IEICE TRANS. INF. & SYST., VOL.E84–D, NO.12 DECEMBER 2001

PAPER

Biologically-Inspired Autonomous Adaptability in a Communication Endsystem: An Approach Using an Artificial Immune Network*

Junichi SUZUKI†, *Nonmember and* **Yoshikazu YAMAMOTO**††, *Regular Member*

SUMMARY This paper describes the adaptability of communication software through a biologically-inspired policy coordination. Many research efforts have developed adaptable systems that allow various users or applications to meet their specific requirements by configuring different design and optimization policies. Navigating through many policies manually, however, is tedious and error-prone. Developers face the significant manual and ad-hoc work of engineering an system. In contrast, we propose to provide autonomous adaptability in communication endsystem with OpenWebServer/iNexus, which is both a web server and an object-oriented framework to tailer various web services and applications. The OpenWebServer's modular architecture allows to abstract and maintain a wide range of aspects in a HTTP server, and reconfigure the system by adding, deleting, changing, or replacing their policies. iNexus is a tool for automated policy-based management of OpenWebServer. Its design is inspired by the natural immune system, particularly immune network, a truly autonomous decentralized system. iNexus inspects the current system condition of OpenWebServer periodically, measures the delivered quality of service, and selects suitable set of policies to reconfigure the system dynamically by relaxing constraints between them. The policy coordination process is performed through decentralized interactions among policies without a single point of control, as the natural immune system does. This paper discusses communication software can evolve continuously in the piecemeal way with biological concepts and mechanisms, adapting itself to ever-changing environment.

key words: *system adaptability, self-configuring system, reflective system, artificial immune system, Internet server*

1. Introduction

The communication system like web servers and middleware has emerged as an important architectural component in building electronic commerce. However, the design, optimization and deployment of communication system are still hard while computing power and network bandwidth have increased dramatically. It has a remarkably rich set of options, or policies, and no single policy fits all applications or environments. For example, available policies for concurrency and I/O event dispatching depend on the type of underlying OS and networking facility, and their optimal policies vary with system's current workload. A system's preferred set of policies also varies with its goal, current system condition and application requirements. For instance, different set of policies for request processing, caching, logging and data transfer should be chosen depending on whether deploying a high performance server on a powerful workstation or an embedded server on a network appliance with limited available resources.

Therefore, it is essential for communication system to have an open-ended framework that allows configuring its optimal setting out of feasible policies. The existence of all the feasible policies ensures that systems can be tailored to meet their users' or applications' requirements. It is tedious, error-prone and economically expensive, however, to understand the tradeoffs among alternative policies and navigate through them manually. Developers face the significant manual efforts of engineering a system, 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.

A key observation to the current and near future communication systems is that autonomous adaptability is a unifying theme on which networking facilities and applications can be constructed. Communication endsystems should autonomously facilitate both static adaptability (e.g. bindings of common operations to high performance mechanisms of the native OS) and dynamic adaptability (e.g. altering runtime behavior on the fly based on present load condition). The autonomous adaptability can increase system's scalability and availability as well as frees developers from manual reconfiguration work.

This paper describes OpenWebServer/iNexus, which is our research vehicle for investigating and demonstrating autonomous adaptability in communication endsystem. OpenWebServer is both an adaptive web server and an object-oriented framework for building optimally configured Internet servers [1]–[3] (see Fig. 1). It abstracts a series of aspects, e.g. concurrency, I/O event dispatching, protocol parsing, connection management, caching, logging, service redundancy, etc,

Manuscript received December 6, 2000.
Manuscript revised July 3, 2001.

†The author is with the Department of Information and Computer Science, University of California, Irvine, CA, 92697–3425, USA.
††The author is with the Department of Information and Computer Science, Keio University, Yokohama-shi, 223–8522 Japan.

*This work has been supported in part by the Japan Society for the Promotion of Science (Fundamental research (C)(2) 12680359).

