Preface

Technology is taking us to a world where myriads of heavily networked devices interact with the physical world in multiple ways, and at multiple scales, from the global Internet scale down to micro and nano devices. Many of these devices are highly mobile and autonomous, and must adapt to the surrounding environment in a totally unsupervised way.

This vision poses severe challenges to existing approaches to the design and management of ICT systems, since the size of this omni-comprehensive network, in terms of both number of constituent nodes and running services, is expected to exceed by several orders of magnitude that of existing information systems. The resulting large-scale communication system could not be handled according to conventional ICT paradigms, which are not able to accommodate the scale, heterogeneity and complexity of such scenarios. A fundamental research challenge is therefore the design of *robust decentralized computing systems* capable of operating under changing environments and noisy input, and yet exhibit the desired behavior and response time, under constraints such as energy consumption, size, and processing power. These systems should be able to adapt on the short term and evolve on the long one, and they should learn how to react to unforeseen scenarios as well as to display properties comparable to social entities.

Biological systems are able to handle many of these challenges with an elegance and efficiency still far beyond current human artifacts. Most of the desired features have been refined by nature over periods of millions of years, generating living organisms that are able to autonomously repair themselves when damaged, produce emergent behavior, survive despite drastic changes in the environment conditions and evolve over time. All these considerations generated a significant interest in the area of *bio-inspired computing*, i.e. the application of biological principles to the design of human artifacts, with the expectation of reproducing the same observed behavior.

Fully in line with this growing interests in biologically inspired computing, the first international conference on *Bio Inspired mOdels of NEtwork, Information and Computing Systems* (BIONETICS) was successfully organized in Cavalese in December 2006. The aim was to bring together researchers and scientists from

several disciplines in computer science and engineering where bio-inspired methods are investigated. The interest in this new research approach has been confirmed by the high number of high quality submissions, and from the high number of conference attendees.

This book collects extended versions of selected outstanding papers originally submitted to BIONETICS 2006. It is structured into four parts, covering different aspects involved in the engineering of future ICT systems:

- Self-Organizing Network Environments
- System Design and Programming
- Sensor and Actor Networks
- Search and Optimization

With this book, we want to address researchers who are interested in bio-inspired computing. According to the broad spectrum addressed by the different book chapters, a variety of biological principles and their application in ICT systems are presented. Based on these information, we want to encourage researchers to initiate interdisciplinary studies especially focusing on biological disciplines.

Erlangen and Trento,   *Falko Dressler*
March 2007   *Iacopo Carreras*

Towards a Biologically-inspired Architecture for Self-Regulatory and Evolvable Network Applications

Chonho Lee, Hiroshi Wada and Junichi Suzuki

Department of Computer Science
University of Massachusetts, Boston
{chonho, shu, jxs}@cs.umb.edu

Summary The BEYOND architecture applies biological principles and mechanisms to design network applications that autonomously adapt to dynamic environmental changes in the network. In BEYOND, each network application consists of distributed software agents, analogous to a bee colony (application) consisting of multiple bees (agents). Each agent provides a particular functionality of a network application, and implements biological behaviors such as energy exchange, migration, reproduction and replication. This paper describes two key components in BEYOND: (1) a self-regulatory and evolutionary adaptation mechanism for agents, called iNet, and (2) an agent development environment, called BEYONDwork. iNet is designed after the mechanisms behind how the immune system detects antigens (e.g., viruses) and produces antibodies to eliminate them. It models a set of environment conditions (e.g., network traffic) as an antigen and an agent behavior (e.g., migration) as an antibody. iNet allows each agent to autonomously sense its surrounding environment conditions (i.e., antigens) and adaptively invoke a behavior (i.e., antibody) suitable for the conditions. In iNet, a configuration of antibodies is encoded as a gene. Agents evolve their antibodies so that they can adapt to unexpected environmental changes. iNet also allows each agent to detect its own deficiencies to detect antigen invasions (i.e., environmental changes) and regulate its policy for antigen detection. Simulation results show that agents adapt to changing network environments by self-regulating their antigen detection and evolving their antibodies through generations. BEYONDwork provides visual and textual languages to design agents in an intuitive manner.

1 Introduction

Large-scale network applications such as data center applications and grid computing applications face several critical challenges, particularly *autonomy* and *adaptability*, as they have been increasing in complexity and scale[1]. They are expected to autonomously adapt to dynamic environmental changes in the network (e.g., workload surges and resource extinction) in order to improve user experience, expand applications' operational longevity and reduce maintenance cost [3, 4, 5, 6].

---

[1] For example, Google, Inc. reportedly runs over 450,000 servers in its data centers [1, 2].

Based on an observation that various biological systems have developed the mechanisms necessary to meet the above challenges, the proposed architecture, called BEYOND[2], applies key biological principles and mechanisms to design autonomous and adaptive network applications. In BEYOND, each application is designed as a decentralized group of software agents. This is analogous to a bee colony (application) consisting of multiple bees (agents). Each agent provides a particular functionality of a network application, and implements biological behaviors such as energy exchange, migration, replication, reproduction and death.

This paper focuses on two key components in BEYOND: (1) a self-regulatory and evolutionary adaptation mechanism for agents, called iNet, and (2) an agent development environment, called BEYONDwork. iNet is designed after the mechanisms behind how the immune system detects antigens (e.g., viruses), how it specifically produces antibodies to eliminate them and how it evolves antibodies to react a massive number of antigens. iNet models a set of environment conditions (e.g., network traffic and resource availability) as an antigen and an agent behavior as an antibody. Each agent contains its own immune system (i.e., iNet), and a configuration of the agent's antibodies defines its behavior policy, which determines which behavior to be invoked in a given set of environment conditions. iNet allows each agent to autonomously sense its surrounding environment conditions (i.e., antigen) for evaluating whether it adapts well to the sensed conditions, and if it does not, adaptively invoke a behavior (i.e., antibody) suitable for the conditions. For example, agents may invoke the replication behavior at the network hosts that accept a large number of user requests for their services. This leads to the adaptation of agent availability; agents can improve their throughput. Also, agents may invoke the migration behavior to move toward the network hosts that receive a large number of user requests for their services. This results in the adaptation of agent locations; agents can improve their response time to user requests.

iNet also allows each agent to detect its own deficiencies to detect antigen invasions, i.e., deficiencies to evaluate whether it adapts well to the current environment conditions. Due to the deficiencies, some agents may invoke their behaviors when they have already adapted well to the current environment conditions. Others may invoke no behaviors when they do not adapt to the current environment conditions. With iNet, each agent can regulate its policy for antigen detection so that it can perform its behaviors at the right time. This self-regulation process is intended to avoid the degradation of agent adaptability and the waste of resource consumption incurred by unnecessary behavior invocations.

