Implementation of an Adaptive Total Ordering Protocol

Yoshihiro Yasutake[†], Shinobu Izumi[††], Kentaro Oda[††] and Takaichi Yoshida[††],

[†]Department of Intelligent Informatics, Kyushu Sangyo University, 2-3-1 Matsukadai, Higashi-ku, Fukuoka, Japan
[††]Program of Creation Informatics, Kyushu Institute of Technology, 680-4 Kawatsu, Iizuka, Japan

Summary
In distributed systems, the communication among group members often needs ordered messages to guarantee that every member receives the messages in the same order. Oftentimes, changes in distributed computing environment occur and this undermines the assumption of any ordering algorithm. Therefore, it is difficult to presume a suitable ordering algorithm and apply a single algorithm throughout the lifetime of the system.
In this paper, we present an adaptive total ordering protocol and its implementation on our reconfigurable object model. Our adaptive protocol selects a suitable ordering protocol from optimistic and pessimistic total ordering protocols dynamically depending on the runtime environment. With optimistic protocol, it is possible to reduce the affect of the network latency which is not negligible in the pessimistic protocol, because the optimistic protocol delivers messages immediately after receiving them. The optimistic protocol though incurs high ordering cost during rollback, and so in this case, it is worth using the pessimistic algorithm instead.
The adaptive protocol is realized on the reconfigurable object model. This object model enables dynamic changing of object behaviors by reconfiguring consists of meta-objects. The total ordering processes are dealt in one of the meta-objects, called re-ordering meta-object. Therefore reconfiguration makes it possible to switch a total ordering protocol to others dynamically depending on the runtime environment.
We introduce the reconfigurable object model and the method to realize the total ordering protocol in this model. To explain the adaptive ordering protocol, we show the implementation including detecting environmental change and reconfiguring object.
We looked into the feasibility of our adaptive total ordering protocol in this paper. We present the requirements to realize our adaptive protocol, and show how the adaptive protocol is implemented on the reconfigurable object model.

Key words:
Total ordering protocol, Adaptive computing, Reconfigurable object model

1. Introduction

In the distributed environment, message multicast serves as a foundation for the communication among group members like replica group and event notification. The group communication often needs the guarantee that each member receives the messages in the same order. For example, if group members have replicated data, it is always expected for them to process messages in the same order to make their states consistent among them.

The total ordering algorithms have been proposed for various systems [1]. They have their own characteristics. Therefore it is expected to apply a suitable algorithm, so that the system performs stable operation. However, their performance always changes according to the characteristics of applications and runtime environments. For example, different applications have different event granularity and message population. The environment has the variation of network traffic, network latency and active users. These make it difficult to presume an ordering algorithm suited for an environment and apply a single algorithm throughout the lifetime of a system.

We propose the adaptive total ordering protocol. Our approach is selecting a suitable total ordering protocol dynamically. In general, the protocol has its own ordering cost depending on the application and environment. Unlike using single protocol, the adaptive protocol can improves the ordering cost by using multiple protocols dynamically.

We use our reconfigurable object model proposed in [2][3] to implement the adaptive total ordering protocol. In this model, the total ordering protocol is implemented as a module. The module is isolated and has weak connections with the other. Therefore the total ordering protocol is isolated from the application behavior. Additionally, the modules can be exchanged with no consideration of others. Thus the adaptive protocol is realized by dynamic exchanging of modules implementing the total ordering protocol.

We present the adaptive total ordering protocol in section 2 and the reconfigurable object model in section 3. In section 4 and 5, we describe the implementation of proposed protocol based on our model. Its discussion is section 6 and the conclusion is section 7.

Manuscript received July 5, 2007
Manuscript revised July 25 2007

2. Adaptive Total Ordering Protocol

The environmental change is not negligible in the distributed system, because it often affects the performance of whole system even if it occurs partially. The performance of total ordering protocols is also influenced by the runtime environment. Therefore it is worth adapting the protocol to the environment dynamically.

We propose switching total ordering protocols dynamically depending on the environment. For the first step, we classify the total ordering protocols into optimistic and pessimistic approaches in the point of the time when the messages are marked with deliverable.