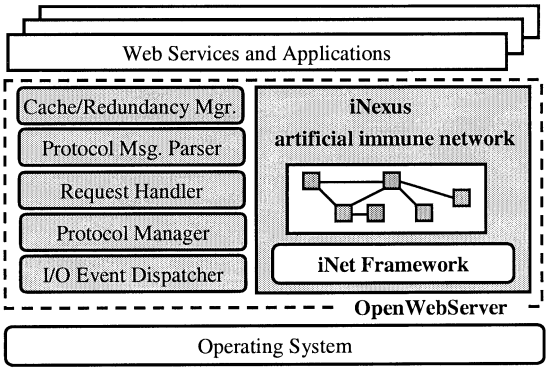

Fig. 1 OpenWebServer/iNexus architecture.

and maintains various policies for them. iNexus is an autonomous policy coordination facility embedded in OpenWebServer (Fig. 1). It allows OpenWebServer to continuously adapt itself to a changing environment by determining which policy should be changed and how policies are re-combined so that it can keep performing well [4], [5]. iNexus is designed as a self-organising facility inspired by a biological system, because the biological system is truly adaptive and scalable. We believe that communication system should have the adaptability feature through autonomous system reconfiguration as a built-in mechanism, and that the self-adaptation can be overcome by adopting key biological principles and mechanisms. The structure and behavior of iNexus are modeled with the principles and mechanisms in the natural immune system. In iNexus, the policy coordination process is performed through decentralized interactions among policies without a single point of control, as the natural immune system does. Our goal is to demonstrate that a communication endsystem augmented by an artificial immune system can effectively evolve in an ever-changing environment in the autonomous manner.

The remainder of this paper is organized as follows; Section 2 compares with existing related work. Section 3 overviews the natural immune system. Section 4 describes the architecture and mechanism of OpenWebServer/iNexus. Section 5 illustrates some experiment results. In Sects. 6 and 7, we conclude with a note on the current status of our project and future work.

2. Related Work

Software adaptation has been studied by various research efforts such as computational reflection [6], open implementation [7], adaptive programming [8], aspect-oriented programming [9], component composition [10], and collaboration-based design [11]. In every approach, there is an entity representing each policy. For example, it is called *metaobject* in the context of reflection, *concern* in the principle of separation of concern [12], *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. In OpenWebServer/iNexus, such an entity is called *metaobject*. In terms of reflection and aspect-oriented programming, this paper addresses the process to determine a strategy composing metaobjects and weaving aspects, respectively.

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 often not orthogonal with each other, but have complex constraints. Most of the above research efforts have not addressed the autonomous policy coordination, i.e. the process for inspecting the dependencies between policies and then resolving co-use/conflict constraints to produce an better combination of feasible policies. The simplest and dominant coordination strategy is writing a long sequence of hard-written if/case statements in a program. Another strategy is using the multiple inheritance in an object oriented language. Both suffer from combination explosion of policies, and cost lots of labor for configuring if/case statements or inheritance relationships. They are also fragile for changing policy specifications. In existing work, it is often ad-hoc and has not been addressed at large how to coordinate policies consistently throughout the system's lifetime. OpenWebServer/iNexus provides a flexible way to manage a wide range of policies so that adding and deleting policies do not affect the other policies. It also determines the most appropriate set of policies suited to a given situation in the dynamic and autonomous fashion by relaxing constraints between them.

Compared with quality of service (QoS) management in the field of computer network [13], our work focuses on the application-level QoS policy coordination within a communication endsystem. We do not address QoS in the transport/network level.

As for the adaptability of HTTP server, JAWS [14] allows developers to choose policies in order to deploy high performance servers. However, JAWS cannot coordinate different competitive policies autonomously; it is the responsibility for developers to do that manually. In contrast, OpenWebServer/iNexus eliminates the chaotic manual reconfiguration work from developers through autonomous adaptation.

Researchers in the fields of artificial life [15], [16] and complexity [17] have studied large scale biological systems and the behaviors of simple entities within those systems. The work has been concentrated on imitating life in a computer and understanding the basic processes. OpenWebServer/iNexus applies the findings of those research efforts, particularly in artificial immune system, to a new domain: the design of communication endsystem that enables the construction and deployment of adaptive, scalable and available network

applications. Our work is not the first to use metaphors in the immune system. [18] and [19] used the immune system as a model for network security and intrusion detection. While their efforts parallel ours, OpenWebServer/iNexus focuses on policy coordination to control the endsystem's configuration.