In iNet, a configuration of antibodies (i.e., behavior policy) is encoded as a gene. Agents evolve their antibody configurations so that the configurations become fine-tuned to the current and even unexpected environment conditions. This evolution process occurs via genetic operations such as mutation and crossover, which alter antibody configurations (genes) during agent reproduction and replication. Agent evolution frees agent developers from anticipating all possible environmental

---

[2] Biologically-Enhanced sYstem architecture beyond Ordinary Network Designs

changes and tuning their agents' antibodies (behavior policies) to the changes at design time. This significantly simplifies the implementation of agents.

Simulation results show that iNet allows agents to autonomously adapt to changing network environments by dynamically regulating their antigen detection and evolving their antibodies through generations.

The second focus of this paper is an application development environment for iNet, called BEYONDwork. BEYONDwork provides visual and textual languages to design agent in an intuitive and easy-to-understand manner. It accepts the visual models and textual programs built with the proposed languages, and transforms them to Java code that are compilable and runnable on a simulator for BEYOND. This code generation enables rapid development and configuration of agents, thereby improving the productivity of agent developers.

2 Design Principles in the BEYOND Architecture

In BEYOND, agents are designed based on the six principles described below.

- **Decentralization:** Inspired by biological systems (e.g., bee colony), there are no central entities to control and coordinate agents in BEYOND so that they can be scalable and simple by avoiding a single point of performance bottlenecks [7] and failures [8] and by avoiding any central coordination in deploying agents [9].
- **Autonomy:** Similar to biological entities (e.g., bees), agents sense their local network environments, and based on the sensed environment conditions, they autonomously behave and interact with each other without any intervention from/to other agents and human users.
- **Emergence:** In biological systems, collective (group) behaviors emerge from local interactions of autonomous entities [10]. In BEYOND, agents only interact with nearby agents. They behave according to dynamic changes in environment conditions such as user demands and resource availability. Through collective behaviors and interactions of individual agents, desirable system characteristics (e.g., load balancing and resource efficiency) emerge in a swarm of agents.
- **Lifecycle:** Biological entities strive to seek and consume food for living. In BEYOND, agents store and expend *energy* for living. Each agent gains energy in exchange for performing its service to other agents or human users, and expends energy to use network and computing resources (e.g., bandwidth and memory). The abundance or scarcity of stored energy affects agent lifecycle. For example, an energy abundance indicates high demand to an agent; thus, the agent may be designed to favor reproduction or replication to increase its availability. An energy scarcity (i.e., an indication of lack of demand) causes death of the agent.
- **Homeostasis:** Biological entities regulate their internal environments to maintain stable conditions (e.g., stable body temperature and blood fluid) even though external environments change. Similarly, in BEYOND, agents strive to maintain the fitness (or the degree of adaptation) to external network environments. When an agent finds that its fitness decreases, it adjusts its antigen detection policy so that it can keep its fitness to dynamic network environments.

- **Evolution:** Biological entities evolve as a species to increase the fitness to the environment across generations. In BEYOND, agents collectively evolve their antibody configurations (behavior policies) across generations. Agents perform this evolution process by generating behavioral diversity and executing natural selection. Behavioral diversity means that different agents possess different antibody configurations (behavior policies). This is generated via mutation and crossover during agent replication and reproduction. Natural selection retains the agents that adapt well to the environment (i.e., the agents that have beneficial/effective behavior policies suitable for the environment) and eliminate the agents that does not adapt to the environment (i.e., the agents that have detrimental/ineffective behavior policies).

3 Agent Structure and Behaviors

Each agent consists of *attributes*, *body* and *behaviors*. Attributes carry descriptive information regarding an agent (e.g., agent ID and energy level). Body implements a functional service an agent provides. For example, an agent may implement a web service in a data center, while another may implement a scientific simulation model in a grid computing system. Behaviors implement the actions inherent to all agents:

- **Migration:** Agents may move between network hosts.
- **Energy exchange and storage:** Agents may gain energy in exchange for providing their services to other agents or users. They may also expend energy for services that they receive from other agents and for resources available at the local network host (e.g., memory space).
- **Replication:** Agents may make their copies in response to higher energy level, which indicates higher demand for the agents. A replicated agent is placed on the host that its parent agent resides on, and it inherits the parent's antibody configuration (behavior policy) as well as the half amount of the parent's energy level. Mutation may occur on the inherited antibody configuration.
- **Reproduction:** Agents may reproduce child agents with other agents (mating partners) running on their local hosts. A child agent is placed on the host that its parents reside on, and it inherits antibody configurations (behavior policies) from both parents through crossover. Each parent gives a child agent the quarter amount of its energy level. Mutation may occur on the antibody configuration of a child agent.
- **Communication**: Agents may communicate with each other for the purposes of, for example, requesting services, exchanging energy units or reproducing child agents.
- **Death**: Agents die due to energy starvation. If an agent cannot balance its energy expenditure with its energy gain, the agent cannot pay for the resources it needs; thus, it dies from lack of energy. When an agent dies, all resources allocated to the agent are released.

Agents expend a certain amount of energy units to invoke each behavior (i.e., behavior cost) except the death behavior.

4 iNet: Agent Adaptation Mechanism in BEYOND

This section overviews how the natural immune system works (Section 4.1) and describes how iNet is designed after the natural immune system (Section 4.2).

