Dynamic Reconfiguration of Network Applications and Middleware Systems in the Bio-Networking Architecture

Junichi Suzuki, Tadashi Nakano, Keita Fujii, Nobuyuki Ikeda and Tatsuya Suda

Abstract— **The ability to reconfigure network applications and distributed systems (e.g. middleware) to adapt to changing network conditions is becoming more important. This paper describes our research efforts regarding dynamic reconfiguration of network applications and middleware. We first describe our approach to reconfigure network applications through a biologically-inspired evolutionary process. The current research status and obtained results are presented in terms of simulation work and empirical implementation work. Then, we describe our extended effort to reconfigure middleware as well as network applications. We also present the current status of this effort by listing the research issues that we have been taking into account for designing our mechanisms to reconfigure middleware.**

Index Terms—**adaptive middleware, evolutionary computing, reconfigurable middleware, reconfigurable network applications.**

I. INTRODUCTION

A key observation to the current and near future communication systems is that dynamic reconfigurability to adapt to changing network conditions is a unifying theme on which network applications and distributed systems (e.g. middleware) can be constructed [1]. We believe that the future networks, which will be orders of magnitude more complex and larger than current networks, should exhibit self-organization with inherent support for scalability and adaptability to environmental changes in networks. In order to make this research vision a reality, we have been investigating the dynamic reconfigurability in terms of the following two different research approaches:

- Network-aware applications that autonomously reconfigure their behaviors to adapt to dynamic network conditions (e.g. network load)
- Reconfigurable middleware systems that reconfigure their internal components to adapt to resource availability (e.g. the amount of available memory and bandwidth, and the kind of available communication/routing protocols)

We have been designing and implementing the above two approaches in the Bio-Networking Architecture project [2, 3]. The Bio-Networking Architecture is motivated by the observation that the above desirable properties (such as scalability and adaptability) have already been realized in various large-scale biological systems, and it applies key biological principles and mechanisms for designing network applications. The Bio-Networking Architecture is a new framework for developing large-scale, highly distributed, heterogeneous and dynamic network applications [3].

The remaining sections are organized as follows: Section II describes our framework to implement network-aware applications. Section III describes our recent investigation on reconfigurable middleware system. In each of Section II and III, we present the motivations to the proposed frameworks, the research issues we have been trying to answer, the current status of our research progress, and future work. We conclude our research efforts in Section IV.

II. RECONFIGURATION OF NETWORK APPLICATIONS

This section overviews the design of network applications in the Bio-Networking Architecture (Section A), and describes our research efforts in simulation study (Section B) and empirical implementation (Section C).

A. Network Application Design

Due to the dynamics of networks, network applications are often required to reconfigure their behaviors in order to satisfy changing demand from users or to deal with the changing network characteristics (e.g., available bandwidth and network topology). As the network scales in its size and complexity, however, it is more difficult and expensive to manually reconfigure the application behaviors. Our approach to this problem is to provide autonomous reconfigurability to network applications by introducing biologically-inspired evolutionary mechanisms. We have been implementing the evolutionary mechanisms in the Bio-Networking Architecture, and evaluating the mechanisms as a means of reconfiguring application behaviors.

In the Bio-Networking Architecture, a network application is modeled as a decentralized collection of autonomous distributed objects called *cyber-entities*. This is analogous to a

J. Suzuki, T. Nakano, K Fujii, N. Ikeda and T. Suda are with the Department of Information and Computer Science, University of California, Irvine, Irvine, CA 92697-3425 USA (email: {jsuzuki, tnakano, kfujii, nobby, suda}@ics.uci.edu).

N. Ikeda is also with Systems Integration Technology Center, Toshiba Corporation (email: ikeda@sitc.toshiba.co.jp).

This work has been supported by the National Science Foundation through Grants ANI-0083074 and ANI-9903427, by DARPA through Grant MDA972-99-1-0007, by AFOSR through Grant MURI F49620-00-1-0330, and by grants from the University of California MICRO Program, Hitachi, Hitachi America, Novell, Nippon Telegraph and Telephone Corporation (NTT), NTT Docomo, Fujitsu, and NS Solutions Corporation.

bee colony (a network application) consisting of multiple bees (cyber-entities). Each cyber-entity provides a functional service(s) related to the application, and performs biological behaviors (e.g. replication, reproduction, migration, death, etc.) similar to biological entities. The Bio-Networking Architecture allows cyber-entities to evolve by generating behavioral diversity among cyber-entities and executing natural selection.

