A Decentralized Policy Coordination Facility in OpenWebServer

Junichi Suzuki
Department of Computer Science,
Graduate School of Science and Technology,
Keio University
Yokohama City, 223-8522, Japan
+81-45-563-3925
suzuki@yy.cs.keio.ac.jp

Yoshikazu Yamamoto
Department of Computer Science,
Graduate School of Science and Technology,
Keio University
Yokohama City, 223-8522, Japan
+81-45-563-3925
yama@cs.keio.ac.jp

Abstract
The system adaptability is one of the most important capabilities required for a next generation open system. Many research efforts have developed adaptable systems that allow various applications to meet their specific requirements by combining different policies. However, the policy coordination is still problematic. Fine-grained policies are sometimes not orthogonal, but often have complex constraint dependencies with each other. This paper describes the principles for sophisticated policy coordination, and then presents our coordination facility designed to be used in OpenWebServer, our adaptive web server. Our method is inspired by the natural immune system, a truly autonomous and decentralized system, and coordination facility is developed with iNet, which is a framework for building an artificial immune network. This facility can relaxes constraints between policies and determine the most appropriate set of policies for a given system condition. The coordination process is performed through decentralized interactions among policies without a single point of control, as the natural immune system does.

Keywords
system adaptation, system evolution, artificial immune system, policy management, component composition

1. Introduction
The integration of system adaptation support into traditional system architecture is important for realizing the next generation of open systems. A promising approach is the development and deployment of a single system infrastructure that can plug-in and combine the various capabilities.

The heart of such an integrated architecture is the need to provide diverse applications with their desired qualities of services. We can solve this problem with one of four approaches. First, we can overbuild capacity so that even the most stringent application meets its requirements. This is unlikely to be cost-effective. Second, we can require all applications to adapt to the current system configuration. This is impractical for a wide range of applications and sometimes impossible. Third, we can scrap an application and re-build a new one from scratch so that it adapts to given requirements. This is often too economically expensive. Therefore, the diversity of requirements is likely to be efficiently satisfied only by the forth alternative, that is, by tailoring system's components and services to the characteristics of the application. This option requires the application specifies its service requirements. For example, a communication end-system may specify its requirement for bandwidth reservation so that it transmits multimedia data continuously. Another application may specify its concurrency policy so that it performs simultaneous information processing efficiently. A key observation to adaptable system is that the policy management is a unifying theme on which the functions and facilities of the new integrated standards can be constructed.

Software adaptation has been studied by various research efforts such as reflection [1], open implementation [2], adaptive programming [3], aspect-oriented programming [4], component composition [5, 24], and collaboration-based design [6], as well as quality of service (QoS) policy management in the networking community [7]. In every approach, there is an entity representing a system execution policy. For example, it is called *metaobject* in the context of reflection, *concerns* in the principle of separation of concern [23], *component* or *plug-in* in component composition, and *aspect* in aspect-oriented programming. Applications can adapt to a given requirement by adding, customizing or replacing the entities. For referring to such entities, we use the term *policy* in this paper.

Policies tend to become fine-grained in highly adaptable systems; thereby the number of them increases. However,

the greater the number of policies, the more complexity and difficulty in maintaining and coordinating them. Fine-grained policies often have complex constraint relationships with each other. In this paper, the policy coordination means inspecting the every dependency between policies and then resolving co-use/conflict constraints to produce an optimized combination from feasible policies. The simplest coordination strategy is writing a long sequence of hard-written if/case statements in a program. Another strategy is using the multiple inheritance in a class-based object-oriented language. Both suffer from combination explosion, and cost lots of labor for configuring if/case statements or inheritance relationships. They are also fragile for changing policy specifications.

