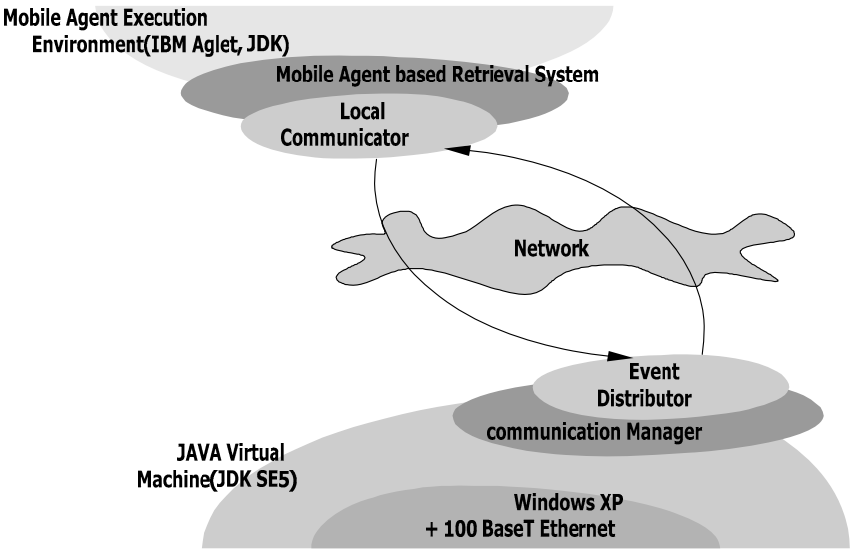
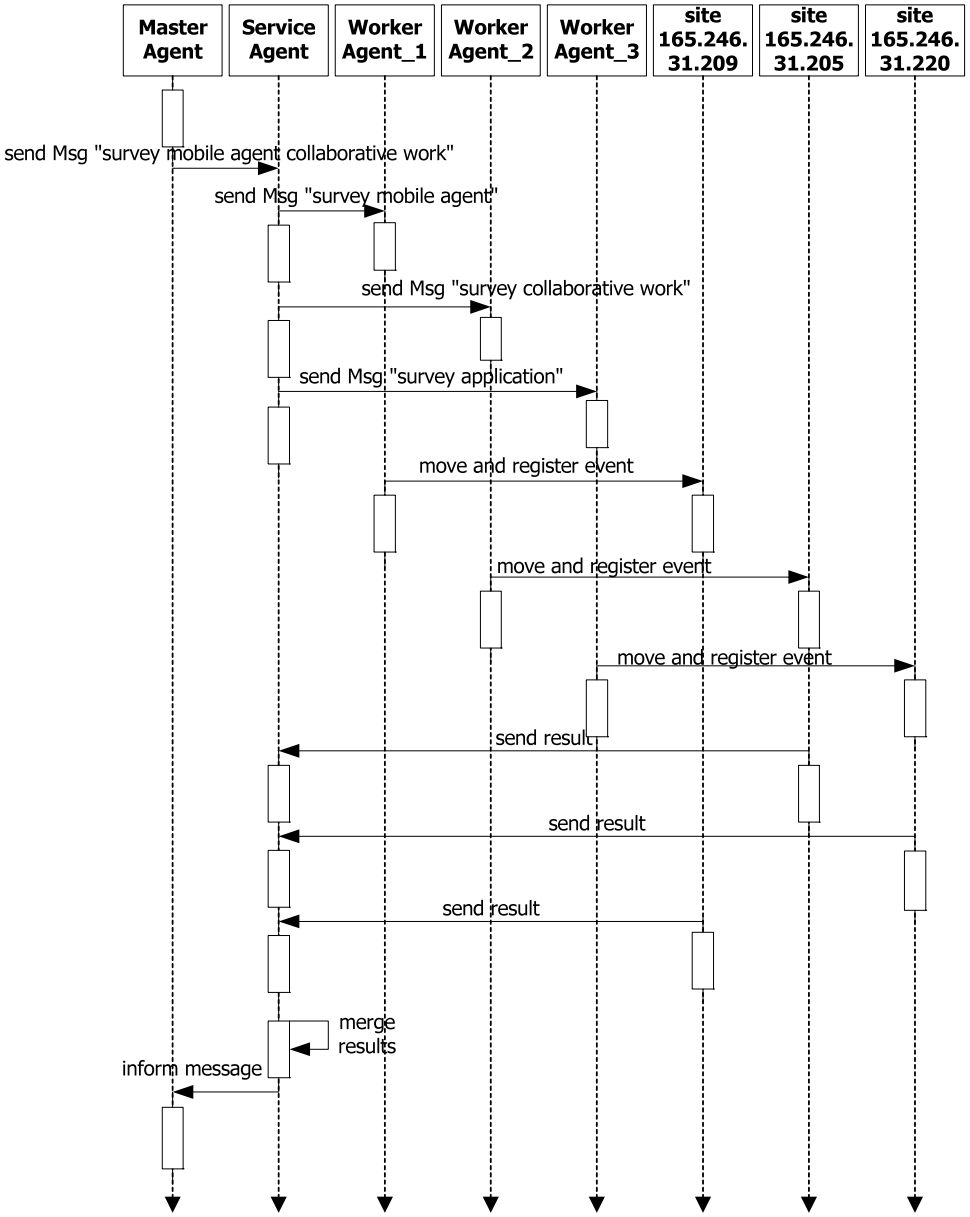