3. Overview of Natural Immune System

The immune system is a subject of great research interests because it provides powerful and flexible information processing capabilities as a decentralized intelligent system. It has some important computational aspects such as self/non-self discrimination, learning, memory, retrieval and pattern matching. There exists several theoretical and software models to explain immunological phenomena. They have been used for machine learning, computer security, fault detection, change management, image processing, searching and robot navigation [20].

The immune system can discriminate between foreign molecules (i.e. non-self) and the body's own cells and proteins (i.e. self). Once non-self is recognized, the immune system enlists the participation of a variety of cells and molecules to mount an appropriate response in order to eliminate or neutralize it. The immune response involves *antigen-presenting cells*, *lymphocytes* and *antibodies*. Lymphocytes are one of many types of white blood cells, and two major population of lymphocytes are *B lymphocytes* (or B cells) and *T lymphocytes* (or T cells). B cells have receptors on their surface, which can recognize antigens invading a human body, e.g. viruses, and then produce antibodies specific to the recognized antigen. The key portion of antigen that is recognized by the antibody is called *epitope*, which is the antigen determinant (see Fig. 2). *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. Each type of antibody has its own antigenic determinant, called *idiotope*. This means an antibody is recognized as an antigen by another antibodies.

Based on this fact, Jerne proposed the concept of the *immune network*, or idiotypic network [21], which states that antibodies and lymphocytes are not isolated, but they are communicating with each other (Fig. 2). 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

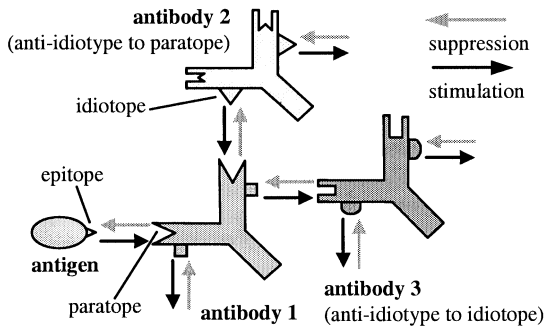

Fig. 2 Interactions in the immune network.

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 structure of immune network is not fixed, but varies continuously according to dynamic changes of environment. The immune system maintains an appropriate set of cells so that the system can adapt to environmental changes in the piecemeal way.

4. OpenWebServer/iNexus

This section describes the design of OpenWebServer kernel and iNexus artificial immune network. Both OpenWebServer and iNexus are developed with Java.

4.1 OpenWebServer Kernel Architecture

OpenWebServer is an object-oriented framework to tailor Internet services or applications, which employs a reflective meta-architecture [6]. It contains a metalevel(s) that specifies a wide range of aspects, i.e. structure and/or behavior, of web servers using fine-grained metaobjects. The system's configuration and behavior can be altered by adding, deleting, changing or replacing metaobjects. The kernel components are organized as shown in Fig. 3.

`SysController` starts the system by creating appropriate metalevels and metaobjects. This object is also responsible for stopping and resuming the system. `MetaSpace` represents a metalevel. It references every metaobject. `Baseobject` is the root object that all the components implementing a service logic or application derives from. Baseobjects can access `MetaSpace` when communicating with their metalevel. `MetaObject` specifies the interfaces for every metaobject. `MetaObjectImpl` and its subclasses provide the implementations of a metaobject. Each metaobject has one or more implementations that perform the same functions but provide different computational algorithms, or policies (Fig. 4). To specify metaobjects,

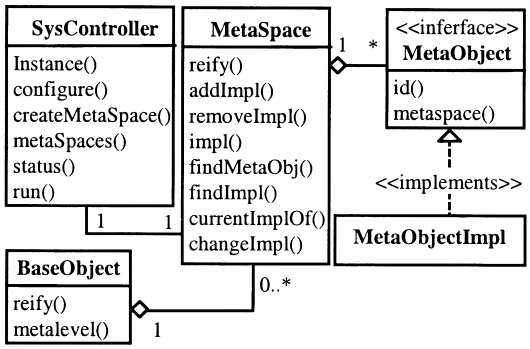

Fig. 3 OpenWebServer kernel components.