A highly adaptable system requires a sophisticated policy control mechanism that coordinates policies in more dynamic and flexible manner. It is often ad-hoc and has not been addressed in the literature at large, unfortunately, how to coordinate policies consistently throughout the system's lifetime. This paper describes our biologically inspired coordination facility developed with iNet [25], which is a framework simulating the natural immune system. It determines an optimized set of policies in the decentralized and dynamic manner. This capability provides robustness for re-configuring a policy space

This paper is organized as follows; Section 2 presents our context and goal of developing a highly adaptable system. Section 3 overviews the natural immune system to explain why our work uses its metaphors. Section 4 presents how to incorporate the metaphors in iNet for coordinating fine-grained policies. Section 5 illustrates preliminary results. In Sections 6 and 7, we conclude with a note on the current status of the project and present future work.

2. Policy management in OpenWebServer

Our research vehicle for demonstrating an adaptable system and exploring an effective policy coordination mechanism is OpenWebServer [8-10]. OpenWebServer is both an adaptive web server and a framework for building a versatile web server. It reifies various aspects in a web server as metaobjects. Each metaobject represents a policy for concurrency, I/O event dispatching, protocol filtering, connection management, caching, logging, and service redundancy. We can produce high-throughput, highly available or fault tolerant servers by tuning the combination of these policies dynamically or statically [9].

Coordinating and composing multiple policies, unfortunately, require precise knowledge of their constraints and implications. For example, a single-threaded reactive server[1] [11] is efficient when (1) it runs on a uni-processor platform, (2) the average size of requested files is relatively small, and (3) the hit rate from simultaneous connections is relatively low. As the hit rate increases, however, the multiple thread strategies such as thread-per-request, thread pool and thread-per-connection are better choices only if the underlying operating system supports threads. They can scale well in a multi-processor platform. A constraint of using the thread-per-connection policy is that the server operates the HTTP version 1.1. As the size of requested files grows, the asynchronous I/O event dispatching [2] outperforms the synchronous concurrency models using BSD standard socket functions [12, 13]. A restriction of using asynchronous I/O is that all the operating systems do not support it. A thread pool can be designed in the active or passive manner. A passive thread pool implemented with a kernel-level asynchronous I/O and simultaneous `accept()` calls[3] is much more efficient than other pools, though it is not a portable solution.

As such, it is complex to determine a suitable set of policies consistently according to the current environment and situation. It is tedious and error-prone. Coordinating fine-grained policies increases the potential for subtle programming mistakes and unexpected fatal errors. Therefore, the rules for combining different policies are pre-defined statically and maintained in the centralized manner in the previous version of OpenWebServer, as the most QoS-enabled systems do today.

Our iNet-based facility addresses this limitation, and provides an autonomous and decentralized policy coordination mechanism that determines an optimized combination of policies by relaxing their constraints and conflicts.

Our experience of developing OpenWebServer shows that a policy coordination facility should meet the following design principles:

- The facility should be integrated for various applications to adapt to their own requirements, but modular for supporting a wide range of fine-grained policies.
- The facility should support a scalable policy space that allows any policy to be introduced, altered and removed dynamically.
- The facility should evaluate the effectiveness of a delivered set of policies and learn a better combination for the future usage.

[1] Typically implemented by calling `select()` for simultaneous connections in a polling loop.

[2] e.g., Windows NT provides asynchronous I/O system calls such as `WSA*()` and `TransmitFile()`, though it can be simulated with user-level library [22].

[3] Multiple idle threads can call `accept()` to a single socket with this mechanism.

- The coordination process should be transparent, or orthogonal, to a system's functional logic.
- The coordination process should be performed through decentralized interactions among competitive and sometimes conflicting policies, so that outcomes emerge with a context and dependencies between policies. This "coordination without coordinator" principle brings flexibility and scalability of policy management.
- The coordination process should be decoupled from the process mapping a policy to a system components

Generally, highly adaptable system requires the observation of a target system, followed by careful coordination of various policies, and finally evaluation of a delivered set of policies. This paper focuses on observing the system's condition and coordinating policies.