Fig. 6 Develop environment

Fig. 7 An experiment process



constructed for each non root node. These tables capture prior probabilities of events and rely on the a priori availability of probability data. This may be determined through experimental evidence as was done for ultrasonic sensors in the development of our sentient model car, where the probability distribution that an arbitrary sensor reading was within a specified threshold was calculated based on experimental observation.

3.4 Specifying rules

The complexity of specifying effective rules and knowledge bases is one of the greatest challenges in context-aware application development. Sentient objects make use of an embedded CLIPS inference engine, but CLIPS syntax is complex and not easily assimilated by the majority of developers. We attempt to make knowledge capture more accessible to domain experts by providing a high level, graphical rule builder, allowing the definition of application behaviors at the level of a context, and hiding the complexities of CLIPS syntax from the user.

4 Design of framework

In this section, we design a mobile agent based framework for mobile ubiquitous application. The framework is composed by hierarchy management for multi-layered mobile agent, coordination management and event notification.

4.1 Overview of framework

Mobile agent platform provides a lifecycle (Creation/Interpretation/Execution/ Destroy) of mobile agent. Additionally, it must provide access control to resource in system. We extend this general mobile agent platform to perform coordination among mobile ubiquitous applications. Figure 1 is a overview of framework in this paper.

We adopt IBM Aglet2.02 as common mobile agent platform that operates on Java Virtual Machine and add 3 components to build the framework. Aglet provides a lifecycle of mobile agent and event notification model based on socket.

4.2 Shared objects and access control

Mobile ubiquitous agent can access common objects with hierarchical role based coordination model and the common object and access control manager control access right and roles. This manager defines objects that will be used in ubiquitous applications as object group unit based on role relational template in group. Common object management ca provides ease of management of objects defined as group unit and acquisition of group information. All objects has class information and identifier of instance. Working agent group can register object group that will be used in working agent group and use the information of object. In addition, it can register and delete of objects dynamically according to necessary. Shared object manager is composed as Object Group Manager, Object Group Factory, Concurrent