4.1 Natural Immune System

The immune system is an adaptive defense mechanism to regulate the body against dynamic environmental changes such as antigen invasions. Through a number of interactions among various white blood cells (e.g., macrophages and lymphocytes) and molecules (e.g., antibodies), the immune system evokes two responses to antigens: *innate* and *adaptive* responses.

In the innate response, the immune system performs self/non-self discrimination. This response is initiated by macrophages and T-cells, a type of lymphocytes. Macrophages move around the body to ingest antigens and present them to T-cells. T-cells are produced in thymus that performs the negative selection. In the negative selection process, thymus removes T-cells that strongly react with the body's own (self) cells. The remaining T-cells are used as detectors to identify foreign (non-self) cells. When a T-cell(s) detects a non-self antigen presented by a macrophage, the T-cell(s) secrete chemical signals to induce the adaptive response.

In the adaptive response, B-cells, another type of lymphocytes, are activated by T-cells. Some of the activated B-cells strongly react to an antigen, and they produce antibodies that specifically kill the antigen. Antibodies form a network and communicate with each other [11]. This immune network is formed with stimulation and suppression relationships among antibodies. With these relationships, antibodies dynamically change their populations and network structure. For example, the population of specific antibodies rapidly increases following the detection of an antigen and, after eliminating the antigen, decreases again.

In order to react a massive number of antigens, the immune system needs to be able to generate a variety of antibodies. A primary repertoire of antibodies is approximately 10^9 using immune genes. B-cells can increase this repertoire further by mutating and recombining immune gene segments so that antibodies can bind an unlimited number of antigens [12].

The immune system regularly encounters anomalies such as immunodeficiency and autoimmunity. Immunodeficiency is a phenomenon that the immune system fails to detect non-self antigens and produce antibodies to eliminate them. Autoimmunity is a phenomenon that the immune system recognizes the constituent self cells as non-self. This results in self-attacks via overreaction of the immune system. When the immune system faces such anomalies, it is alerted with danger signals by cells damaged by the anomalies [13]. Currently, two types of danger signals are known: uric acid [14, 15] and heat shock proteins (HSP) [16, 17]. Uric acid is produced in

response to immunodeficiency, and it stimulates macrophages so that T-cells detects non-self antigens properly. This accelerates the production of antibodies. HSP is produced in response to autoimmunity. HSP reforms broken proteins in macrophages and T-cells so that they stop attacking self cells. This suppresses antibody production.

4.2 iNet Artificial Immune System

The iNet artificial immune system consists of the environment evaluation (EE) facility and behavior selection (BS) facility, which implement the innate and adaptive immune responses, respectively (Figure 1). The EE facility allows an agent to continuously sense a set of current environment conditions as an antigen and classify the antigen to self or non-self. A self antigen indicates that the agent adapts to the current environment conditions well, and a non-self antigen indicates it does not. When the EE facility detects a non-self antigen, it activates the BS facility. The BS facility allows an agent to choose a behavior as an antibody that specifically matches with the detected non-self antigen.
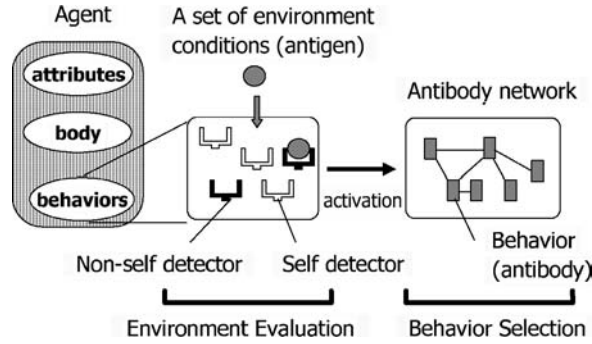
Fig. 1. Design of iNet Adaptation Mechanism

Environment Evaluation Facility

The EE facility performs two steps: initialization and self/non-self classification. The initialization step produces detectors that identify self and non-self antigens. Each antigen is represented as a feature vector (X), which consists of a set of environment conditions, or features, (F_i) and a class value (C):

$$X = (F_1, F_2, ..., F_n, C) \tag{1}$$

C indicates whether a given antigen (i.e., a set of environment conditions) is self (0) or non-self (1). If an agent senses resource utilization and workload (the number of user requests) on the local host, an antigen is represented as follows.
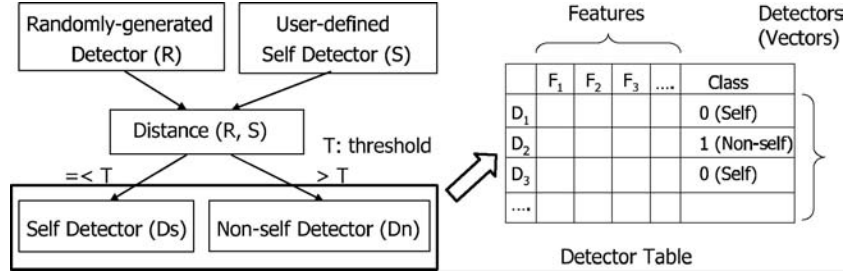
Fig. 2. Initialization Step in the EE Facility

$$X_{current} = ((Low : ResourceUtilization, Low : Workload), 0) \tag{2}$$

The initialization step in the EE facility is designed after the negative selection process in the immune system (Figure 2). As the immune system randomly generates T-cells first, the EE facility generates detectors (feature vectors) randomly. Then, the EE facility separates the detectors into self detectors, which closely match with self antigens, and non-self detectors, which do not closely match with self antigens. This separation is performed via similarity measurement between randomly generated feature vectors (X) and self antigens (S) that human users supply. After the vector matching, both self and non-self detectors are stored in the detector table (Figure 2)[3].

The second step in the EE facility is self/non-self classification of an antigen (a set of current environment conditions). It is performed with a decision tree built from detectors in the detector table and classifies an antigen into self or non-self[4] The decision tree is built using the information gain technique [18]. First, consider one node as a root of decision tree, and it contains all detectors in the detector table. Then, divide the detectors based on one of feature into two subsets of detectors (Assume that each feature has two distinct values.) Each subset goes to one of two child nodes. If all detectors in the subset have the same class value, then the node becomes a leaf node with the class value; otherwise, divide the subset again based on one of the other features into the subsets. Information gain technique suggests how to select a feature at each dividing step so that the number of paths to leaf nodes and the height of tree can be minimized.