Behavioral diversity among cyber-entities means that it is likely different cyber-entities implement different policies on their behaviors. It is generated through mutation and crossover during replication and reproduction processes. Natural selection is performed based on the concept of *energy*. Each cyber-entity stores and expends energy for living, as biological entities naturally strive to gain energy by seeking and consuming food. Cyber-entities gain energy in exchange for performing their services, and they expend energy to use computing resources such as CPU cycles and memory space on network nodes that they reside on. The abundance or scarcity of stored energy affects various behaviors and contributes to the natural selection process in evolution. For example, an abundance of stored energy is an indication of higher demand for the cyber-entity; thus the cyber-entity may be designed to favor reproduction in response to higher level of stored energy. A scarcity of stored energy (an indication of lack of demand or ineffective behaviors) may eventually cause the cyber-entity's death. Therefore, through successive generations, beneficial features are retained while detrimental behaviors become dormant, enabling cyber-entities (i.e. network applications) to adapt to dynamic and changing network environments.

Major behaviors that each cyber-entity has are listed below:

- *Energy exchange and storage*: Cyber-entities may gain/expend and store energy as described above.
- *Communication*: Cyber-entities communicate with others for requesting a service, fulfilling a service, or routing messages (e.g. discovery messages) for other cyber-entities.
- *Migration*: Cyber-entities may migrate from bionet platform to platform.
- *Replication and reproduction*: Cyber-entities may make copies of themselves (replication). Two parent cyber-entities may create a child cyber-entity (reproduction), possibly with mutation and crossover.
- *State change*: Each cyber-entity may have autonomous, active and inactive (sleeping) states during its lifetime. Cyber-entities in different states consume resources differently; therefore they expend energy at different rates.
- *Death*: Cyber-entities may die because of old age or energy starvation.
- *Relationship establishment*: Cyber-entities may have their own relationships with others.
- *Discovery*: Cyber-entities may discover other cyber-entities by sending a discovery message through their relationships.
- *Pheromone emission and sensing*: Cyber-entities may leave their pheromones on their local platform when it migrates to another platform. A pheromone is a pointer to a cyber-entity, which helps other cyber-entities to find it. Cyber-entities may also emit their pheromones to neighboring platforms for attracting other cyber-entities to come or establish relationships with them.
- *Resource sensing*: Cyber-entities may sense the type, amount, and unit cost of resources (e.g. CPU cycles and memory space) available on both a local and neighboring platforms. Each platform determines the unit cost of resources provided on that platform based on their availability.

Cyber-entities have their own policies for each behavior. Each behavior policy consists of one or more functions, or factors, which evaluate its environment and return numeric values. Each factor is given a certain weight relative to its importance, and behaviors are invoked if the total result of the evaluating function (e.g. weighted sum of factor values) exceeds a certain threshold. For example, the factors in the migration behavior, which affect when to migrate and where to migrate, include:

- Migration cost: A high cost for migration may discourage migration.
- Energy seeking, which encourages cyber-entities to move toward energy sources (i.e. end users requesting the services provided by the cyber-entities).
- Mutual repulsion, which encourages cyber-entities to repel with each other.
- Resource cost, which encourages cyber-entities to migrate to a network node whose resource cost is cheaper.

Behavioral diversity is generated, through mutation and crossover during replication and reproduction, by changing factors and their weight values associated with behaviors. In replication, for example, a newly replicated cyber-entity may derive its behavior factors and weight values from its parent with mutation. Mutation may remove/add factors and change their weight values, thereby ensuring a sufficient degree of behavioral diversity necessary for adaptation and evolution.

B. Simulation Study

In order to evaluate and demonstrate the reconfigurability of network applications through the evolutionary process described in the above Section A, we have developed the Bio-Networking Evolution Simulator [4]. The evolutionary mechanisms are implemented on the simulator based on genetic algorithms [5]. Each cyber-entity behavior is implemented as a chromosome, and each factor weight associated with a behavior is implemented as a gene.

Through the simulation study, we have been trying to answer the following research issues:

(1) *Effectiveness of energy*: As we described above, the natural selection in the Bio-Networking Architecture is designed to improve adaptability of network applications (i.e. cyber-entities) by removing the cyber-entities that do not fit the current network environment well. We need to evaluate the effectiveness of energy as a means to drive natural

selection process.