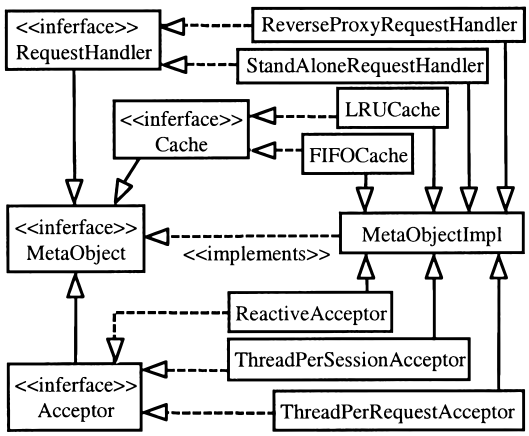

Fig. 4 Metaobjects and their implementations.

we identified services and entities by looking for typical events or constructs found during the execution of web servers. Abstracting these events and constructs, OpenWebServer provides the following metaobjects and their implementations by default:

- `Initializer` initializes and reconfigures the system. Its implementations include `StaticInitializer`, which configures the system when it starts, and `DynamicInitializer`, which can reconfigure the system at runtime.
- `IO` defines the I/O dispatching policy. Its implementations include `SyncIO` and `AsynchIO`.
- `Acceptor` waits for and accepts incoming requests. After accepted, a request is processed by `RequestHandler`. `ThreadPerRequestAcceptor`, `ThreadPerConnectionAcceptor`, and `ReactiveAcceptor` are its implementations (Fig. 4). They vary concurrency policy.
- `RequestHandler` encapsulates the different policies for interpreting a request. Its implementations include `StandAloneRequestHandler` and `ReverseProxyRrequestHandler`, which redirects incoming requests to the appropriate back-end replicated server by rewriting the original URL and HTTP headers (Fig. 4).
- `Protocol` defines protocol specific information and provides operations to parse its header and contents. `HTTP10`, `HTTP11`, `HTTP11Mux`, and `SOAP` are the default set of implementations.
- `ContentFinder` finds and obtains a target resource. Its implementations are `HTMLContentFinder` and `XMLContentFinder`.
- `Cache` defines caching policy. Its implementations include `LRUCache` and `FIFOCache` (Fig. 4).
- `Logger` defines logging policy. Its implementations include `OnDemandLogger`, which performs a logging operation whenever a request is processed, and `OnCommandLogger`, which performs it in every certain period.
- `Connection` defines the connection retention policy. Its implementations include `TransientConnection` and `PersistentConnection`, which is a long-lived session allowing multiple requests to be sent over the same connection.
- `Redirector` defines the policy for redirecting requests to a back-end replicated server(s). Its implementations include `RandomRedirector`, `PriorityDrivenRedirector`, and `LatencyDrivenRedirector`.
- `ExecManager` provides interfaces to execute external entities like CGI scripts, Servlet and Java Server Pages.

As depicted in Fig. 4, `MetaSpace` aggregates all the metaobject implementations so that it can change the metaobject's behavior dynamically. For example, the concurrency policy can be changed by replacing `Accepter`'s implementation. Each metaobject knows only `MetaSpace`, not any other metaobject, thereby reducing the number of interconnections. This means that both a metaobject and its implementations should be independently extensible; changes in an implementation does not have no impact on clients of its interface. `MetaSpace` facilitates loose coupling among metaobjects by acting as an intermediary. The modular architecture allows OpenWebServer to maintain a wide range of fine-grained metaobjects and to change their implementations dynamically. The detail design of OpenWebServer is described in [1].

4.2 iNexus Design

iNexus is a tool for automated policy-based management of Web services and applications. It collects performance and resource usage data of OpenWebServer in every certain period, measures the delivered quality of service, and selects suitable set of policies to reconfigure the system dynamically. In the selection process, iNexus coordinates a series of policies, each of which is represented by a metaobject implementation object, by relaxing the constraints among them. As described earlier, iNexus applies the self-adaptation and self-regulation mechanisms in the natural immune

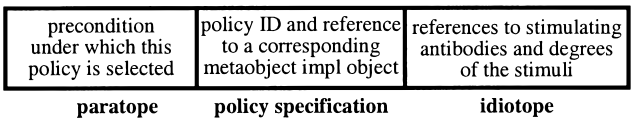

Fig. 5 Antibody structure in the iNexus immune network.