Figure 3 shows an example of decision tree. Each node in the tree specifies which feature (environment condition) is considered. Based on the feature values in a given

---

[3] The immune system removes non-self detectors through negative selection. However, in iNet, both self and non-self detectors are used to perform self/non-self classification.

[4] The reasons for using decision trees as an antigen classifier are implementation simplicity and algorithmic efficiency. Decision trees perform classification much faster than other algorithms such as clustering, support vector machine and Markov model algorithms [18]. The efficiency of classification is one of the most important requirements in iNet because each agent periodically senses and classifies its surrounding environment conditions.
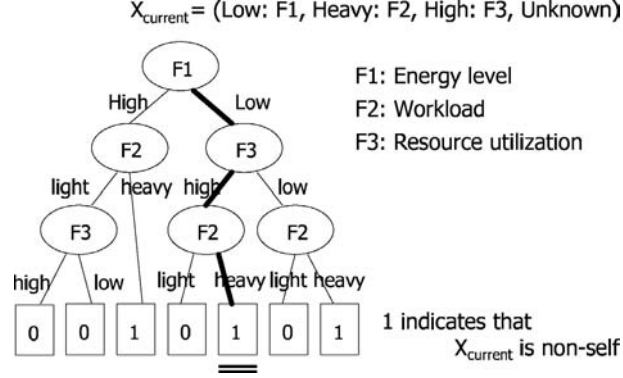
Fig. 3. An Example Decision Tree

antigen, the EE facility travels through tree branches. If the EE facility classifies the antigen to non-self, it activates the BS facility.

Behavior Selection Facility

The BS facility selects an antibody (i.e., agent's behavior) suitable for the detected non-self antigen (i.e., environment conditions). Each antibody consists of three parts: a *precondition* under which it is selected, *behavior ID* and *relationships* to other antibodies. Antibodies are linked with each other using stimulation and suppression relationships. Each antibody has its own concentration value, which represents its population. The BS facility identifies candidate antibodies (behaviors) suitable for a given non-self antigen (environment conditions), prioritizes them based on their concentration values, and selects the most suitable one from the candidates. When prioritizing antibodies (behaviors), stimulation relationships between them contribute to increase their concentration values, and suppression relationships contribute to decrease it. Each relationship has an affinity value, which indicates the degree of stimulation or suppression.

Figure 4 shows a generalized network of antibodies. The antibody i stimulates M antibodies and suppresses N antibodies. m_{ji} and m_{ik} denote affinity values between antibody j and i, and between antibody i and k. m_i is an affinity value between an antigen and antibody i. The concentration of antibody i, denoted by a_i, is calculated with the following equations.

$$\frac{dA_i(t)}{dt} = \left(\frac{1}{N} \sum_{j=1}^{N} m_{ji} \cdot a_j(t) - \frac{1}{M} \sum_{k=1}^{M} m_{ik} \cdot a_k(t) + m_i - k \right) \cdot a_i(t) \qquad (3)$$

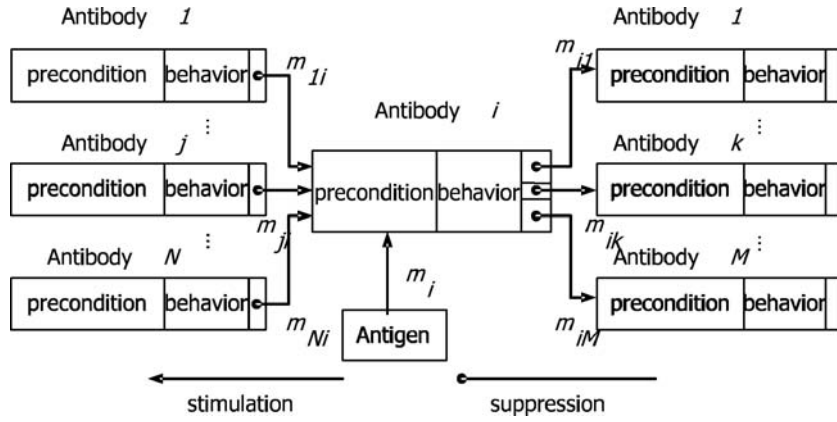$$a_i(t) = \frac{1}{1 + exp(0.5 - A_i(t))} \qquad (4)$$

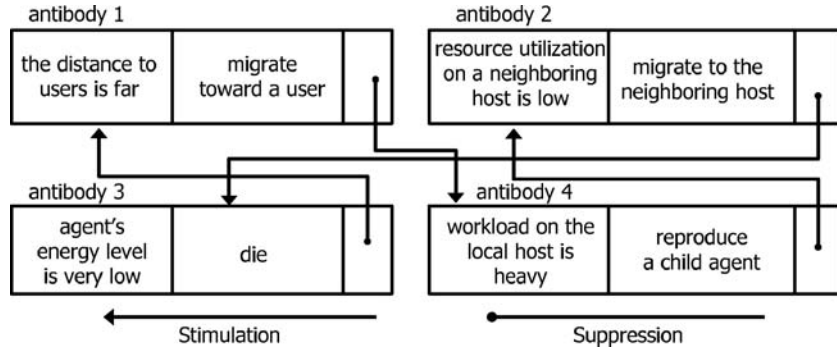**Fig. 4.** A Generalized Antibody Network

Fig. 5. An Example Antibody Network

In Equation (3), the first and second terms in a bracket denote the stimulation and suppression from other antibodies. m_{ji} and m_{ik} are positive between 0 and 1. m_i is 1 when antibody i is stimulated directly by an antigen, otherwise 0. k denotes the dissipation factor representing the natural death of an antibody. Equation (4) is a sigmoid function used to squash the $A_i(t)$ value between 0 and 1.