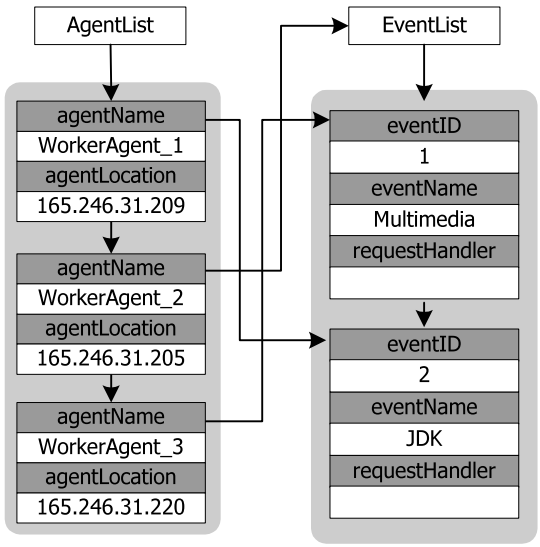
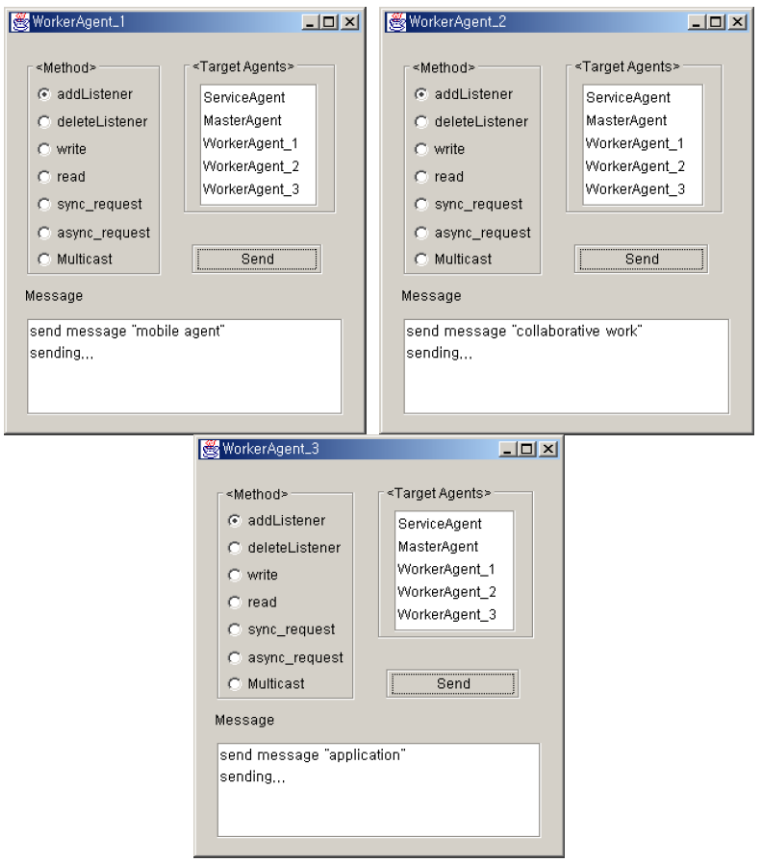