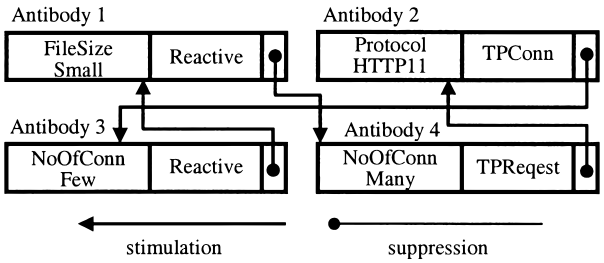

Fig. 6 A sample iNexus immune network.

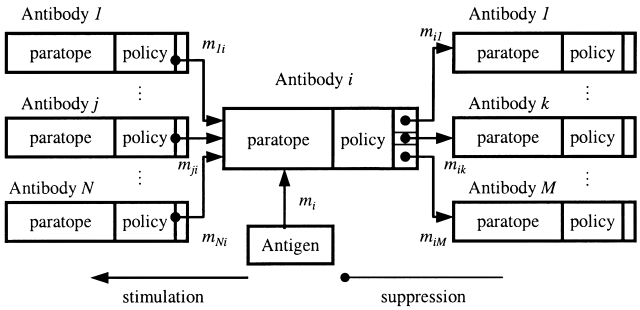

Fig. 7 A generalized view of iNexus immune networks.

system. Every policy and relationship between policies are specified in the iNexus artificial immune network. Our artificial immune network is modeled based on the work by Farmer et al. [22] and Ishiguro et al. [23]. iNexus is developed with iNet, which is an object-oriented reusable framework for building artificial immune networks (see Fig. 1). [24] describes how to design iNexus with iNet.

iNexus specifies each system's conditions 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 policy are modeled 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). Figure 5 shows the antibody structure. The policy specification compartment contains a policy ID and reference to a metaobject implementation object that provides the policy. 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). iNexus supports a series of paratopes and policies shown in Table 1 in order to create antigens and antibodies. Major IDs of paratope and policy represents categories of system condition and policy respectively. Minor IDs of paratope and policy represents specific system condition and policy. We can produce high-throughput, highly available, fault tolerant or minimum footprint servers by tuning the combination of these antibodies (i.e. policies) dynamically or statically. Antibodies are linked with each other based on the stimulation and suppression relationships. The relationship is weighted according to constraints between policies. When two policies must not be used together, there should not be a relationship between them. For example, the standalone request handling policy does not have a relationship with any redundancy policies.

Figure 6 shows a sample immune network. This network is used to determine a suitable concurrency policy according to a current system condition. It contains four antibodies representing three kinds of policies; single-threaded reactive, thread-per-request and thread-per-connection. Antibody 1 represents that the single-threaded reactive policy is activated when the average size of requested files is relatively small. However, the thread-per-request policy is activated if the number of simultaneous connections grows, because antibody 1 stimulates antibody 3. Inversely, the reactive policy is suppressed by the thread-per-request 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. 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 policy, 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 7 shows a generalized view of an antibody within an immune network. The antibody i 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. The antibody population is represented by the concept of concentration. The concentration of antibody i, denoted by a_i, is calculated with the following equations.

Table 1 Supported types of antibodies in OpenWebServer/iNexus.

Paratope major ID	Paratope minor ID	Policy major ID	Policy minor ID
FILE_SIZE	L, M, S	CONCURRENCY	REACTIVE
NO_OF_CONNECTIONS	M, A, F		THREAD_PER_REQUEST
NO_OF_CPU	M, S		ACTIVE_THREAD_POOL
OS_THREAD_SUPPORT	T, F		PASSIVE_THREAD_POOL
AVAILABLE_THREADS	M, A, F		THREAD_PER_CONNECTION
SUPPORTED_PROTOCOL	HTTP10, HTTP11	IO	SYNCH
SLAVE_HOSTS_AVAILABLE	T, F		ASYNCH
NO_OF_THREADS	M, A, F	CACHING	LRU
			FIFO
		LOGGING	ON_DEMAND
			ON_COMMAND
		CONNECTION	TRANSIENT
			PERSISTENT
		PROTOCOL	HTTP10
			HTTP11
		REQUEST_HANDLING	STANDALONE
			REDIRECT
		REDUNDANCY	RANDOM
			PRIORITY_DRIVEN
			LATENCY_DRIVEN