The optimistic algorithm [4] generally delivers the messages immediately after receiving them. If members receive messages in the same order, there is no conflict in the receiving order among members. In the such case that the order of receiving messages matches the ideal one with no conflict, the optimistic approach can be considered as an effective method. However, when the order of the messages is detected to be conflicted, the delivered messages must be undone and delivered again in the correct order. In the case that the conflict often occurs, the cost to cancel sending messages and undo delivered messages is not negligible in general. Thus the messages are delivered at the small cost, and the rollback generally causes expensive computation.

In the pessimistic approach [5], the messages are preserved until their delivered order would be valid. Thus the pessimistic algorithm guarantees the message order without rollback. Though there is no conflict, the cost of ordering is more expensive than the optimistic approach, because this approach needs to exchange messages in order to determine the next receiving message. In addition, the member has to wait the confirmation messages from the other members to decide the next delivered message, and thus the elapsed time waiting them has to be considered. Therefore the ordering cost is relatively expensive and the failure and latency at a single point affects all group members.

The group members exist on each network location and then their messages are affected from the logical distance and network condition between the communicating members. If the network latency becomes larger, the message order received by each member would be more mixed up. The frequency of the message multicast, which depends on the application and runtime environment, also causes the conflict between multicast messages. In the case that the message order conflicts less frequently, the optimistic ordering algorithm would be better than the pessimistic one in general. In another case that has the high possibility for the message conflict, the pessimistic algorithm would be better to keep stable performance, because it confirms each message order before delivering them. We propose to switch an appropriate ordering algorithm implemented in an optimistic or a pessimistic protocol according to dynamic environment changes.

We give the case, which the runtime environment changes for a certain period, as an example. Fig. 1 shows ordering cost of optimistic protocol and pessimistic protocol. In this case, most messages are received at the same order except for the period between time A and B. It can be consider that the message order often conflict as a result of the environmental changes between time A and B. Thus the pessimistic protocol provides the stable performance and the optimistic protocol is sensitive to the execution conditions. Since the pessimistic protocol takes the higher ordering cost than the optimistic protocol except the period between time A and B, the pessimistic protocol is proper in the certain period and the optimistic protocol is better in the other. We consider that it is ideal to switch from the optimistic protocol to the pessimistic protocol at time A and switches again to the optimistic protocol at time B. To select a suitable protocol adaptively enables to reduce the ordering cost in the comparison with the case using single protocol.

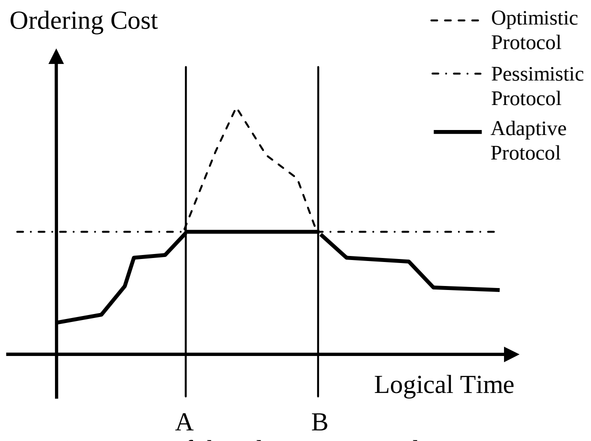

Fig. 1 Cost of the adaptive protocol.

The overview of our adaptive protocol consists of the following steps.

1. Cost Estimation: Members of a group receive messages and deliver them according to an ordering algorithm. And each member calculates the costs in target algorithms (the current algorithm and the other algorithms).

2. Comparison of ordering costs: Each member compares the calculated costs, respectively.

3. Agreement: A most suitable algorithm for the current environment is selected as a result of an agreement among members.

4. Preparing for dynamic switching: The member manages the pending messages to keep the message order in a next total ordering protocol.

5. Switching to a next protocol: The members resume pending and delivering messages with the new ordering algorithm.

We described the effectiveness of our adaptive protocol in [6]. In this paper, we describe the implementation of the protocol.

3. Reconfigurable Object Model

We proposed to adapt objects to the environmental changes by changing their behavior [2][3]. There are several environmental changes: hardware resource (CPU load, available memory and so on), network condition (topology and latency), software evolution and user preferences. For adapting to these changes, we enable objects to reconfigure their meta-composition as appropriate. The reconfigurable object consists of meta-objects (Fig. 2). The reconfigurable object has the same interface as the normal object by encapsulating its meta-objects.