Every antibody's concentration is calculated 200 times repeatedly. This repeat count is obtained from a previous simulation experience [19]. If no antibody exceeds a predefined threshold during the 200 calculation steps, the antibody whose concentration value is the highest is selected (i.e., winner-tales-all selection). If one or more antibodies' concentration values exceed the threshold, an antibody is selected based on the probability proportional to the concentration values (i.e., roulette-wheel selection).

Figure 5 shows an example network of antibodies. It contains four antibodies, which represent the migration, replication and death behaviors. Antibody 1 repre-

sents the migration behavior invoked when the distance to users is far from an agent. Antibody 1 suppresses Antibody 3 and stimulate Antibody 4. Now, suppose that a (non-self) antigen indicates (1) the distance to users is far, (2) workload is heavy on the local host and (3) resource utilization is low on a neighboring platform. This antigen stimulates Antibodies 1, 2 and 4 simultaneously. Their populations increase, and Antibody 2's concentration value becomes highest because Antibody 2 suppresses Antibody 4, which in turn suppresses Antibody 1. As a result, the BS facility would select Antibody 2.

Evolution of Antibodies

As Section 4.1 describes, the immune system diversifies antibodies by mutating immune genes so that antibodies can react to unanticipated antigens. Similarly, iNet diversifies antibodies via gene operations such as mutation and crossover so that agents can adapt to unanticipated environment conditions. In iNet, each agent encodes and possesses its own antibody configuration (behavior policy) as a set of genes (genotype). The agent genotype consists of the antibody genes, which specify the presence of antibodies, and the affinity genes, which specify relationships among antibodies and their affinity values. When a new agent is born through a replication or reproduction process, it interprets the genes given by its parent(s) and form an antibody network. Figure 6 shows an example genotype and phenotype.

Each agent periodically keeps track of its *fitness*, which quantifies how much it adapts to the the current environment conditions. Agents strive to increase their fitness values by altering their genes through generations. Fitness is calculated as a weighted sum of fitness factors (f_i):
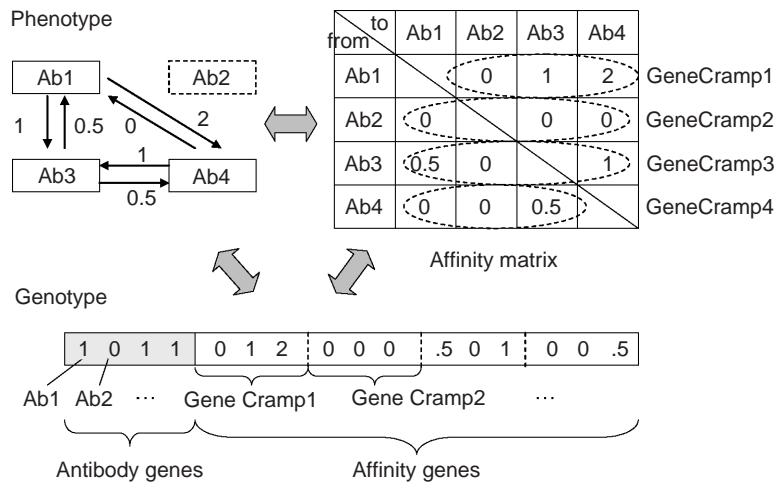
Fig. 6. Phenotype and Genotype of Antibody Network

$$Fitness = \sum w_i \cdot f_i \tag{5}$$

Currently, iNet considers the following six fitness factors. Each factor value is non-negative between 0 and 1.

- **Response time** (f_1): R is the time for each agent to process a single user request. RT is the total of R and the time for a user request and agent response to travel between a user and agent.

$$f_1 = \frac{R}{RT} \tag{6}$$

- **Throughput** (f_2): indicates how many user requests agents process.

$$f_2 = \frac{\# \ of \ user \ requests \ processed \ by \ all \ agents}{Total \ \# \ of \ user \ requests} \tag{7}$$

- **Energy utility** (f_3): indicates the rate of an agent's energy expenditure to its energy gain during its lifetime.

$$f_3 = 1 - \frac{Energy \ expenditure \ during \ lifetime}{Energy \ gain \ during \ lifetime} \tag{8}$$

- **Load balance** (f_4): indicates how user requests (workload) are distributed over agents. m denotes the number of user requests that an agent processes in a unit time. μ_m denotes the expected average number of user requests that each agent is expected to process. M_{max} denotes the maximum number of user requests that an agent can process in a unit time.

$$f_4 = 1 - \frac{|m - \mu_m|}{M_{max}} \ where \ \mu_m = \frac{Total \ \# \ of \ user \ requests}{Total \ \# \ of \ agents} \tag{9}$$

- **Resource utilization balance** (f_5): indicates how resource utilization is distributed over hosts. r denotes the resource utilization rate on the local host that an agent resides on. This is measured as the ratio of the amount of resources consumed by agents on the host to the amount of resources available on the host. μ_r denotes the expected average of resource utilization rate over all hosts that agents reside on.

$$f_5 = 1 - |r - \mu_r| \ where \ \mu_r = \frac{Sum \ of \ resource \ utilization \ rate \ on \ all \ hosts}{\# \ of \ hosts \ that \ agents \ resides \ on} \tag{10}$$

- **Age** (f_6): denotes the lifetime of an agent. S is the total simulation time.

$$f_6 = \frac{Lifetime \ of \ an \ agent}{S} \tag{11}$$

Upon invoking the reproduction behavior, each agent searches mating partner candidates whose fitness values are higher than the agent's fitness value. The candidates are searched on the local host. If the agent cannot find any candidates, it
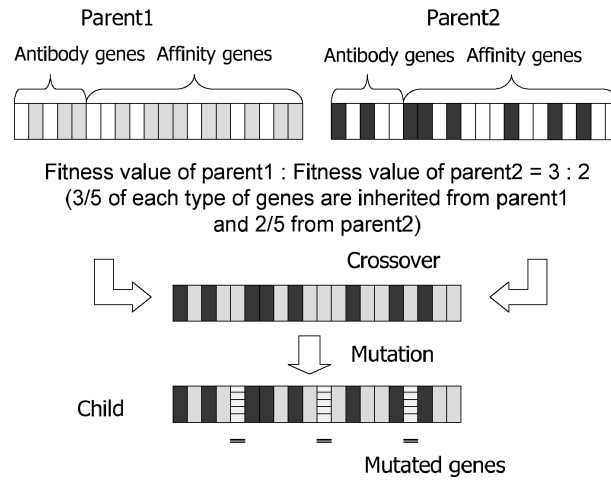
Fig. 7. A Genetic Operations

performs the replication behavior rather than the reproduction behavior. This mating partner selection contributes to increase the population of agents that provide services in higher demand and maintain higher fitness.