$$A_i(t) = A_i(t-1) + \left(\frac{\sum_{j=1}^{N}(m_{ji}a_j(t-1))}{N} - \frac{\sum_{k=1}^{M}(m_{ik}a_k(t-1))}{M} + m_i - d \right) a_{i(t-1)}\Delta t \quad (1)$$

$$a_i(t) = \frac{1}{1+\exp(0.5 - A_i(t))} \quad (2)$$

In Eq. (1), the first and second terms in a big bracket 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. m_i is 1 when antibody i is stimulated directly by an antigen, otherwise 0. d 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. Equation (2) 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. This times are adopted from existing simulation experiences [23]. 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* selection. 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. An antibody whose concentration is below 0.1 is never selected.

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 reward and penalty reinforcement signals as shown in the Eq. (3), either when concentrations of two arbitrary antibodies exceed the predefined threshold (0.7) during the 30 calculation steps described earlier, or when one or more antigens stimulate two antibodies simultaneously. In these equations, $T_p^{A_{b_i}}$ and $T_r^{A_{b_i}}$ denotes the number of times of receiving penalty and reward signals when A_{b_i} is selected. $T_{A_{b_j}}^{A_{b_i}}$ denotes the number of times when both A_{b_i} and A_{b_j} have stimulated by specific antigens. This mechanism allows an iNexus artificial immune network to learn from results of OpenWebServer's behavior. 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.

$$m_{ij} = \frac{T_p^{A_{b_i}} + T_r^{A_{b_j}}}{T_{A_{b_j}}^{A_{b_i}}} \quad (3)$$

5. Results of Adaptation Experiments

We have conducted some experiments to validate our policy coordination method in OpenWebServer/iNexus. We built an artificial immune network incorporating 20 paratopes and 20 policies listed in Table 1. It defines 76 antibodies and 146 stimulation/suppression relationships. In this setting, we conducted three kinds of experiments: static adaptability test, benchmarking test of different configurations, and dynamic adaptability test. The initial conditions in all the experiments were same. Every relationship was manually configured with initial affinity (m_{ij}) of 0.5.

In our first experiment, we evaluated the system's static adaptability by observing which antibodies iNexus selects according to an underlying system platform when OpenWebServer starts up. For example, when OpenWebServer is executed on a uni-processor

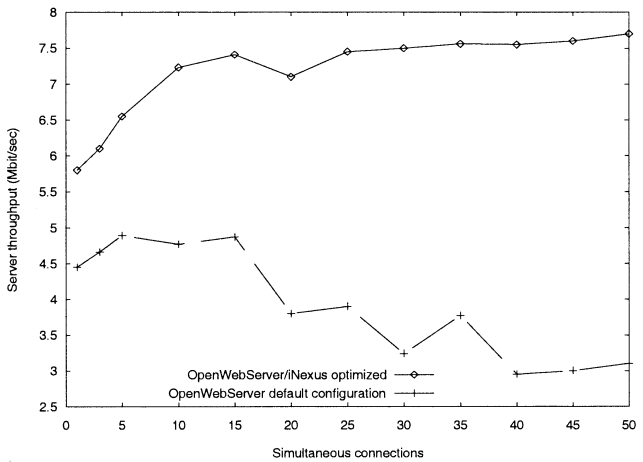

Fig. 8 Comparative performance for OpenWebServer.

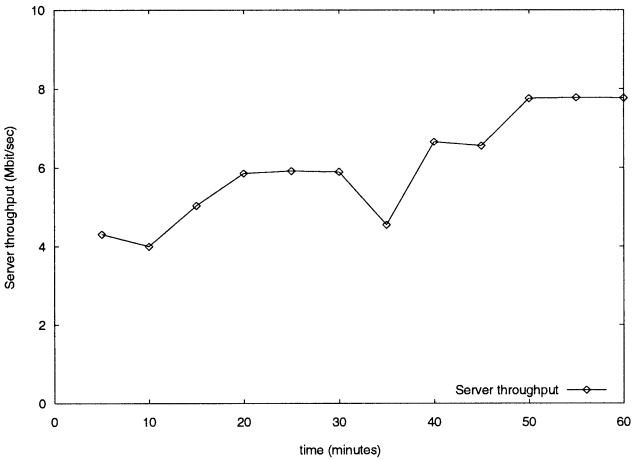

Fig. 9 OpenWebServer/iNexus performance transition.