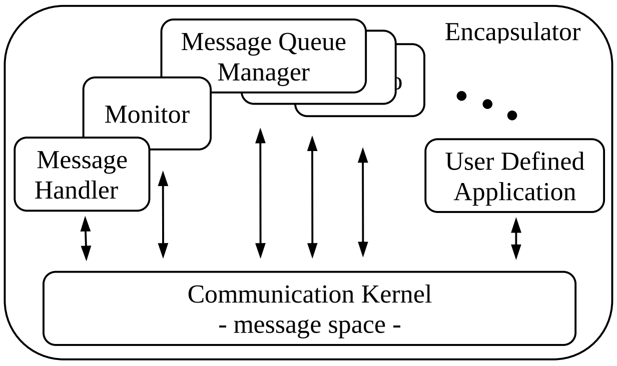

Fig. 2 The reconfigurable object model.

The meta-object is the functional component of object. The objects having different meta-object have different function. If it is allowed to exchange meta-object, it is possible to change object behavior. The reconfigurable object provides the way to exchange meta-objects to adapt the object itself. For the dynamic and reliable exchange, the communication among meta-objects is loosely-coupled in this model.

The meta-objects communicate through the message space. This model is the tuple space introduced in the Linda distributed programming language [7]. We use the term `message' instead of tuple, and use `message space' instead of tuple space as analogous to object orientation. The message space provides indirect, asynchronous and content-addressing communication. The meta-objects use the operations against the message space; `out()` puts a message to the message space, `in()` withdraws a message, and `read()` reads a message without withdrawing. To put the specified label on the messages, the other meta-objects can get necessary messages by identifying the label.

The configurable object consists of the meta-objects: message handler controls end-to-end communication between remote objects, message queue manager controls the order of messages, executor controls concurrent executions, monitor collects internal activities, adaptation manager determine the adaptation strategy. These meta-objects communicate indirectly by withdrawing and putting messages through the message space. For reconfiguration, they can be replaced. For example, when the different remote communication protocol is required, the current message hander is removed, and then the new message hander is added.

4. Implementation

Fig. 3 shows the overview of our adaptive protocol. The messages are received in message handler. Then the received messages are processed by the total ordering protocol and delivered to the application. This flow is monitored and reported to the adaptation strategy manager. This manager decides which protocol is efficient in current condition on the monitored result and user preferences.

The total ordering functions, which are required in each group member, can be implemented in the re-ordering meta-object. The re-ordering object can be considered a kind of message queue manager. The re-ordering meta-object withdraws received messages from the message space before applying other message processing, and executes total ordering processes: putting time stamp on messages, queueing and sorting messages.

In our adaptive protocol, the total ordering protocol is changed dynamically. This protocol change is realized by exchanging a re-ordering meta-object. Each meta-object implements its own protocol. In other words, there is the same kind of meta-objects as the total ordering protocols. When the object needs to change a current total ordering protocol, the re-ordering meta-object is exchanged for the other meta-object implementing a different protocol.

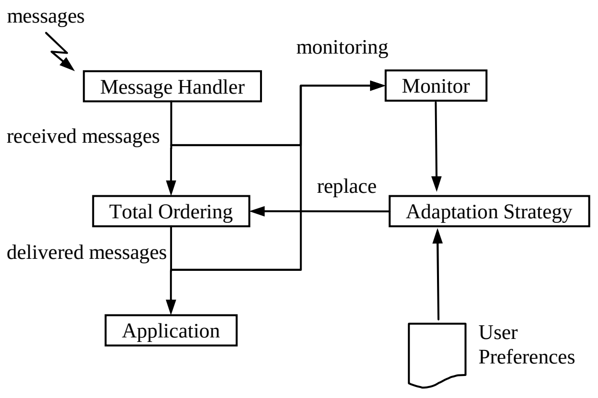

Fig. 3 The overview of our adaptive protocol.

The following is the explanation of each function in our adaptive protocol (Fig. 3).

Message Handler

The message handler is the communication interface of the reconfigurable object. The task is basically the control of message sending and receiving. There are various types of message handler according to end-to-end communication protocol: communication with remote objects, encrypted communication, asynchronous communication and so on.

Total Ordering

The function for total ordering is the main subject in this paper. We explain how the total ordering protocol works in the reconfigurable object, and replaced dynamically. For the implementation of total ordering protocol, it is possible to add the re-ordering meta-object composed of message queues (explained in Section 3 as message queue manager). After getting a delivered message, the application treats it and puts a result message to the message space.