Fig. 8 Agent information before event registration

Fig. 9 Event registration



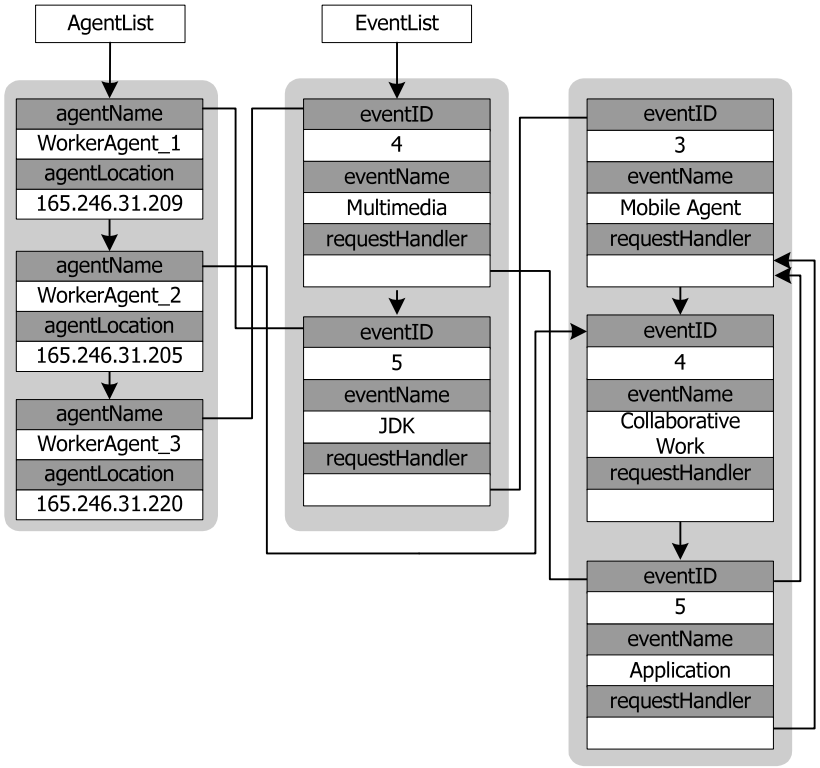


Fig. 10 Agent information after event registration

Controller and Notification Module. Figure 2 is a design of shared object manager.

Object Group Manager is charged of a lifecycle of object and has an interface for communication with Access Controller. Object Group Factory maintains all information of object group. Notify module send a notification to agent group whenever it detects changed state of common object. Figure 3 is a class diagram of Shared Object Manager.

4.3 Communication Manager

Communication Manager provides a communication channel among agents using multicasting method. This is composed of Local Communicator and Event Distributor. Local Communicator is charged of communication among mobile agents that are act a mission behavior. Event Distributor manages a communication between local communicator.

4.3.1 Local Communicator

Local Communicator manages a registration/deletion of events from agent. To notify and receive events to/from event distributor, event notification module and message board. Notification module support synchronized communication and Message board support asynchronous communication. Request Handler is an interface for agent to communicate with Local Communicator. If agent wants to communicate with other agent using asynchronous method, Message Board is used for that. This is similar with general indirected communication.

4.3.2 Event Notification Module

This is composed of Event Sender and Event Receiver. Event Sender contains Register for registration/deletion of event and Notifier to send event notifier events. And, event receiver generates event for proper agent. Figure 5 is Event Notification Module.

Event Sender is composed of Register for event registration and Notifier. Register is acted by request through request handler. This is, agents can receive interesting events not all events by registration of needed events. Registered events are stored event table.

5 Experiment

To validate of this framework, we develop a simple information searching system using this mobile agent based ubiquitous application. Figure 6 is a overview of the development environment for this system.

We checked below points to validate of this framework.

- Can agent register interesting events?
- Can framework provide a proper multicasting?
- Can framework provide a reliable asynchronous communication?