platform running an operating system that supports threads and asynchronous I/O, iNexus chose the single-threaded reactive policy with synchronous I/O at first, instead of an multi-threaded policy with asynchronous I/O. Our artificial immune network is configured to select policies conservatively, and scale to more sophisticated ones once the current policies are not appropriate. In a multi-processor platform, the multi-threaded 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 threading, the single-threaded reactive policy was selected independently of the underlying CPU architecture. With several experiments in addition to the above ones, iNexus demonstrated it can select reasonable antibodies statically at the system's startup time.

In the second experiment, we conducted a comparative performance benchmarking experiment using different set of configurations. Figure 8 depicts how OpenWebServer performs well in terms of server throughput as the number of simultaneous connections increases from 1 to 50. We conducted systematic benchmarking of different configurations of OpenWebServer under different server load conditions. Then, we selected the combination of features that yielded the best overall performance. This result shows how flexible nature of 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. Our benchmark is conducted with the standard setting of WebStone 2.5, running a Java 1.2.3 VM (from Sun Microsystems, Inc.) on a Microsoft Windows NT Server 4.0 SP5 (Intel Pentium II 400 MHz MMX, 256 MB RAM) with idle 10 Mbps Ethernet connection. This testbed configuration was also used for the third experiment.

The third experiment examines system's autonomous adaptability by observing how iNexus changes antibodies (i.e. policies) dynamically during the system's execution. Figure 9 shows a continuous performance transition of OpenWebServer/iNexus using an artificial immune network. The server started running in a default configuration (see Fig. 8). OpenWebServer/iNexus reconfigured its own policies in every 5 minutes; reconfiguration occurred 12 times. It receives HTTP requests through 30 simultaneous connections during the first 30 minutes, and then through 60 connections during the last 30 minutes. Figure 9 shows OpenWebServer/iNexus dynamically reconfigured itself to adapt the change of system environment. The autonomous policy coordination allows the server to increase its throughput from 30% to 90%. The average error rate (errors/sec) was stable after the number of connections increased (0.008 in 30 connections and 0.012 in 60 connections). Note that the coordination process is not performed by a single coordinator, but through decentralized interactions among antibodies.

6. Current and Future Work

OpenWebServer/iNexus has been used as foundation in a web content personalization service [25] and distributed collaborative environment [26]. We plan further experiments using real world applications.

This paper focuses on the policy coordination for critical determinants to HTTP server performance in order to configure OpenWebServer to be a high-throughput server. We plan to prepare different kinds of paratopes and policies so that we can tune OpenWebServer as a embedded thin server.

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 immunolo-

gists. iNexus currently uses a cyclic network model. We are now evaluating other network models in detail. Also, we plan to incorporate some additional immunological concept, e.g. tolerance and immune memory.

7. Conclusion

This paper describes our biologically-inspired approach to design communication system that can autonomously adapt to a changing environment in the piecemeal manner. OpenWebServer/iNexus supports autonomous policy coordination and system reconfiguration as built-in mechanisms. Augmented by an artificial immune network, it can select a suitable set of policies through decentralized interactions among them. We believe our work provides a blue print showing an autonomous adaptation mechanism as a next logical extension to existing adaptable systems.