The switching protocol is realized by the replacing the re-ordering meta-object. It is simple to switch the total ordering protocol in the reconfigurable object model. The new object has the same operations (used in the generative communication) and different protocol (used in the total ordering). The other message handling processes like the receiving and executing messages can continue their processes while the re-ordering meta-object is being replaced, because they are independent of the other meta-objects in the generative communication. This aspect contributes to switch the total ordering protocol simply and effectively.

There are the other meta-objects communicating with the re-ordering meta-object through the message space. The application meta-object treats the messages which have been processed by the re-ordering meta-object. The monitor meta-object observes the runtime environment, and reports to the adaptation strategy. The adaptation strategy meta-object decides and controls the adaptive behavior depending on the information reported from the monitor.

Application

The application object has only behavioral function of user defined application. In this object, there are no functions as like as the total ordering processes, because the messages have already been ordered at the re-ordering meta-object. Therefore, it is not necessary for the application programmers to consider the total ordering processes in their program.

Monitor

The monitor gathers the internal states and activities of the reconfigurable object. In the message space, there are labeled messages waiting for begin pulled by a meta-object. The monitor object can statistically know the rate of received and delivered massages by observing periodically the messages in the message space. The monitor reports the observational result to the other meta-objects through the message space.

Adaptation Strategy

The adaptation strategy maintains the composition of meta-objects. In this object, the method for adapting environment is decided and started reorganizing of composition (reconfiguration). The adaptation strategy meta-object obtains the information created by the monitor through the message space. Then it is decided whether it starts a configuration or not by obtaining the consensus among group members.

The adaptation strategy meta-object is independent of the other meta-objects (The same can be said for other meta-objects). Therefore, the agreement process is executed concurrently with the application process. The details of adaptation protocol are described in section 5.

User Preferences

The application programmers need to care only functional behavior of their application, because the composition of reconfigurable object is hidden from the programmer. The way to access the meta-function is setting given properties by the reconfigurable object. The property is written in a file, and loaded by the adaptation strategy meta-object at initialization time.

For the adaptive total ordering protocol the following properties are provided and mainly loaded by the adaptation manager.

- Target protocols: The total ordering protocol is provided as one of the modules in runtime library of reconfigurable object. The protocols selected by the programmer are loaded in the adaptation manager at proper time.

- Trigger condition of switching protocol: Basically the low cost protocol is dynamically selected in our model. In addition, if the programmers want to reflect time and number of users to their system, they can describe settings including these attributes.

5. Adaptive Protocol

We consider it is feasible that the total ordering algorithms have no central entity like the sequencer and token on the distributed environment. Therefore, our adaptive protocol is designed to contain the total ordering algorithms not having a specific ordering entity. These algorithms are classified into communication history and destination agreement algorithm in [1]. The message sender is simple and the receiver respectively has the process for the total ordering. In our adaptive protocol, each receiver's ordering process is implemented in the re-ordering meta-object in each object.

There are the several adaptive protocols which support multiple total ordering protocols [8][9]. The characteristic of our adaptive protocol is to support the optimistic protocols as well as the pessimistic protocols. As a result of optimistic delivering, the group member needs the rollback and restarting processes when the message order differs from the others. It makes difficult to estimate the ordering cost, because the message order is not fixed at the deliver time. Our protocol estimates the ordering cost from the received and delivered messages passed among meta-objects. The estimation is achieved only by the reflective function provided by the reconfigurable object.

This section provides more detail explanation of our adaptive protocol described as the overview in section 2. The group member observes the runtime environment and decides the protocol change is necessary or not. If necessary, all members in the group start the reconfiguration. On the total ordering protocol, the runtime environment is the processing states of messages, and the reconfiguration is the switching the re-ordering meta-object.

Cost Estimation

The monitor meta-object estimates the cost of total ordering protocols from three conditions of message handling. One is the rate of receiving message. This rate implies how many messages are waiting for being handled in a certain time. Another is the processing time of each message. The message throughput is estimated from the average of processing time. The other is the message log with the timestamp. This make possible know the receiving order with each message timestamp.