In reproduction, two parent agents contribute their genes, via crossover, to a child agent. The amount of their gene contributions follow the ratio of their fitness values. For example, in Figure 7, the fitness value ratio is 3:2 between the parent agent 1 and 2. Thus, the parent agent 1 contributes 60% (3/5) of its genes to a child agent, and the parent agent 2 contributes the rest (2/5). In replication, a parent agent contributes its whole genes to a child agent. In both reproduction and replication, mutation may occur on the genes of a child agent in a certain probability (mutation rate). A reproduced child inherits the quarter amount of energy units from each parent, and a replicated child inherits the half of energy units from its parent.

Self-regulation Process

As Section 4.1 describes, the immune system regulates itself with danger signals when it detects anomalies. Similarly, iNet allows each agent to detect its own deficiencies to recognize antigens, i.e., deficiencies to evaluate whether it adapts well to the current environment conditions. Due to the deficiencies, some agents may invoke their behaviors when they adapt well to the current environment conditions. Others may not when they do not adapt to the current environment conditions. With iNet, each agent can adjust its policies for antigen recognition so that it can perform its behaviors at the right time.

Figure 8 describes the flow of the self-regulation process. Corresponding to danger signals such as Uric acids and Heat shock proteins, each agent responds to two types of signals. *Signal 1* is produced when the current fitness decreases
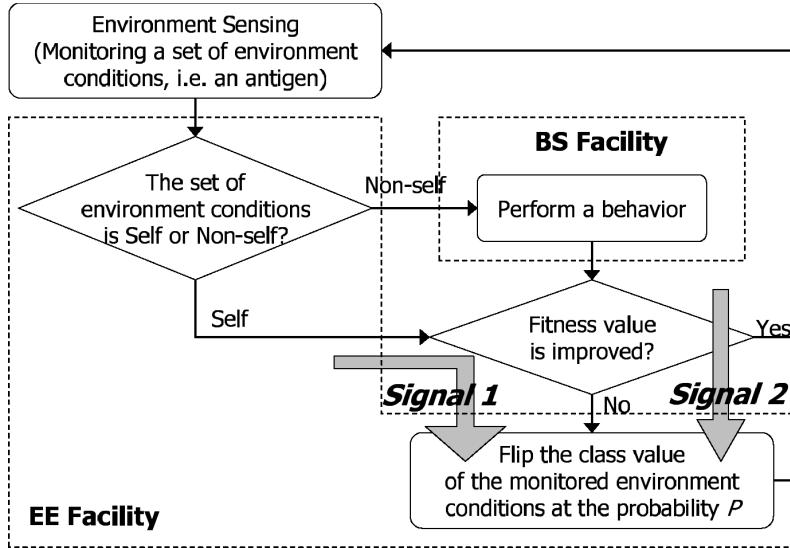
Fig. 8. A Self-regulation Process

by classifying the current environment conditions as self and by not performing any behaviors even though an agent does not adapt well to the conditions (this corresponds to immunodeficiency). *Signal 2* is produced when the current fitness decreases by classifying the current environment conditions as non-self and by performing inappropriate behaviors although there is no necessity to perform behaviors because an agent adapts well to the conditions (this corresponds to autoimmunity).

When an agent receives either of signals, it flips the class value (self ↔ non-self) of the detector, which indicates the miss-classified environment conditions. The strength of the danger signals is represented as a probability, P, that an agent flips the class value. The probability is calculated the weighted sum of the agent's previous fitness, F_{t-1} and the decay of the current fitness, $F_{t-1} - F_t$ as follow:

$$P = \alpha * F_{t-1} + (1 - \alpha) * (F_{t-1} - F_t) \tag{12}$$

5 Simulation Results

This section presents several simulation results to evaluate the autonomous adaptability of agents (network applications). The simulations are carried out on the BEYOND simulator. Figure 9 shows a simulated network as a server farm consisting of network hosts connected in a 10x10 grid topology. BEYOND platform is running on each network host, and each agent implements a web service. Service requests travel from users to agents via user access point. This simulation study assumes that a single (emulated) user runs on the access point and sends service requests to agents. When
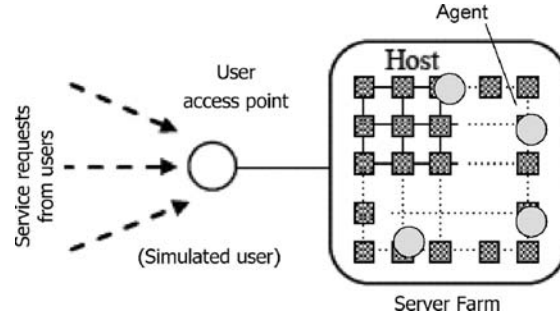
Fig. 9. A Simulated Network

Agent Type	The EE Facility		The BS Facility	
	Tree configuration	Regulation	Antibody network configuration	Evolution
R.ee + R.bs	Random	No	Random	No
R.EE + R.BS	Random	Yes	Random	Yes
M.EE + M.BS	Manual	Yes	Manual	Yes

Fig. 10. Agent Type

a user issues a service request, request messages are broadcasted to search a target agent that can process the issued service requests.