In terms of the networking QoS research, our work is categorized in the application-level QoS policy coordination within a communication end-system. We do not discuss QoS policy guarantees and transport/network level policy specifications. In terms of reflection and aspect-oriented programming, this paper focuses on the process to determine a strategy of metaobject composition and aspect weaving, respectively.

3. Natural immune system

This section overviews the natural immune system. We have worked for applying metaphors in the natural immune system, especially the immune network, to our policy coordination facility in OpenWebServer.

3.1 Immune system

The natural immune system is a subject of great research interests because it provides powerful and flexible information processing capability as a decentralized intelligent system. It has some important computational aspects such as self/non-self discrimination, learning, memory, retrieval, pattern matching and emergent behavior. The immune system provides an excellent model of adaptive operation at the local level and of emergent behavior at the global level. There exists several theories to explain immunological phenomena and software models to simulate various components in the immune system. They have been used for machine learning, computer security, fault detection, image processing and searching, and even used for prototypes of business production systems [14].

The basic components of the immune system are macrophages, antibodies and lymphocytes. B-lymphocytes are the cells maturing in the bone marrow. Roughly 10^7 distinct types of B-lymphocytes are contained in a human body, each of which has a distinct molecular structure and produces antibodies from its surface. The antibody

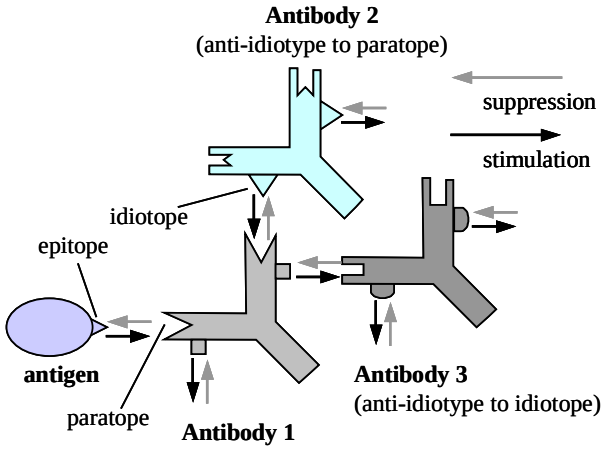

Figure 1: Interactions in the immune network. The immune response to an antigen results in the formation of anti-idiotype antibodies specific to the individual idiotope of the primary antibody (Antibody 1). These anti-idiotype antibodies (Antibody 2 and 3) in turn induce the formation of anti-anti-idiotype antibodies.

recognizes and eliminates a specific type of antigens, e.g. viruses, which are the foreign substances invading a human body. The key portion of antigen that is recognized by the antibody is called *epitope*, which is the antigen determinant (see Figure 1). *Paratope* is the portion of antibody that corresponds to a specific type of antigens (Figure 1). Once an antibody combines an antigen via their epitope and paratope, the antibody start to eliminate the antigen. Recent studies in immunology have clarified that each type of antibody also has its own antigenic determinant, called an *idiotope* (Figure 1). This means an antibody is recognized as an antigen by another antibody.

3.2 Immune network

Based on this fact, Jerne proposed the concept of the *immune network*, or *idiotypic network* [26-28], which states that antibodies and lymphocytes are not isolated, but they are communicating with each other (Figure 1). The idiotope of an antibody is recognized by another antibody as an antigen. This network is formed on the basis of idiotope recognition with the stimulation and suppression chains among antibodies. Thus, the immune response eliminating foreign antigens is offered by the entire immune system (or, at least, more than one antibody) in a collective manner, although the dominant role may be played by a single antibody whose paratope fits best with the epitope of the specific invading antigen. The immune network also helps to keep the quantitative balance of antibodies. Through stimulation/suppression interactions, the populations of specific antibodies increase very rapidly following the recognition of any foreign antigen and, after eliminating the antigen, decrease again [29]. Performed based on this self-

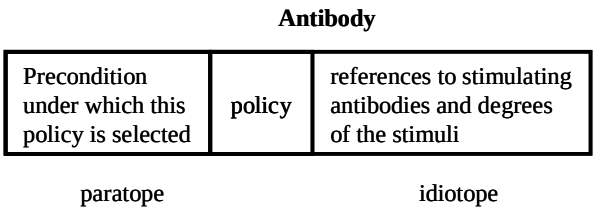

Figure 2: The antibody structure in iNet

regulating mechanism, the immune response [4] has an emergent property through many local interactions.