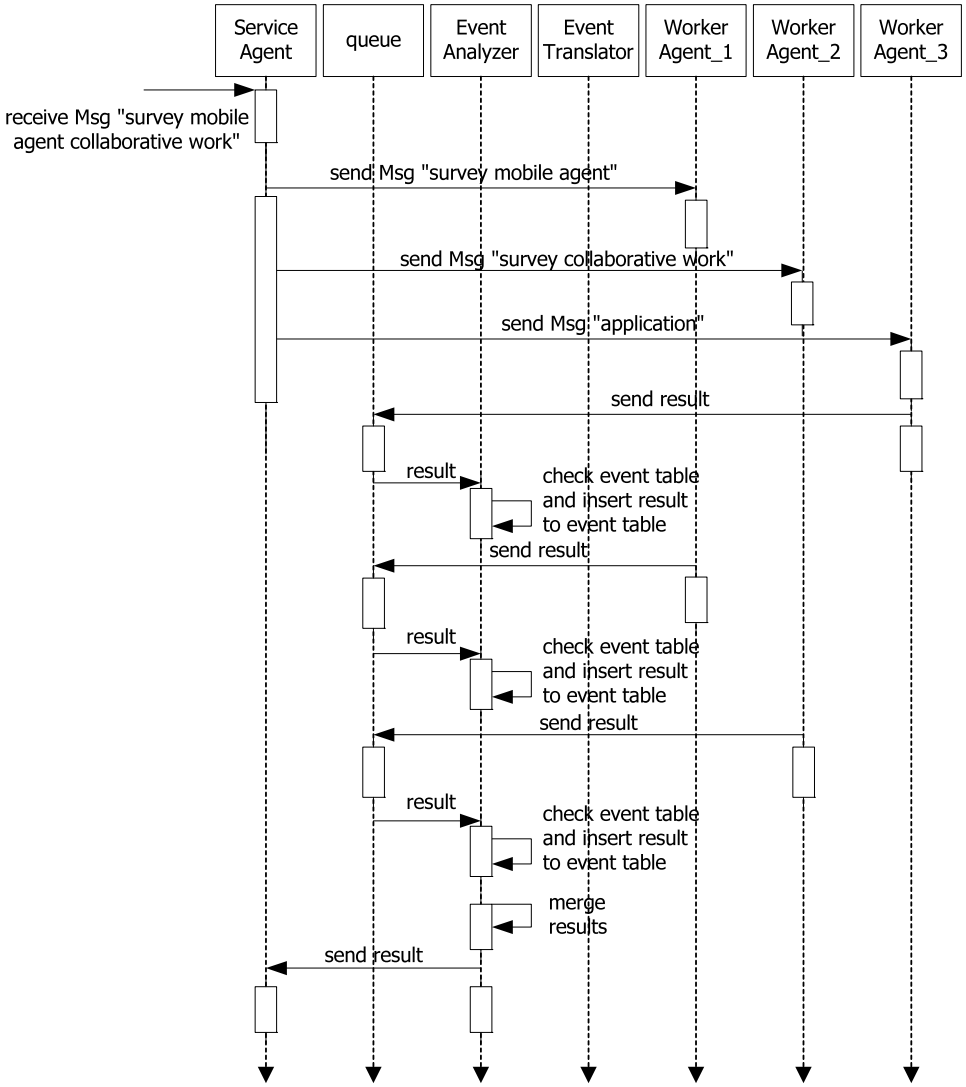


Fig. 11 Agent information after event registration

Figure 7 describes an experiment process for a prototype application.

This application is composed of five mobile applications (MasterAgent, ServiceAgent, WorkerAgent_1, WorkerAgent_2, WorkerAgent_3). MasterAgent want to get information from environment, so it send message to Service Agent to start a work. ServicAgent asks the WorkerAgent to gather context information. WorkerAgent_1 is moving to place 165.246.31.209, WorkerAgent_2 is moving to 165.246.31.205 and WorkerAgent_3 is moving to 165.246.31.220 for getting a context information. Each agent registers interesting events to be notified.

Figure 8 shows an agent information table before event registration.

Figure 9 shows that agents register events on site. After event registration, an agent information table is updated like Fig. 10. ServiceAgent asks multicat when it requests a proper work to WorkerAgent. Therefore, each result from WorkerAgent must be merged an object. This result is added event table when ServiceAgent send message. If all results are arrived, merged date is sent to ServiceAgent.

ServiceAgent requests a multicast process when it want to send a message to WorkerAgent. So, this framework needs to send a result as combined objects to ServiceAgent. When ServiceAgent sends a message, Event Notifier registers an information of ServiceAgent on event table, and add an result from each agent on table. If all results are registered on table, it makes an result object with all result and reply to ServiceAgent.

Figure 11 is a sequence diagram to show a process of sending/receiving of multicast messages from SerivceAgent. This process is done on event receiver module in local communicator.