(2) *Effectiveness of crossover and mutation*: Having established the natural selection mechanism based on energy concept, we also need to validate the effectiveness of crossover and mutation as the mechanisms to generate behavioral diversity.

(3) *Performance of evolutionary process in dynamic networks*: We then need to investigate whether cyber-entities successfully adapt to the dynamic environmental changes (e.g. changes in user's location, network load, and network topology).

(4) *Acceleration of evolutionary processes*: As the network environment becomes more complex and dynamic, it is expected that the evolutionary process will take longer time. We need to investigate and design the strategies to accelerate evolutionary process.

(5) *Diversity Maintenance*: In order for network applications to be more robust and adaptive against dynamic environmental changes, it is important to maintain behavioral diversity among cyber-entities. We need to investigate and design a mechanism to keep the degree of behavioral diversity, and evaluate its effectiveness.

We have already evaluated core evolutionary mechanisms including natural selection, crossover and mutation (research issues 1 and 2) under relatively static and simple network environments. The simulations deployed various types of cyber-entities (e.g., productive cyber-entity that makes children very often to increase its availability, wandering cyber-entity that randomly moves around network to find new end users or network nodes whose resource cost is cheaper, etc.) to examine which types of cyber-entities increase their fitness to the environment. We confirmed that cyber-entities gradually reconfigure their behavioral policies through evolution in order to adapt themselves to the current network condition with regard to response time, distance to users, and resource consumption.

We have also examined if the evolution mechanisms evaluated in the static environments still work well in dynamic environments where end users are mobile and network topology changes (research issue 3). We confirmed that cyber-entities adapt themselves to dynamically changing network condition by reconfiguring their behavioral policies through evolution. Figure 1 depicts one of the simulation results obtained from a simulation where end users are configured to randomly move around network. In this figure, the vertical axis indicates energy gain (i.e. acquired energy – consumed energy), and the horizontal axis indicates simulation cycle (scaled in 600 cycles). The simulation result shows that evolutionary cyber-entities gain more energy than non-evolutionary ones, which means evolutionary cyber-entities adapt better to dynamic network condition by moving toward end users and avoiding network nodes whose resource cost is expensive (i.e. by increasing weight values of energy seeking and resource cost factors).

As Figure 1 shows, evolutionary cyber-entities decrease their energy gain at the early stage of simulation for exploring a new set of factor weights that are better suited to network environment, and then reach higher energy gain around 60,000 simulation cycles (approximately the 550th generation of cyber-entities). We are currently investigating how to make the evolutionary process more efficient (research issues 4) by reducing the overhead associated with the evolutionary mechanisms. The overhead includes energy loss and time delay in evolutionary process (see Figure 1). We are also investigating how to keep maintaining diversity in cyber-entity population, which is necessary and enough for evolutionary process (research issue 5).

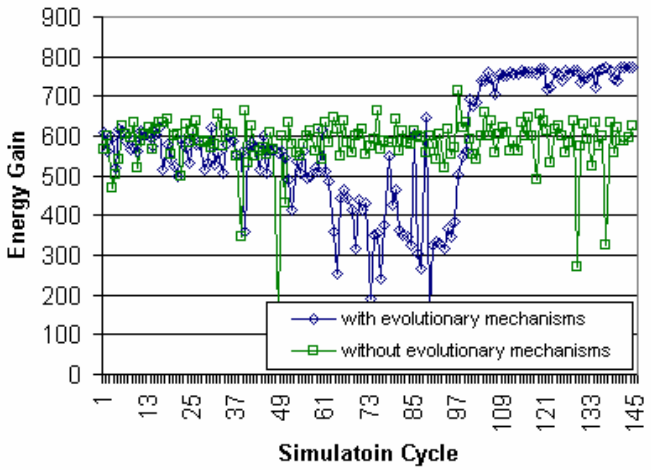

Fig. 1. The amount of energy gained by cyber-entities with and without evolutionary behavior reconfiguration (i.e. mutation and crossover mechanisms)

C. Empirical Implementation

Given an initial set of successful simulation results, we have been developing the Bio-Networking platform [6, 7], a middleware system that provides reusable software components for deploying and executing cyber-entities, in order to evaluate the characteristics of evolutionary reconfiguration of network applications on actual network environment.