The network structure is not fixed, but varies continuously according to dynamic changes of environment. This flexible self-organizing function is realized mainly by incorporating newly generated cells and/or removing useless ones. The new cells are generated by both gene recombination in bone marrow and mutation in the proliferation process of activated cells. Although many new cells are generated every day, most of them have no effect on the existing network and soon die away without any stimulation. Due to such enormous loss, the immune system maintains an appropriate set of cells so that the system can adapt to environmental changes in the piecemeal way.

4. Immunologically-inspired policy coordination with iNet

As described in the previous section, the phenomena appeared in natural immune network meet most of the design requirements for sophisticated policy coordination listed in Section 2. This section presents our iNet framework for simulating the immune network, and describes how we can design a policy coordination facility from the immunological point of view.

4.1 Policy coordination facility as an artificial immune network

In our immunological perspective, a system's current conditions are considered as an antigen, e.g. the number of simultaneous network connections, average size of requested files, types of operating systems, the number of available processors, and supported types of protocols. Each QoS policy is regarded as an antibody, e.g. concurrency policies (thread-per-request, active/passive thread pool, thread-per-connection, etc.) and I/O event dispatching policies (synchronous and asynchronous). Policies are linked with each other based on the stimulation and suppression relationships. The interaction relationships are modeled based on constraints in combining policies.

The goal of our method is that a policy coordination facility augmented by an artificial immune network determines the best combination of policies suitable for a given condition, as the natural immune network does. This coordination process is performed in the bottom-up, or emergent, manner through the decentralized local interactions between policies without any single point of control.

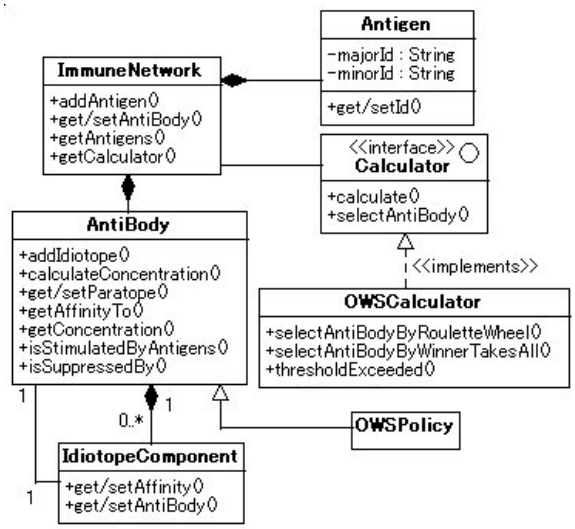

Figure 3: A UML class diagram for the kernel of iNet

4.2 iNet architecture

Our policy coordination facility is developed with iNet [25]. iNet is a Java-based framework for building artificial immune networks. It has been open for public use at Keio University since 1999, and will be released for researchers simulating the immune network mechanisms and exploring the design space of artificial immune networks. iNet serves as an infrastructure in our several research projects building biologically inspired multi-agent systems, where artificial immune networks optimize the strategy and behavior of an agent in a pursuit game [15] and Robocup soccer game simulation [16].