References

[1] J. Suzuki and Y. Yamamoto, "OpenWebServer: An adaptive web server using software patterns," IEEE Communications, vol.37, no.4, pp.46–52, April 1999.
[2] J. Suzuki and Y. Yamamoto, "Dynamic adaptation in the web server design space using OpenWebServer," Proc. 2nd JSSST Systems for Programming and Applications, March 1999.
[3] J. Suzuki and Y. Yamamoto, "Building an adaptive web server with a meta-architecture: AISF approach," Proc. 1st JSSST Systems for Programming and Applications, March 1998.
[4] J. Suzuki and Y. Yamamoto, "A decentralized policy coordination facility in OpenWebServer," Proc. 3rd JSSST Systems for Programming and Applications, 2000.
[5] J. Suzuki and Y. Yamamoto, "Building an artificial immune network for decentralized policy negotiation in a communication endsystem: OpenWebServer/iNexus study," Proc. 4th World Multiconferences on Systemics, Cybernetics and Informatics, July 2000.
[6] G. Kiczales, J. Rivieres, and D.G. Bobrow, The Art of the Metaobject Protocol, MIT Press, Cambridge, MA, 1991.
[7] G. Kiczales, "Beyond the black box: Open implementation," IEEE Software, vol.13, no.1, pp.8–11, 1996.
[8] K.J. Lieberherr, The Art of Growing Adaptive Object-Oriented Software, PWS Publishing Company, 1995.
[9] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda C. Lopes, J.-M. Loingtier, and J. Irwin, "Aspect-oriented programming," Proc. ECOOP'97, 1997.
[10] M. Aksit, K. Wakita, J. Bosch, L. Bergmans, and A. Yonezawa, "Abstracting object-interactions using composition filters," in Object-based Distributed Processing, ed. R. Guerraoui, et al., 1993.
[11] M. Mezini and K. Lieberherr, "Adaptive plug-and-play components for evolutionary software development," Proc. Object-Oriented Programming, Systems and Languages, Oct. 1998.
[12] W.L. Hursch and C.V. Lopes, Separation of Concerns, TR-NU-CCS-95-03, Northeastern University, 1995.
[13] T.A. Campbell, "QoS architectures," in Multimedia Communications Networks, Chapter 3, ed. M. Tatipamula and B. Khasnabish, Artech House, 1998.
[14] D.C. Schmidt and J. Hu, "Developing flexible and high-performance web servers with frameworks and patterns," ACM Computing Surveys, May 1998.
[15] R. Collins and D. Jeffereson, "AntFarm: Towards simulated evolution," Proc. Artificial Life II, 1992.
[16] M. Millonas, "Swarms, phase transitions, and collective intelligence," Proc. Artificial Life III, 1994.
[17] S. Kauffman, At Home in the Universe, Oxford University Press, 1995.
[18] J. Kephart, "A biologically inspired immune system for computers," Proc. Artificial Life IV, 1994.
[19] P. D'haeseller, S. Forrest, and P. Helman, "An immunological approach to change detection," Proc. IEEE Symposium on Security and Privacy, 1996.
[20] D. Dasgupta, ed., Artificial Immune Systems and Their Applications, Springer, 1999.
[21] N.K. Jerne, "Idiotypic networks and other preconceived ideas," Immunological Review, vol.79, no.5, 1984.
[22] J.D. Farmer, N.H. Packard, and A.S. Perelson, "The immune system, adaptation, and machine learning," Physica, D 22, vol.184, no.204, 1986.
[23] A. Ishiguro, T. Kondo, Y. Watanabe, and Y. Uchikawa, "An immunological approach to behavior arbitration for autonomous mobile robots," Proc. International Symposium on Artificial Life and Robotics, 1996.
[24] J. Suzuki and Y. Yamamoto, "iNet: An extensible framework for simulating immune," Proc. IEEE International Conference on Systems, Man. & Cybern. 2000, Oct. 2000.
[25] J. Suzuki and Y. Yamamoto, "Document brokering with agents: Persona approach," Proc. 1998 Workshop on Interactive System and Software, Dec. 1998.
[26] J. Suzuki and Y. Yamamoto, "Leveraging distributed software development," IEEE Computer, vol.32, no.9, pp.59–65, 1999.

Junichi Suzuki received B.S., M.S. and Ph.D. degrees in computer science from Keio University, Japan. He is currently a post doctoral researcher at University of California, Irvine. His research interests include highly distributed systems, adaptive communication systems, autonomous decentralized agent computing, and biologically-inspired software architecture. He is a member of ACM, IEEE-CS, IPSJ and JSSST.

Yoshikazu Yamamoto received B.S., M.S. and Ph.D. degrees in administration engineering from Keio University, Tokyo. He is currently an associate professor of Department of Computer and Information Science at Keio University. He worked at Linkoping University, Sweden as a visiting professor from 1981 to 1983. His current research interests include distributed discrete event simulation and modeling, OOP, agent programming, intelligent interface and documentation. He is a member of ACM, IEEE-CS, IPSJ and also councilor board of JSSST.