The architecture of the Bio-Networking platform is organized as shown in Figure 2. The Bio-Networking platform runs on a Java virtual machine on a network node. It consists of four components; bionet services, bionet message transport, bionet container and bionet class loader. The bionet services provide a set of runtime services that cyber-entities frequently use for performing their services, sensing the current network conditions, and invoking their behaviors. Key bionet services are briefly described below:

- *Bionet lifecycle service*; allows cyber-entities to change their internal state, replicate, and reproduce.
- *Bionet relationship management service*; allows cyber-entities to establish, examine, update and eliminate their relationships with other cyber-entities.

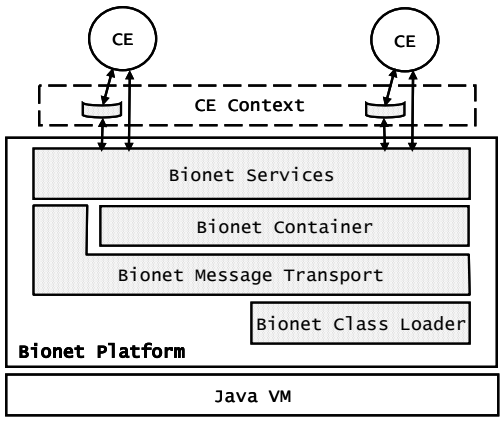

Fig. 2. Architecture of the Bio-Networking platform. CE stands for cyber-entity.

- *Bionet resource sensing service*; allows cyber-entities to inquire the type, amount and cost of available resources (CPU cycle and memory space).
- *Bionet energy management service*; keeps track of the energy levels of cyber-entities running on a local platform, and allows the cyber-entities to pay energy units for receiving services from another cyber-entity, for utilizing resources (CPU cycle and memory space), and for performing their behaviors.
- *Bionet discovery service*; allows cyber-entities to search for other cyber-entities on a remote node through their relationships. This discovery is called cyber-entity level discovery because the discovery is performed with the knowledge of cyber-entities (i.e. relationships).
- *Bionet cyber-entity sensing service*; allows cyber-entities to search for other cyber-entities running on a local and neighboring platforms. This discovery is called platform level discovery because the discovery is performed with the knowledge of platforms (i.e. platform connectivity).
- *Bionet pheromone emission service*; allows cyber-entities to emit their pheromones (traces) and to sense pheromones emitted by other cyber-entities.
- *Bionet topology sensing service*; allows cyber-entities to sense the existence of remote platforms within N hops.
- *Bionet migration service*; allows cyber-entities to migrate to another platform.

The Bionet message transport abstracts low-level operating and networking details such as I/O, concurrency, messaging and network connection management. The Bionet container dispatches incoming messages to cyber-entities running on a local Bio-Networking platform. The Bionet class loader dynamically loads class definitions of cyber-entities into a Java virtual machine when they migrate from a Bio-Networking platform to another. The CE context is an entry point for a cyber-entity to access bionet services (see Figure 2). It examines if a bionet service requested by a cyber-entity is available on the Bio-Networking platform, and if it is, it obtains a reference to the requested bionet service. The CE context is created and associated with each cyber-entity implicitly (automatically), when a cyber-entity is created, replicated, reproduced or migrated from another host.

We have implemented bionet message transport, bionet container, bionet class loader, CE context, and five bionet services (lifecycle, relationship, resource sensing, energy, and migration). We have been measuring the performance of each of those components. We confirmed that the performance results are competitive with those of existing distributed object platforms and mobile agent platforms. We have been also working actively in the Object Management Group (OMG) to reflect the key designs of the Bio-Networking platform to the OMG Super Distributed Objects specifications [8, 9].

We are currently implementing our evolution mechanisms, which have been used and evaluated in simulation study, on the Bio-Networking platform [10]. Through this empirical implementation work, we will start evaluating the reconfigurability of actual network applications through evolutionary process, along with the research issues described in the above Section B. We are planning to evaluate several applications such as content distribution [5], decentralized object sharing and discovery [11], smart home networks [3], and disaster response networks [12].

III. Reconfiguration of Middleware

The previous section describes our research work to design and implement network-aware applications that autonomously reconfigure themselves to adapt to dynamic network conditions. We have been expanding our research scope for making not only network applications but also underlying middleware system to be reconfigurable so that we can provide more adaptable networking systems.

Our approach to realize reconfigurable middleware is to compose middleware as a set of components. In this approach, middleware (1) senses its context such as available resources and system's configuration, (2) decides which components are required or most desirable against the obtained context, and then (3) loads the selected components to activate them. This approach allows middleware to adjust its functionality, performance and resource requirements according to the context.