In Fig. 11, ServiceAgent request a multicast process to WorkerAgnets and get a result as just an object. This re-

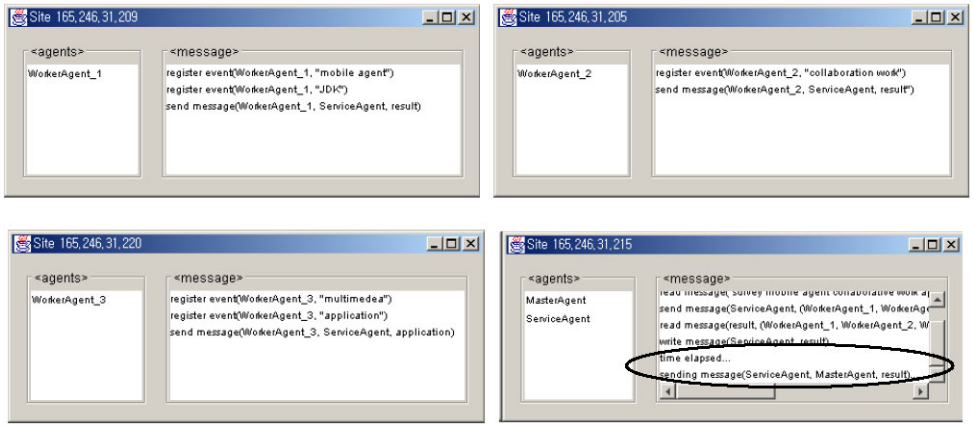


Fig. 12 Asynchronous communication with message board

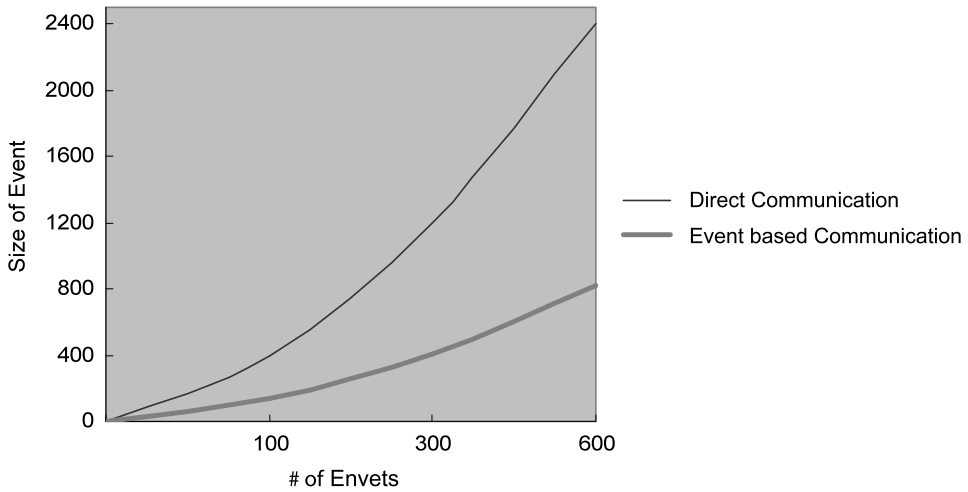


Fig. 13 Comparison of # of Event

sult object is transferred to MasterAgent from ServiceAgent. ServiceAgent also can provide an asynchronous communication methods with message board like Fig. 12.

Site 165.246.31.215 shows message is transferred to event notification muddle after an interval time automatically. In this experiment, we can check this framework can support a synchronous and asynchronous communication between multi agents.

6 Evaluation

In this section, we evaluate this framework can support a cooperation work among multiagents.

We can assume a size of total event

$$m \times k \times n$$

n : # of Agent / m : # of Site / k : # of Movement of Agent.

