Building an Artificial Immune Network for Decentralized Policy Negotiation in a Communication Endsystem: OpenWebServer/iNexus Study

Junichi Suzuki (suzuki@yy.cs.keio.ac.jp)
Department of Computer Science, Graduate School of Science and Technology,
Keio University
Yokohama City, 223-8522, Japan

and

Yoshikazu Yamamoto (yama@ics.keio.ac.jp)
Department of Computer and Information Science, Faculty of Science and Technology,
Keio University
Yokohama City, 223-8522, Japan

Abstract

This paper describes the adaptability of communication software through a biologically-inspired policy negotiation. Many research efforts have developed adaptable systems that allow various applications to meet their specific requirements by configuring different design and optimization policies. Navigating through many policies manually, however, is tedious and error-prone. Our negotiation engine, named iNexus, provides an autonomous and decentralized policy management. iNexus is a foundation part of OpenWebServer, which is both a web server and an object-oriented framework for building Internet versatile servers. Its design is inspired by the natural immune system, particularly immune network, a truly autonomous and decentralized system. iNexus manages a wide range of policies, even inter-dependent instead of orthogonal ones, and determines the most appropriate set of policies for a given system condition by relaxing constraints between them. The policy negotiation process is performed through decentralized interactions among policies without a single point of control, as the natural immune system does. This paper discusses a communication system can evolve continuously with the biological concepts and mechanisms, adapting itself to ever-changing environment.

Keywords

system adaptation, system evolution, artificial immune system, policy management, policy negotiation, web server

1. Introduction

The communication software like web servers and middleware has emerged as an important architectural component in building the electronic commerce. However, the design of communication software is still hard. It contains both inherent complexities, such as fault detection and recovery, and accidental complexities, such as the continuous rediscovery and re-invention of key concepts and components. It also has a remarkably rich set of options, or policies, when designing, optimizing and configuring a communication endsystem, e.g. web server. No single policy fits different kinds of endsystems or their workloads [1, 2]. Therefore, it is essential to develop open-ended and adaptive framework that allows building optimally configured network systems [3].

The existence of all the feasible policies ensures that endsystems can be tailored to their users' or applications' requirements. Navigating through many design and optimization policies, however, is tedious and error-prone. Developers face the significant efforts of engineering an endsystem from the ground up, resulting in ad-hoc solutions. Such systems are often hard to maintain, customize and tune, since much of the engineering tasks are spent just for trying to get the system operational. It has not been addressed in the literature at large, unfortunately, how to navigate and coordinate policies consistently throughout the system's lifetime.

This paper describes our policy negotiation facility, named iNexus, which manages and coordinates policies in the autonomous manner. Its core engine is designed with the metaphors in the natural immune system. iNexus dynamically determines the most appropriate set of policies from a policy pool for a given system condition by relaxing constraints between policies. Our biologically-inspired negotiation process is performed through decentralized interactions among policies without a single point of control, as the natural immune system does.

iNexus is deployed on a web server, called OpenWebServer [4], so that it evolve the server in a changing environment by re-configuring the system components based on policies suited to the current system condition. OpenWebServer is both an adaptive web server and an object-oriented framework for developing an Internet versatile server (see Figure 1). It abstracts various policies for concurrency, I/O event dispatching, protocol parsing, connection management, caching, logging and service redundancy, etc.

This paper is organized as follows; Section 2 overviews related work. Section 3 overviews the natural immune system to explain why our work uses its metaphors. Section 4 describes the architecture and mechanism of iNexus policy negotiation engine and its learning capability. Section 5 illustrates some empirical simulation results. In Sections 6