For the initial phase to sense a context such as what kind of resources are available on a network node, how much bandwidth is available on a link, and which routing algorithms are used to communicate with neighboring nodes, we have identified the following research issues that we should address.

- Is context sensing performed when middleware is first deployed onto a network node (static reconfiguration), or continuously performed even after the deployment (dynamic reconfiguration)?
- What kind of information should be sensed as context? How does middleware sense context information?

For the second phase to determine a strategy to reconfigure

middleware according to an obtained context, the following research issues have been identified:

- In the case of dynamic reconfiguration, what are the triggers to start reconfiguring middleware?
- How does middleware decide its necessary functionalities or requirements based on the obtained context information, and determine its action(s) to satisfy these functionalities or requirements? For example, the actions may include determining which component to be activated/disabled or which component to be downloaded as mobile code.
- How does middleware know the effect of disabling components, replacing components with others, or newly loading components? Does it need to use any constraint specification (e.g. dependency and conflicts) to handle this problem? If middleware uses it, how does it describe the constraints? Also, is the constraint specification defined statically, or dynamically as well?
- Does middleware target single or multiple goals for its reconfiguration (e.g. minimization of resource utilization and guarantee of a defined throughput rate, etc.)? If it targets multiple goals, how does middleware prioritize multiple reconfiguration strategies?
- Does middleware know a statically defined set of components available for reconfiguration, or can middleware administrator dynamically introduce components? If middleware accepts newly-defined components at runtime, how does it know their characteristics such as functionality, performance and resource requirement? Should each component be self-descriptive? If it is, how does it describe its characteristics?
- Can we apply any biological metaphors to middleware reconfiguration process, as we did for reconfiguration of network applications? For example, [13] successfully uses a metaphor of the immune system to construct an adaptable web server which continuously measures the delivered quality of service and dynamically reconfigure its internal components by relaxing constraints between them. Can we apply a similar approach to reconfigurable middleware? Alternatively, can we possibly apply techniques of multiobjective evolutionary algorithm [14] when middleware targets multiple goals for its reconfiguration (see above)?

For the last phase to execute the reconfiguration strategy decided in the previous phase, we have identified the following research issues to address:

- How does middleware find necessary components to load? Where does it load them from? Does middleware load them from a local file system or download them through network [15]?
- How does middleware coordinate and bind different components? Can we effectively use or extend existing techniques such as component configurator [16], aspect weaving [17, 18], and metaobject protocols [19]?
- How small the granularity of components should be? In general, middleware can be more reconfigurable by using finer-grained components. However, the smaller the granularity of components is, the more difficult it is to bind them with each other. Can we effectively use or extend existing techniques such as multidimensional separation of concerns [20]?
- In the case of static reconfiguration, can we take any model-driven approach for middleware reconfiguration process, instead of reconfiguration through programming, for reducing its complexity? For example, can we use UML (Unified Modeling Language) [21] for specifying the characteristics of components and OCL (Object Constraint Language) [21] for specifying constraint specifications? Can we effectively use or extend the existing research work that tries to specify aspects and/or metaobjects in design models [22]? Can we possibly generate middleware code automatically from design models, for example, using UML action semantics and languages [23, 24, 25]?
- How does middleware detect errors that occurs in reconfiguration process (e.g. conflicts between components and failure of loading components), and recover (e.g. roll back) from the errors?
- How does middleware load components in a secure manner? For example, how does it avoid loading malicious components?

Keeping the above research issues in our minds, we have started investigating middleware reconfiguration mechanisms using the components implemented in the Bio-Networking platform. We will ultimately evaluate the characteristics and impact of the distributed system environment where both of network applications and middleware are reconfigurable to adapt to environment.

IV. Conclusion

We believe that one of the most important requirements in the near future communication systems is dynamic reconfigurability. This paper describes our research efforts regarding dynamic reconfiguration of network applications and middleware. It overviews the motivations to the proposed frameworks, the research issues we have been addressing, the current research status, and future work.

References

[1] *Report of Workshop on New Visions for Large-scale Networks: Research and Applications*, Large Scale Networking (LSN) Coordinating Group of the Interagency Working Group (IWG) for Information Technology Research and Development (IT R&D)

[2] http://netresearch.ics.uci.edu/bionet/

[3] T. Suda, T. Itao and M. Matsuo, "The Bio-Networking Architecture: The Biologically Inspired Approach to the Design of Scalable, Adaptive, and Survivable/Available Network Applications," K. Park (ed.) *The Internet as a Large-Scale Complex System*, Princeton University Press, 2002.