In order to investigate how the self-regulation (the EE facility) and evolution process (the BS facility) impact the adaptability of agents, three different types of agents, described in Figure 10, are evaluated. (1) *R.ee+R.bs*, an agent with a randomly configured tree in the EE facility and a randomly configured antibody network in the BS facility does not perform a regulation and evolution process, (2) *R.EE+R.BS*, an agent with a randomly configured tree and a randomly configured antibody network does dynamically perform a regulation and evolution process, and (3) *R.ee+R.bs*, an agent with a manually tuned tree and a manually tuned antibody network; and does also dynamically perform a regulation and evolution process.

At the beginning of simulations, four agents are randomly deployed on the network. This simulation generates a random workload for web service agents as described in figure 11. The workload trace is designed based on a daily request rate for the www.ibm.com site in February, 2001 [20]. The request rate peaks to about 5,500 requests per min in the morning and 9,000 requests per min in the evening.

Figure 12 shows how agents autonomously adapt their population to workload changes. When agents receive requests, they start to provide their service for users and gain more energy from users. Agents (M.EE+M.BS) successfully adapt their population in timely manner. For example, at 3:00, 6:00, and 15:00, when the workload surges, they increase their population by performing replication or reproduction
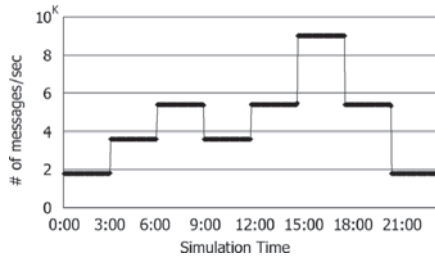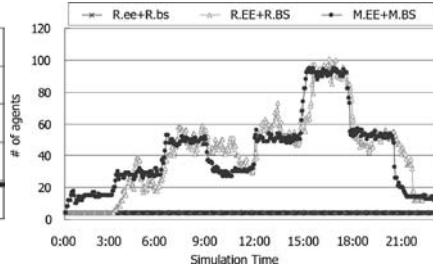
Fig. 11. Workload

Fig. 12. Population of Agents

behavior; subsequently, at 9:00 and 18:00, when the workload drops, they immediately decrease their population by performing death behavior. On the other hand, agents (R.ee+R.bs) did not perform any behaviors because their EE facility classified environment conditions (e.g., workload is high) as self and also their self-regulation process is never executed; so they could not adapt their population. However, agents (R.EE+R.BS) dynamically regulate their policies for environment evaluation in the EE facility so that they perform behaviors to adapt their population. Especially, after 3:00, agents start to update the EE facility as a response to danger signals; subsequently, although they could not immediately reduce their population when the workload drops at 9:00, they adjust the EE facility and successfully perform death behavior at about 11:00. In addition, agents (R.EE+R.BS) may invoke inappropriate behaviors (not suitable for the current environment conditions) in the early stage of simulation due to a randomly configured antibody network. For example, between 3:00 and 6:00, the plotted line for agent population is swinging. This implies that some agents keep invoking death behaviors inappropriately. But, in the late stage of simulation, the plotted line is almost following that of manually configured agents (M.EE+M.BS). It follows that agents tried to dynamically adjust their antibody network by performing reproduction behavior. They reproduce children having the adapted antibody network by which appropriate behaviors are selected in timely manner.

Figure 13 shows how agents autonomously adapt response time for a user. At the beginning of simulation, response time becomes very high because only four agents process 2,000 requests a minute and a distance between the agent and users is long. However, after agents (M.EE+M.BS) accumulate enough energy from users and start migrating towards users or replicating themselves, they rapidly decrease response time. For example, at 3:00, 6:00, and 15:00, the response time dramatically spikes (up to about 7 sec) due to the workload surges, but the agents reduce the response time quickly by adapting their population as described in Figure 12. On the other hand, agents (R.ee+R.bs) cannot reduce their response time at all because they could not perform any behaviors. However, when agents (R.EE+R.BS) recognize that the EE facility wrongly evaluated the environment conditions (i.e., receive danger signals) at 3:00, they tried to regulate evaluation policies in the EE facility. In
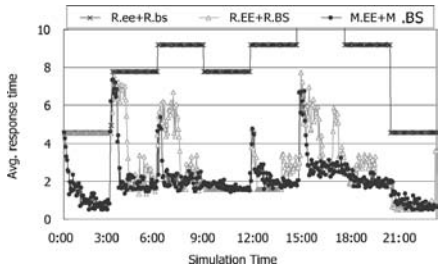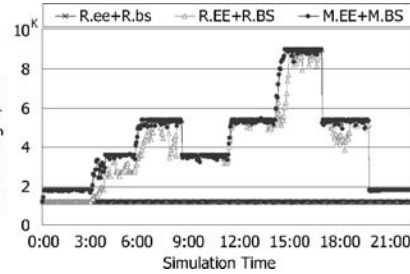
Fig. 13. Response Time
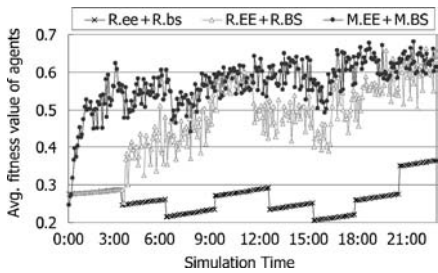

Fig. 14. Throughput of Agents
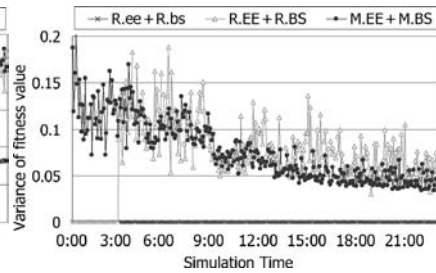

Fig. 15. Average Fitness Value

Fig. 16. Variance of Fitness Value

addition, the reproduced children keep adjusting their antibody network to perform behaviors suitable for the current environment conditions in timely manner.

Figure 14 also shows how three different types of agents dynamically adapt their throughput. It is measured as the number of responses that users receive a minute from agents. Agents (M.EE+M.BS) successfully maintain high throughput by dynamically adapting their locations and population through migration and reproduction behaviors while agents (R.ee+R.bs) could not improve their throughput because the agents did not migrate or replicate at all. Until 3:00, agents (R.EE+R.BS) also could not improve their throughput. However, after 3:00, the agents regulate the behavior invocation by dynamically updating a tree in the EE facility, and the reproduced children adjust an antibody network in the BS facility to invoke appropriate behaviors in timely manner. As the result, they increase their throughput (i.e., tried to reply all user requests in timely manner).