The monitor meta-object collects the internal states and activities of the reconfigurable object. In the message space, there are labeled messages waiting for begin pulled by a meta-object. The monitor meta-object can statistically know the rate of received and delivered massages by observing periodically the messages in the message space. The read operation, described in section 3, is utilized to observe the messages. It means there is no interrupt in the message processing.

Comparison of ordering costs

For selecting a suitable total ordering protocol, the monitor meta-object in each object compares the target protocols with their ordering costs. We define the condition (1): ordering and switching costs. Let Cost(CUR) be the cost of the current protocol, Cost(NEXT) be the cost of the another protocol and Cost (CUR_S) be the switching cost from the current protocol.

$$Cost(CUR) > Cost(NEXT) + Cost(CUR\_S) . \quad (1)$$

This condition is used in the monitor, and the result is reported to the adaptation strategy meta-object. If the condition (1) is satisfied, it can be considered that the switching is effective at the current environment.

Agreement

The adaptation strategy meta-object receives the information from the monitor meta-object and starts the agreement process among group members. The adaptation strategy meta-objects in a group have respective monitoring results. They communicate to decide a suitable protocol on the current runtime environment which they exist.

The decision is based on the consent of the majority. The adaptation strategy meta-objects in each member vote a suitable protocol on its environment, and obtain a consensus. When the adaptation strategy meta-object decides to start reconfiguration, it sends a request message, which requests replacing with a new re-ordering meta-object.

When the group members decide a next protocol, they also decide a global time T when they switch to the next protocol. T is set at sufficiently bigger time than the local time in the group members to avoid the rollback for switching.

Preparing for dynamic switching

For the preparation to switch from a current total ordering protocol, the adaptation strategy meta-object requires the re-ordering meta-object to prepare being replaced. The condition (2) is required to separate the messages applied different protocols. t_m and t_n are the time stamps of message m and n. A current total ordering protocol apply the message m and a next protocol apply the message n. T is the global time of switching.

$$t_m \le T < t_n \,. \tag{2}$$

Switching to a next protocol

When the preparation is done, the re-ordering meta-object sends an acknowledgement to the adaptation strategy. Then the adaptation strategy removes the current re-ordering meta-object and adds the new meta-object implementing a total ordering protocol agreed among group members.

The meta-objects can be replaced by the self-contained manner. No other objects need to manage them for replacing themselves. To exchange the re-ordering meta-object, the time stamp and message queues are transferred through the message space.

1. The re-ordering meta-object packs and sends the current states for the message ordering protocols to the message space, such as local time and message queues.

2. The adaptation strategy removes the current re-ordering meta-object and adds the new meta-object. It is not required to synchronize the other meta-objects, because there are weak connections between them.

3. The new re-ordering meta-object withdraws the previous states from the message space, and initiates with them.

The message order is kept before and after switching, and the total ordering protocol is exchanged during execution. Moreover, the message handling processes, such as receiving message in the protocol hander and processing in the application meta-object, keep their process while the ordering protocol is suspended. Therefore, the entire system does not halt during the reconfiguration.

6. Discussion

The multiple message handlings are mostly implemented on the filter structure [10]. Our implementation requires handling messages: receiving, sending, observing messages, total ordering, and processing requests. However, they are not necessarily applied in sequence. For example, the monitor observes received and delivered messages, but the observation result has no relation with the message handlings. The message space, called tuple space in Linda, provides the weak relation between modules. It helps to organize complex process modules on the proper structure, and each module can behave effectively. In addition, it allows each module to have the same interface which is used to communicate with the message space. Thus it is possible to replace a module without the consideration of other modules.

Our adaptive total ordering protocol includes the optimistic protocols as well as the pessimistic protocols. One of the features of optimistic protocols is doing rollback when the message sequence turns into a different with the other message receivers. The reconfigurable object model also supports the rollback. The basic concept is the same as replacing meta-object.

1. The application object puts its own states including local time into the message space. The states are packed in a message and sent to the message space.

2. At rollback time, the application object withdraws the state message having the target time stamp from the message space.

Many state messages for rollback will be put into the message space. However, there is a method to clean up them. The method is preparing GVT (Global Virtual Time [6]) manager. GVT gives the assurance that all receivers have a bigger local timestamp than a certain time (GVT). Therefore the GVT managers in the group members communicate to calculate GVT and the each manager deletes the state messages respectively based on the knowledge.