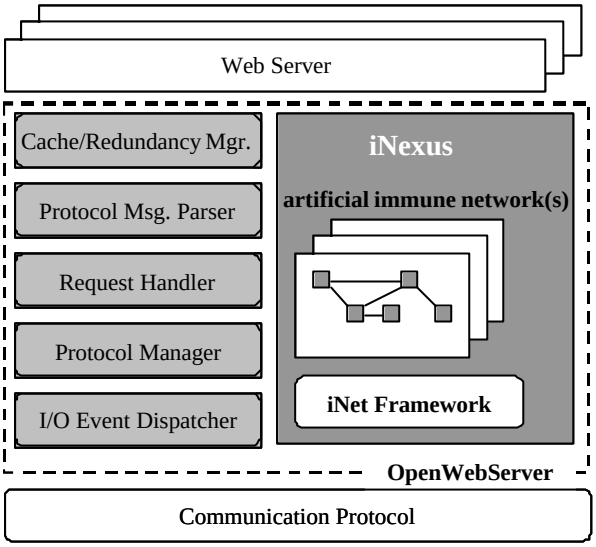


Figure 1: OpenWebServer/iNexus architecture

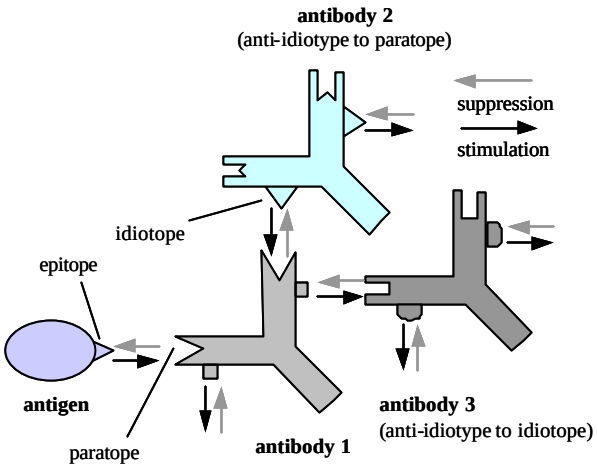


Figure 2: Interactions in the immune network. The immune response to an antigen results in the formation of anti-idiotype antibodies specific to the individual idiotope of the primary antibody (antibody 1). These anti-idiotype antibodies (antibody 2 and 3) in turn induce the formation of anti-anti-idiotype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

and 7, we conclude with a note on the current status of our project and present future work.

2. Related Work

Software adaptation has been studied by various research efforts such as reflection, open implementation, adaptive programming, aspect-oriented programming, component composition, and collaboration-based design, as well as quality of service (QoS) policy management in the networking community. In every approach, there is an entity representing a policy. For example, it is called *metaobject* in the context of reflection, *concern* in the principle of separation of concern, *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.

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 are not orthogonal with each other in many cases, but have complex constraints. Most of the above research efforts have not addressed the autonomous policy negotiation, i.e. the process for 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.

Our work contributes to propose an autonomous policy negotiation in a communication system. The negotiation process is 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 increases the flexibility and scalability of policy management. iNexus allows any policy to be introduced, altered and removed dynamically.

In terms of the network QoS research, our work is categorized in the application-level QoS policy management within a communication endsystem. We do not discuss QoS enforcement. The level of our policy guarantee is currently best-effort. In terms of reflection and aspect-oriented programming, our work addresses the mechanisms of autonomous metaobject composition and aspect weaving, respectively.

3. Natural Immune System and Immune Network

The structure and behavior of iNexus is designed with the metaphors in the natural immune system, particularly immune network. This section describes their primary mechanism.

Immune System

The natural immune system is a subject of great research interests because it provides powerful and flexible

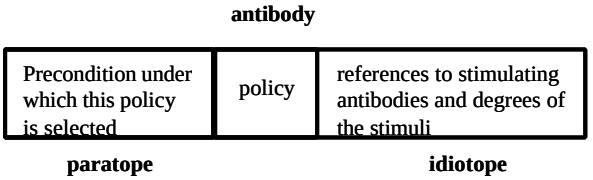


Figure 3: The antibody structure in iNexus immune

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.

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 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. 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*. This means an antibody is recognized as an antigen by another antibody.