Figure 2 shows the structure of an antibody used in iNet. Each antibody represents a policy. Paratope represents a precondition under which a certain policy is selected. Idiotope represents the references to other stimulating antibodies with degrees of the stimuli. Both paratope and epitope are typed with string. Matching both string values means an antibody responds to an antigen. Table 1 shows a list of preconditions and policies supported in OpenWebServer. Both precondition and policy are specified by a combination of major and minor IDs. Each major policy ID specifies a policy group. Figure 3 depicts a class diagram showing the kernel of iNet. Every policy used in OpenWebServer is defined by the class `OWSPolicy` derived from `AntiBody`. `Calculator` and its implementation classes are used for calculating all the

[4] the humoral immune response, precisely

Paratope major ID	Paratope minor ID	Policy major ID	Policy minor ID
FILE_SIZE	L, M, S	CONCURRENCY	REACTIVE
NO_OF_CONNECTION	M, A, F		THREAD_PER_REQUEST
NO_OF_CPU	M, S		ACTIVE_THREAD_POOL
OS_THREAD_SUPPORT	T, F		PASSIVE_THREAD_POOL
OS_ASYNC_IO_SUPPORT	T, F		THREAD_PER_CONNECTION
AVAILABLE_THREADS	M, A, F	IO	SYNCH
SUPPORTED_PROTOCOL	HTTP10, HTTP11		ASYNCH
		CACHE	LRU
			FIFO
		PROTOCOL	HTTP10
			HTTP11

Table 1: A list of supported kinds of paratope and policies

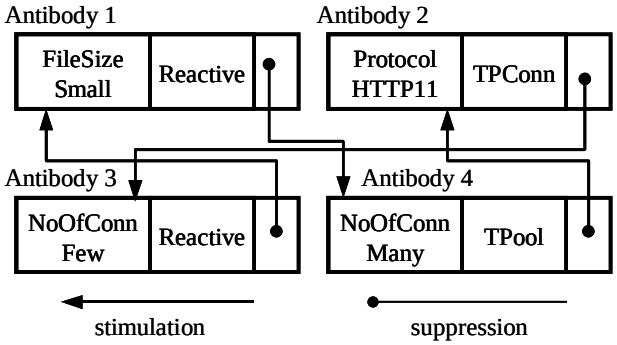

Figure 4: A sample immune network

antibodies' populations through their interactions and selecting an antibody whose population is the highest.

4.3 Interactions among antibodies

This section describes the interaction among antibodies, which is the basic structure and dynamics of our policy decision network, with a simple sample.

Figure 4 shows a simple subset of our immune network. This network is used to determine the best-suited concurrency policy according to a current system condition. It contains four antibodies representing three kinds of policies; single-threaded reactive, thread pool and thread-per-connection. Antibody 1 represents that the single-threaded reactive policy is activated when the average of requested files is relatively small. However, the thread pool policy is activated if the number of HTTP simultaneous connections grows, because antibody 1 stimulates antibody 3. Inversely, the reactive policy is suppressed by the thread pool policy, if the server has to handle many connections even when the average file size is small.

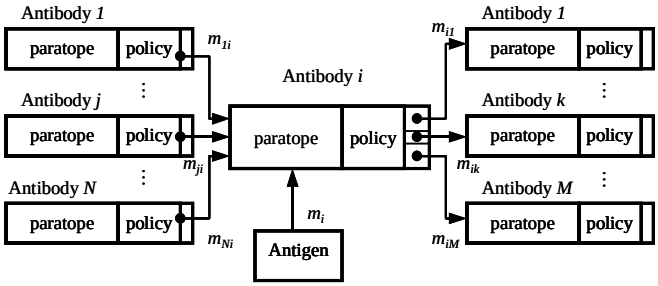

Figure 5: An antibody and its neighbors

Now, suppose that OpenWebServer (1) transfers relatively small size of files, (2) handles relatively many connections, and (3) supports the HTTP version 1.1. In this situation, these three antigens stimulate antibodies 1, 2 and 4 simultaneously. Then, the populations of the antibodies increase. However, each population varies through the stimulating/suppressing interactions indicated by arrows between antibodies. As a result, the population of the antibody 2, i.e. thread-per-connection, would increase, and then it would be selected by the immune network.