7. Conclusion

We described the necessity of adaptive total ordering protocol in the distributed system. Our approach was selecting the total ordering protocols according to the environmental changes. The dynamic protocol switching was smoothly implemented in the reconfigurable object model, because this model was designed to adapt object itself to the environment. The total ordering protocol was implemented in a meta-object which could be replaced by the self-contained manner. And the other required functions, observing messages and decision an appropriate total ordering protocol, were also realized in the reconfigurable object model. We presented the requirements and implementation of our adaptive protocol, and verified the possibility of the protocol. Moreover, we were convinced that the reconfigurable object model is one of the most suitable object models to implement the adaptive system that described in this paper.

References

[1] X. Defago, A. Schiper, and P. Urban, Total order broadcast and multicast algorithms: Taxonomy and survey, ACM Computing Surveys, Vol.36, No.4, pp.372-421 (2004).
[2] K. Oda, H. Najima, Y. Yasutake, T. Yoshida, A Simple, Safe Reconfigurable Object Model with Loosely-Coupled Communication, Proceedings of the IEEE 20th International Conference on Advanced Information Networking and Applications (AINA 2006), 2006, pp.406-41.
[3] K. Oda, S. Izumi, Y. Yasutake, T. Yoshida, A Simple Reconfigurable Object Model for a Ubiquitous Computing Environment, International Journal of Computer Science and Network Security (IJCSNS), Vol. 7 No. 5, pp. 8-16, May, 2007.
[4] D. R. Jefferson, Virtual time, ACM Transactions on Programming Languages and Systems, Vol.7, No.3, pp.404-425 (1985).
[5] K. P. Birman and T. A. Joseph, Reliable communication in the presence of the failures, ACM Transactions on Computer Systems, Vol.5, No.1, pp. 47-76 (1987).
[6] Y. Yasutake, T. Kadowaki and T. Yoshida. Adaptable Ordering Protocols in Distributed Computation. Proceedings of the Inter-national Symposium on Communications and Information Technology, 2004, Vol.2, pp.1126-1131.
[7] D. Gelernter, Generative Communication in Linda, ACM Trans. Prog. Lang. Syst., 1985, Vol. 7, No. 1, pp. 80-112.
[8] P.D. Ezhilchelvan, R.A. Macedo and S.K. Shrivastava, Newtop: a fault-tolerant group communication protocol, Proceedings of 15th IEEE International Conference on Distributed Computing Systems, pp.296-306 (1995).
[9] L. Rodrigues, H. Fonseca and P. Verissimo, Totally ordered multicast in large-scale systems, Proceedings of the 16th International Conference on Distributed Computing Systems, pp.503-510 (1996).
[10] L. Bergmans and M. Aksit, Composing Crosscutting Concerns Using Composition Filters, Comm. ACM, Vol.44, No.10, pp.51-57 (2001).

Yoshihiro Yasutake received the B.S. and M.S. from the Department of Artificial Intelligence, Kyushu Institute of Technology, Japan, in 2000, 2002 respectively. He has been an assistant professor of the Department of Intelligent Informatics at Kyushu Sangyo University since 2005. His current research interests include reliable distributed systems, adaptive middleware architecture. He is a member of the IPSJ (Information Processing Society of Japan).

Shinobu Izumi received the B.S. and M.S. from the Department of Artificial Intelligence, Kyushu Institute of Technology, Japan, in 2004, 2006 respectively. Since April 2006, he has been a PhD student at Kyushu Institute of Technology. His current research interests include disabled access GIS, peer to peer networking and distributed system.

Kentaro Oda received the B.S. and M.S. from the Department of Artificial Intelligence, Kyushu Institute of Technology, Japan, in 1999, 2001 respectively. He has been an assistant professor of the Program of Creation Informatics at Kyushu Institute of Technology since 2004. His current research interests include adaptive middleware architecture, multi-agent systems (robotics soccer RoboCup), and distributed systems. He is a member of the ACM, IEEE (IEEE Computer Society).

Takaichi Yoshida received the B.S. degree in electrical engineering form Keio University, Japan, in 1982, and the M.S. and Ph.D degree in computer science form Keio University in 1984 and 1987, respectively.
Since 1987, he has been at Kyushu Institute of Technology, and currently is a professor in the Graduate School of Computer Science and Systems Engineering, Kyushu Institute of Technology. His research interests include distributed computing and object-oriented computing. He is a member of the ACM, IEEE (IEEE Computer Society).