Immune Network

Based on this fact, Jerne proposed the concept of the *immune network*, or *idiotypic network* [5], 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. Performed based on this self-regulating mechanism, the immune response 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. iNexus Policy Negotiation Engine

This section presents the architecture and mechanism of iNexus policy negotiation engine. They are modeled based on the work by Farmer et al. [6, 7] and Ishiguro et al. [8]. iNexus is developed with iNet [25], which is a framework for building artificial immune networks (see Figure 1). iNet 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. It has served as an infrastructure in our several research projects building biologically inspired multi-agent systems, where artificial immune networks optimize agents' strategies in a pursuit game and Robocup soccer game simulation. The architecture and design of iNet are described in [9].

iNexus Immune Network Model

iNexus specifies a system's current conditions as antigens, 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 (see Table 1). Policies are regarded as antibodies (see Table 1), 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. This relationship is weighted according to constraints between policies.

Figure 3 shows the antibody structure. The paratope represents a precondition under which a certain policy is selected. The iNexus immune network begins immune response when an antigen and an antibody's paratope are matched. The idiotope represents the references to other stimulating antibodies with degrees of the stimuli (or

affinities). Sample immune network is depicted in Figure X. iNexus selects a set of antibodies by calculating every antibody's current population through their interactions.

The goal of iNexus policy negotiation is that (1) it controls the system's configuration dynamically and continuously in a changing environment, and (2) its negotiation process is performed in the bottom-up, or emergent, manner through the decentralized local interactions between policies without any single point of control, as the natural immune network does

Policy Negotiation

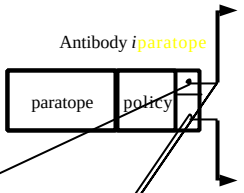
Figure 4 shows a generalized view of an antibody within in an immune network. The *i*-th antibody stimulates *M* antibodies and suppresses *N* antibodies. m_{ji} and m_{ik} denote affinities between antibody *j* and *i*, and between antibody *i* and *k*, respectively. The affinity means the degree of 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. The concentration of the *i*-th antibody, denoted by a_i, is calculated with the following equations.

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. 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 0.1 is never selected.

System Adaptation through Learning



In the previous iNexus immune network, the stimulation/suppression relationships are fixed with static affinity values [3]. This means the policy negotiation process is exactly performed as an immune network designer expected at the design time. However, this network does not provide a true dynamic feature described in Section 3. Defining static affinity values is time-consuming task. We have chose a conservative strategy for building an artificial immune network in traditional versions of iNexus, because the "trial and error"-style negotiation can result in fatal errors due to delivering a disallowed combination of policies.

For OpenWebServer to evolve effectively in an ever-changing environment, iNexus can re-arrange the immune network structure at run-time by changing affinity values.

They are modified with the reword and penalty reinforcement signals as shown in the equation 3 and 4, either when concentrations of two arbitrary antibodies exceed the predefined threshold (0.7) during the 30 calculation steps described in the previous section, or when one or more antigens stimulate two antibodies simultaneously. This means iNexus immune network learns from results of its own behaviors. Note that this learning mechanism can even work under the situation where the immune network is not structured, i.e. the idiotope of every antibody is initially blank. We expect iNexus to provide the emergence of the knowledge of policy combination.

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

5. Biologically-inspired Policy Negotiation in OpenWebServer/iNexus

This section presents policies supported in OpenWebServer/iNexus and describes our policy negotiation mechanism augmented by an artificial immune network. Then, we illustrates some empirical simulation results.

OpenWebServer Policies

OpenWebServer/iNexus supports the policies listed in Table 1.

We can produce high-throughput, highly available, fault tolerant or minimum footprint servers by tuning the combination of these policies dynamically or statically [10].

The key determinants of web server throughput are concurrency, I/O event dispatching, and file access policies. 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. 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.

Figure 5 shows how flexible nature of the OpenWebServer/iNexus enables it to adapt from its baseline performance to stable high-throughput performance. It demonstrates it is possible to improve server performance through superior server design. A flexible server framework like OpenWebServer need necessarily not perform poorly, while a hard-coded server can provide excellent performance. We achieved this performance through systematic benchmarking of different configurations of OpenWebServer under different server load conditions. We then selected the combination of features that yielded the best overall performance. Our benchmark is conducted with WebStone 2.5, running Sun's Java 1.2 VM on a Windows NT Server 4.0 SP5 (400Mhz 256 MB RAM) with idle 10Mbps Ethernet connection.

iNexus Policy Negotiation

iNexus provides OpenWebServer an autonomous policy negotiation continuously that provides an optimized combination of policies as described in the previous section. We built an artificial immune network incorporating 11 policies listed in Table 1. It defines 39 stimulation/suppression relationships.