This means that size of transferring events increase by number of site, agent and movement of agent. But, this framework can reduce event size like below.

$$m \times (m - 1)$$

This shows our system can reduce event size that is defined only by number of site. That is number of agent and movement of agent do not affect a size of event. So, our framework can support developer can implement a large scaled application without concerning of event size because they can neglect number of agent and movement.

And, this framework can support agent can register event on site that it want to get. So, number of transferred event will be reduced that agent need check and proceed.

Assume n is # of agent, k is # of event. # of event will be like below.

$$n \times k$$

And, if agent can register event that agent want to get, # of event will be like below.

$$n \times e(e \leq k)$$

In our experiment, # of transferred event will be reduced almost 34% of previous system. Figure 13 shows event based communication can reduce a frequency of event.

In this experiment, we can know this framework can reduce a transferred event and provide an efficient framework

to developers that they can implement a large scale application without concerning of size of agent and movement.

7 Conclusion

In this paper, we have presented a framework, based on the sentient object model, for developing context-aware applications in a mobile, ad-hoc environment. Our framework provides a systematic approach to context-aware application development, including the ability to fuse context fragments and deal with uncertainty in a probabilistic manner, the ability to represent context within the application, and the ability to easily compose rules to reason efficiently about context. This functionality is offered in a tool that is easily accessible to a wide range of developers, permitting the rapid design and development of applications, based on sentient objects, for ubiquitous computing environments.

We have successfully applied our framework to the development of a number of proof-of-concept applications, including a simple sentient model car application that drives and obeys traffic signals autonomously.

References

1. Strang, T. (2003). *Service interoperability in ubiquitous computing environments*. Ph.D. Thesis, Ludwig-Maximilians University.
2. Szumel, L., LeBrun, J., & Ownes, J. D. (2005). Towards a mobile agent framework for sensor networks. In *Proceedings of the second IEEE workshop on embedded networked sensors* (pp. 79–88).
3. López de Ipiña, D., Mendonça, P. R. S., & Hopper, A. (2002). *Personal and Ubiquitous Computing*, *6*(3), 206–219.
4. Forman, G. H., & Zahorjan, J. (1994). The challenges of mobile computing. *IEEE Computer*, *27*(6).
5. Dey, A.K., & Sohn, T. (2003). Supporting end user programming of context-aware applications. In *Conference on Human Factors in Computing Systems (CHI) Workshop on Perspectives in End User Development*, Fort Lauderdale, FL, April, 5-10, 2003
6. Park, N., Lee, K., & Kim, H. (2005). A middleware for supporting context-aware services in mobile and ubiquitous environment. In *Proceedings of the international conference on mobile business* (pp. 694–697).
7. Lee, S., Lim, K., & Lee, J. (2005). The design of webservices framework support ontology based dynamic service composition. In *LNCS: Vol. 3689. Proceedings of the second Asia information retrieval symposium* (pp. 721–726). Berlin: Springer.
8. Yan, L., & Sere, K. (2004). A formalism for context-aware mobile computing. In *Proceedings of the third international workshop on parallel and distributed computing, third international symposium on/algorithms, models and tools for parallel computing on heterogeneous networks* (pp. 14–21).
9. Lee, S. (2007). Context modeling and inference system for heterogeneous context aware service. In *Lecture notes in computer science: Vol. 4558* (pp. 413–422). Berlin: Springer.

Seungkeun Lee the Computer Science & Engineering degree and the M.Sc. and Ph.D. degrees from Inha University, South Korea in 1996, 1998 and 2006 respectively.
He was a PostDoc researcher in INRIA, France from 2006 and 2007.
Since 2008 he has been at Joah Cooperation AG, Switzerland, where he is a managing director and CIO.
Hie research interests include ubiquitous computing environment, context awareness and home network system.

Kuinam Kim received his B.S. degree of Mathematics, University of Kansas in 1989. He received his M.S. degree of Statistics and Ph.D. degree of Industrial Engineering from Colorado State University. He is currently a professor in the Industrial Security Department, Kyonggi University, Korea. His research interests include industrial security.