[4] http://netresearch.ics.uci.edu/bionet/resouces/evolution_simulator/

[5] M. Wang and T. Suda, M. Wang and T. Suda, "The Bio-Networking Architecture: A Biologically Inspired Approach to the Design of Scalable,

Adaptive, and Survivable/Available Network Applications," Proc. of the First IEEE Symposium on Applications and the Internet (SAINT), January 2001.

[6] http://netresearch.ics.uci.edu/bionet/resouces/platform/

[7] J. Suzuki and T. Suda, "An Overview of the Bio-Networking Architecture," The Super Distributed Objects Forum, Object Management Group TC meeting at Helsinki, OMG document number: sdo/02-10-05, Helsinki, Finland, October 2002.

[8] S. Sameshima, J. Suzuki and T. Suda, "The Initial Submission to the Platform Independent Model (PIM) and Platform Specific Model (PSM) for Super Distributed Objects," Super Distributed Objects Domain SIG, Object Management Group, September 2002.

[9] S. Sameshima, S. Arbanowski and J. Suzuki, "OMG Super Distributed Objects White Paper," Super Distributed Objects Domain SIG, Object Management Group, July 2001.

[10] J. Suzuki and T. Suda, "Adaptive Behavior Selection of Autonomous Objects in the Bio-Networking Architecture," Proc. of the First Annual Symposium on Autonomous Intelligent Networks and Systems, May 2002.

[11] M. Moore and T. Suda, "Distributed Discovery in Peer-to-Peer Network," *Proc. of the First Annual Symposium on Autonomous Intelligent Networks and Systems*, January 2002.

[12] J. Suzuki and T. Suda, "Middleware Support for Disaster Response Infrastructure," *Proc. of the First IEEE Workshop on Disaster Recovery Networks*, June 2002.

[13] J. Suzuki and Y. Yamamoto, "Biologically-inspired Autonomous Adaptability in a Communication Endsystem: An Approach Using an Artificial Immune System," *IEICE Transactions on Information & System*, Vol. E84-D, No. 12, pages 1782-1789, December 2001.

[14] D. A. Van Veldhuize and, G. B. Lamont, "Multiobjective Evolutionary Algorithms: Analyzing the State-of-the-Art," Evolutionary Computation, Vol. 8, No. 2, pages 125-147, 2000.

[15] N. Narasimhan and V. Vasudevan, "Migratable Middleware for Mobile Computing," Proc. of the First Annual Symposium on Autonomous Intelligent Networks and Systems, May 2002.

[16] F. Kon and R. H. Campbell, "Supporting Automatic Configuration of Component-Based Distributed Systems," *Proc. of 5th USENIX Conference on Object-Oriented Technologies and Systems (COOTS'99)*, May 1999.

[17] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm and W. G. Griswold, "An Overview of AspectJ," *Proc. of the European Conference on Object-Oriented Programming (ECOOP), 2001.*

[18] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, J. Irwin, "Aspect Oriented Programming," *Proc. of the European Conference on Object-Oriented Programming (ECOOP),* 1997.

[19] G. Kiczales, J. D. Rivieres, D. Bobrow, *The Art of the Metaobject Protocol*, MIT Press, 1991.

[20] P. Tarr, H. Ossher, W. Harrison, S. M. Sutton, "N degrees of Separation: Multidimensional Separation of Concerns," *Proc. of the ACM International Conference on Software Engineering*, 1999.

[21] The Object Management Group, *Unified Modeling Language Specification 1.4.*, 2000.

[22] J. Suzuki and Y. Yamamoto. "Extending UML for Modeling Reflective Software Components." Proc. of *the International Conference on Unified Modeling Language,* 1999.

[23] The Object Management Group, *Action Semantics for UML Specification*, 2001.

[24] G. Sunyé, F. Pennaneac, W-M. Ho, A. L. Guennec, J.-M. Jézéquel, "Using UML Action Semantics for executable modeling and beyond," *Advanced Information Systems Engineering,* Springer LNCS 2068, 2001.

[25] S. Mellor, M. Balcer, *Executable UML: A Foundation for Model Driven Architecture*, Addison Wesley, 2002.