In the case where OpenWebServer (1) transfers relatively small size of files, (2) does not have to handle many connections, and (3) supports the HTTP version 1.1, antibody 1, i.e. the reactive policy, would be selected in the same way.

4.4 Dynamics of antibody selection

Figure 5 shows a generalized view of an antibody within in an immune network. The *i*-th antibody stimulates *M* antibodies and suppresses *N* antibodies. m_{ji} and m_{ik} denote affinities between antibody *j* and *i*, and between antibody *i* and *k*, respectively. The affinity means the degree of stimulation or suppression. m_i is an affinity between an

antigen and antibody *i*. All affinities are supposed to be defined a priori in this paper (see also Section 6).

The antibody population is represented by the concept of concentration in our method. The concentration of the *i*-th antibody, denoted by a_i, is calculated with the following equations, which is originally proposed by Farmer et al. [17, 18] and then extended by Ishiguro et al. [19, 20]:

$$\frac{dA_i(t)}{dt} = \left(\frac{1}{N} \sum_{j=1}^{N} m_{ji} \cdot a_j(t) - \frac{1}{M} \sum_{k=1}^{M} m_{ik} \cdot a_k(t) + m_i - k \right) a_i(t)$$

$$a_i(t) = \frac{1}{1 + \exp(0.5 - A_i(t))}$$

In the first equation, the first and second term of the right hand side denote the stimulation and suppression from other antibodies. m_{ji} and m_{ik} are positive values between 0 and 1. The third term, m_i, is 1 when antibody *i* is stimulated directly by an antigen, otherwise 0. The forth term, *k*, denotes the dissipation factor representing the antibody's natural death. This value is 0.1. The initial concentration value for every antibody, i.e. $a_i(0)$, is 0.01.

The second equation is the function that is used to squash the parameter $A_i(t)$, calculated by the first equation, between 0 and 1.

Every antibody's concentration is calculated 30 times repeatedly. Then, an antibody is selected from each policy group. If no antibody exceeds the predefined threshold (0.7) during the 30 calculation steps, the antibody of the highest concentration is selected, i.e. winner-tales-all strategy. If an antibody's concentration exceeds the threshold, an antibody is selected based on the probability proportional to the current concentrations, i.e. roulette-wheel selection strategy. An antibody whose concentration is below 0.2 is never selected.

5. Results of preliminary experiments

We have conducted some initial experiments to validate the feasibility of our policy coordination method. We built an artificial immune network incorporating 10 policies listed in Table 1. It defines 38 stimulation/suppression relationships with their fixed affinities (see Section 6).

In our first experiment, we evaluated the static system adaptability by supplying some sets of antigens regarding the underlying system platform. For example, when OpenWebServer is executed on a uni-processor platform running an operating system that supports the thread and asynchronous I/O, our policy coordination facility chose the single-threaded reactive policy at first, instead of an asynchronous concurrency policy. Our artificial immune network is configured to select a policy conservatively, and scale to more sophisticated ones when the original policy is not appropriate in the current system condition. In a multi-processor platform, the threading concurrency policies had higher priorities than a single-threaded concurrency model. In the situation that the system runs on an operating system that does not support the threading capability, the single-threaded reactive policy was selected independently of the underlying CPU architecture.

The second experiment tested the dynamic system adaptability by supplying antigens indicating the runtime system conditions. While the average size of requested files is small and the server's hit rate is low, the single-threaded reactive policy was selected. When the file size is within the middle range, the server was still single-threaded, or incorporated thread-per-request or thread pool policy. As the file size grows relatively large, an asynchronous concurrency policy had higher priority. This policy was sometimes used with a caching policy. When the number of network connections increases, the thread pool, thread-per-connection and asynchronous concurrency were selected depending on the multi-CPU, HTTP 1.1 and asynchronous I/O support respectively.

Our initial experiments are simple, but we can see our facility selects appropriate antibodies to both the static and dynamic situations by changing the priorities among antibodies. Note that the coordination process is not performed by a single coordinator entity, but through decentralized interactions among antibodies.