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

[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.

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

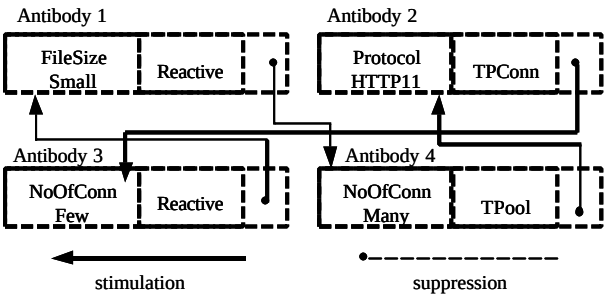


Figure 4: A sample immune network

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.

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.

Figure 5 shows a continuous performance transition of OpenWebServer/iNexus using an artificial immune network. It receives HTTP requests from 30 simultaneous connections during 30 minutes, and then from 60 connections after that. In this change of system condition, OpenWebServer/iNexus dynamically re-configures the concurrency policy from single-threaded reactive to thread pool. This autonomous policy decision allows the server to increase its throughput from 30% to 90%.

6. Future Work

This paper focuses on the policy coordination of a critical determinant to the HTTP server performance: concurrency. We are testing our artificial immune network with more complex experiments using greater number of antibodies. Our latest immune network includes new 12 policies regarding sever redundancy, application-level service

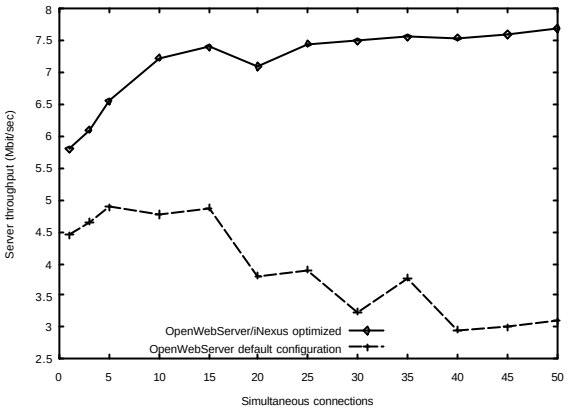


Figure 5: Comparative performance for OpenWebServer

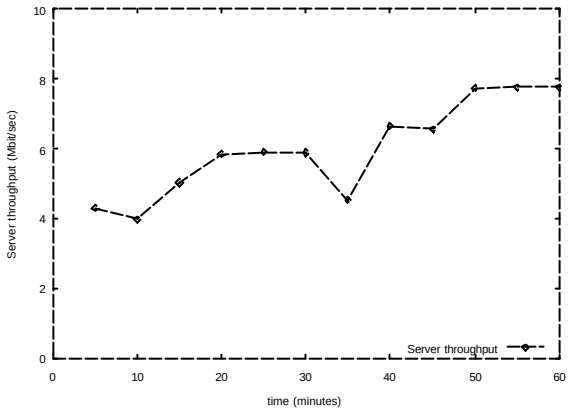


Figure 5: OpenWebServer/iNexus performance

support (e.g. CGI, Servlet and Java Server Pages), logging, and protocol pipeline-parsing.

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 [12]. 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.

7. Conclusion

This paper describes our policy negotiation facility for communication software augmented by an artificial immune network. It 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 an autonomous and

decentralized coordination mechanism as a next logical extension to existing adaptive and QoS-enabled systems.

8. References

[1] 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.

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

[3] J. Suzuki and Y. Yamamoto. Decentralized Policy Coordination Facility in OpenWebServer. In *Proceedings of SPA'00*, March 2000.

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

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

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

[7] 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.

[8] 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.

[9] J. Suzuki and Y. Yamamoto, iNet: An Extensible Framework for Simulating Immune Network. In *Proceedings of IEEE SMC'00*, 2000. to appear.

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

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

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