6. Future work

This paper focuses on the policy coordination of critical determinants to the HTTP server performance: concurrency and I/O [12, 13]. We are testing our coordination facility with more complex experiments using greater number of antibodies. Our latest immune network includes new 12 policies regarding sever redundancy, application-level service support (e.g. CGI, Servlet and Java Server Pages), logging, and protocol pipeline-parsing.

In the current structure of the iNet-based immune network, the stimulation/suppression relationships are defined statically using fixed affinity values. This means the coordination process is basically performed as an immune network designer expected at the design time. However, this network does not provide a true dynamic feature as described in Section 3.2. The coordination facility should be able to add/remove relationships between antibodies and change affinity values at run-time for OpenWebServer to evolve more effectively. We chose a conservative strategy for building an artificial immune network as the first research step, because the "trial and error"-style coordination can often result in fatal errors due to delivering a disallowed combination of policies. Now that we have validated the effectiveness of our artificial immune network for OpenWebServer, however, we are moving forward to more dynamic coordination as an extension of

this work. The next goal is to provide the emergence of the knowledge of policy combination.

As for a mathematical model to simulate the phenomena of the natural immune network, there exist several models such as liner networks, cyclic networks, Cayley-tree-like network and generalized shape-space model, which are proposed by theoretical immunologists [21]. Our coordination facility uses a cyclic network model proposed by Farmer et al. [17, 18]. We plan to evaluate other network models in more detail.

Also, we plan to incorporate some additional immunological concept, e.g. tolerance and immune memory in iNet.

7. Conclusion

This paper addresses current problems and limitations in coordinating fine-grained QoS policies, and then proposes a new decision-making method inspired by the natural immune network. Our iNet-based coordination facility defines each policy as an antibody, and selects the most appropriate set of policies through the decentralized interactions among antibodies. We believe our work provides a blue print showing a decentralized coordination mechanism as a next logical extension to existing adaptive and QoS-enabled systems. Information regarding iNet and OpenWebServer can be obtained from www.yy.cs.keio.ac.jp/~suzuki/project/aisf/ and www.yy.cs.keio.ac.jp/~suzuki/immunity/.

8. Acknowledgements

We thank Toshimitsu Minami and Mio Yamamoto for our constructive discussions, which have been critical to develop iNet and OpenWebServer. They expanded application studies of our iNet-based artificial immune network. We are also grateful to Nozomu Matsui for improving an earlier version of this paper.

9. References

[1] G. Kiczales et al. *The Art of the Metaobject Protocol*. MIT Press, 1991.

[2] G. Kiczales, Beyond the Black Box: Open Implementation, *IEEE Software*, vol. 13, no. 1, 1996.

[3] K. J. Lieberherr, *The Art of Growing Adaptive Object-Oriented Software*, PWS Publishing Company, 1995.

[4] G. Kiczales et al., Aspect-Oriented Programming, In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP'97)*, Springer LNCS 1241, 1997.

[5] M. Aksit et al., Abstracting Object-Interactions Using Composition-Filters, In *Object-based Distributed Processing*, R. Guerraoui et al. (eds.), 1993.

[6] M. Mezini and K. Lieberherr, Adaptive Plug-and-Play Components for Evolutionary Software Development, In *Proceedings of the Conference of Object-Oriented Programming, Systems and Languages (OOPSLA'98)*, 1998.

[7] T. A. Campbell, QoS Architectures, In *Multimedia Communications Networks*, M. Tatipamula and B. Khasnabish (Eds.), Artech House Publishers, Chapter 3, 1998.

[8] J. Suzuki and Y. Yamamoto. OpenWebServer: An Adaptive Web Server using Software Patterns. In *IEEE Communications*, Vol. 37, No. 4, pages 46-52, April 1999.

[9] J. Suzuki and Y. Yamamoto, Dynamic Adaptation in the Web Server Design Space using OpenWebServer, In *Proceedings of SPA '99*, March 1999.

[10] J. Suzuki and Y. Yamamoto. Building an Adaptive Web Server with a Meta-architecture: AISF approach. In *Proceedings of SPA'98*, March 1998.

[11] D. C. Schmidt. Reactor - An Object Behavioral Pattern for Event Demultiplexing and Event Handler Dispatching. In *Proceedings of the First Pattern Languages of Programs*, 1994.

[12] J. C. Hu and D. C. Schmidt. Developing Flexible and High-performance Web Servers with Frameworks and Patterns. In *ACM Computing Surveys*, May 1998.

[13] J. C. Hu, S. Mungee and D. C. Schmidt. Principles for Developing and Measuring High-Performance Web Servers over ATM". In *Proceedings of INFOCOMM'98*, 1998.

[14] D. Dasgupta (Ed.), *Artificial Immune Systems and Their Applications*, Springer, 1999.

[15] T. Minami, *Modeling a Multi-Agent System with Artificial Immune Network*, M.S. Thesis, Department of Computer Science, Graduate School of Science and Technology, Keio University, 2000 (in Japanese).

[16] M. Yamamoto, *iPlayers: A Decentralized Behavior Arbitration among Soccer Agents*, B.S. Thesis, Department of Information and Computer Science, Keio University, 2000 (in Japanese).

[17] J. D. Farmer, N. H. Packard and A. S.Perelson, The Immune System, Adaptation, and Machine Learning, *Physica, D 22*, 184/204, 1986.

[18] J. D. Farmer, S. A. Kauman and N. H. Packard, Adaptive Dynamic Networks as Models for the Immune System and Autocatalytic Sets, Technical Report LA-UR-86-3287, LosAlamos National Laboratory, 1986.

[19] A. Ishiguro, T. Kondo, Y. Watanabe and Y. Uchikawa, Dynamic Behavior Arbitration of Autonomous Mobile Robots Using Immune Networks, In *Proceedings of*

IEEE International Conference on Evolutionary Computation, 722/727, 1995.

[20] A. Ishiguro, T. Kondo, Y. Watanabe and Y. Uchikawa, An Immunological Approach to Behavior Arbitration for Autonomous Mobile Robots, In *Proceedings of International Symposium on Artificial Life and Robotics*, 132/137, 1996.

[21] D. Chowdhury, Immune Network: An Example of Complex Adaptive Systems, In *Artificial Immune Systems and Their Applications*, D. Dasgupta (Ed.), pp. 89-104, Springer, 1999.

[22] I. Pyarali, T. Harrison, D. C. Schmidt, and T. Jordan, Proactor: an Object Behavioral Pattern for Demultiplexing and Dispatching Handlers for Asynchronous Events, In *Pattern Languages of Program Design 4*, Harrison, Foote and Rohnert (eds.), Addison-Wesley, 1999.

[23] W. L. Hursch and C. V. Lopes, Separation of Concerns, Technical Report NU-CCS-95-03, Northeastern University, 1995.

[24] Y. Ichisugi, EPP: Extensible Type System Framework for a Java Pre-Processor, In Proceedings of *SPA'99*, 1999.

[25] J. Suzuki and Y. Yamamoto, iNet: An Extensible Framework for Simulating Immune Network, Submitted to *Special Track on Artificial Immune Systems and Their Applications* at *IEEE International Conference on Systems, Man, and Cybernetics 2000* (SMC'00), 2000.

[26] N. K. Jerne, The Immune System, *Scientific American*, Vol. 229, No. 1, pp. 52-60, 1973.

[27] N. K. Jerne, The Generative Grammar of the Immune System, *EMBO Journal*, Vol. 4, No. 4, 1985.

[28] N. K. Jerne, Idiotypic Networks and Other Preconceived Ideas, *Immunological Review*, Vol. 79, No.5-24, 1984.

[29] J. Kuby, *Immunology*, third edition, Freeman, 1997.