Figure 15 shows the average fitness value of agents (i.e., the degree of adaptation to the environment) as described in Section 4.2. Agents (M.EE+M.BS) dynamically improve their fitness value to about 0.6-0.7 while other agents (R.ee+R.bs) could not improve the fitness value (although the fitness value slightly increases because of energy utility (f3) and age (f6) factors). On the other hand, agents (R.EE+R.BS) keep trying to improve their fitness value after 3:00. In early stage of simulation, because their iNet configuration is not optimized yet, the plotted line for fitness value is swinging (i.e., the fitness value easily drops) compared to that of manually

configured agents (M.EE+M.BS). Some of agents with high fitness value might die unexpectedly. However, in the late stage of simulation, the trace of their fitness value eventually close in that of manually configured agents. It follows that self-regulation and evolution process contribute for agents to autonomously improve their adaptability by dynamically tuning their iNet configurations.

Finally, Figure 16 shows the variance of agents' fitness values; that is, how the fitness values are spread around the average. The variance for agents (M.EE+M.BS) has gradually converged. The lower variance implies that every agents' fitness values are close to each other. Together with the results in figure 15, figure 16 explains that most agents improve their fitness values at the same time. This concludes that the optimal or adapted antibody network is successfully spread out to other surviving agents by evolution; thus, agents adapts to the environment conditions well through generations. Agents (R.EE+R.BS) also reduce the variance gradually. However, the plotted line is unstable compared to that of manually configured agents. Some agents still posses non-adapted antibody network and then invoke inappropriate behaviors. This may kill agents with high fitness value and make the spread speed of the adapted iNet configuration slow.

6 BEYONDwork: Agent Development Environment in BEYOND

BEYONDwork is an application development environment for iNet. It provides two visual modeling languages and a textual programming language for configuring environment conditions, detectors and behavior policies. Figure 17 shows the iNet configuration process with BEYONDwork. BEYONDwork consists of four facilities:
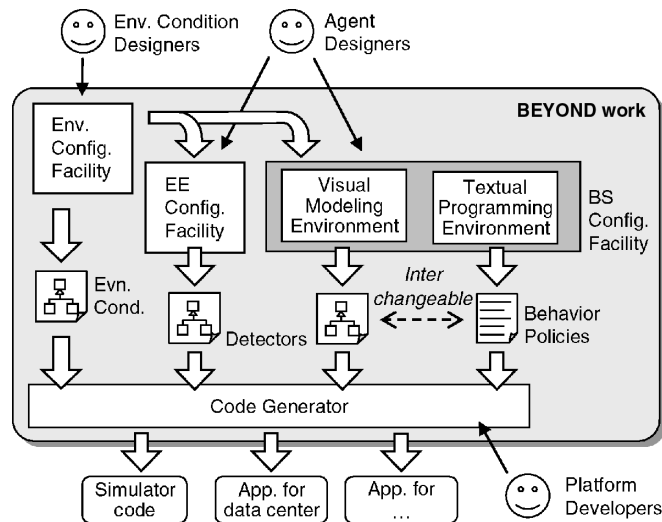

Fig. 17. iNet Configuration with BEYONDwork

environment configuration facility, EE configuration facility, BS configuration facility and code generator. The environment configuration facility allows environment condition designers to configure environment conditions with a visual language. The EE configuration facility allows agent designers to configure a set of detectors to identify self and non-self antigens (see Section 4.2) based on environment conditions configured in the environment configuration facility. The BS configuration facility allows agent designers to configure their agents' behavior policies (antibody configuration) with visual or textual languages. Both languages have the same level of expressiveness, and the artifacts of the languages (models and programs) are transparently translatable with each other. Agent designers can configure the behavior policy of each agent through the use of either language.

Once environment conditions, detectors and a behavior policy are complete in the form of visual models or textual programs, the code generator transforms them to compilable source code by following a transformation rule between the languages and source code. The transformation rules are implemented by platform developers, who know the details of platform technologies. (e.g., operating systems, middleware, simulators and programming languages) Through changing one transformation rule to another, the code generator can generate source code that are compatible with different deployment environments such as simulators and real networks. Environment condition designers and agent designers do not have to write different models/programs for the same agent running on different platform technologies. This flexible code generation feature improves the productivity of agent designers. Currently, BEYONDwork supports Java code generation for a simulator in BEYOND.

Figure 18 shows the environment configuration facility. As Figure 18 illustrates, the visual language visualizes an environment condition as a rectangle. Each rectangle can contain arbitrary number of rounded rectangles representing categories of a corresponding environment condition. For example, in Figure 18, the `LocalWorkload` environment condition has two categories, `HEAVY` and `LIGHT`. Also, each category specifies a condition to classify a corresponding environment
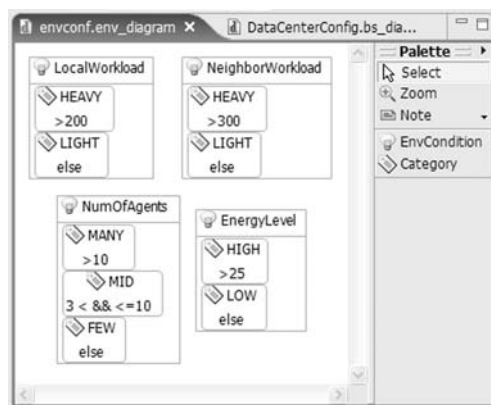

Fig. 18. BEYONDwork Environment Configuration Facility

condition. In iNet, each environment condition is supposed to have one representative value, e.g., workload value or the number of agents, and the representative value is used to identify the category of a corresponding environment condition. For example, in Figure 18, the `LocalWorkload` environment condition is classified as `HIGH` when its representative value exceeds 200, otherwise classified as `LIGHT`.

The details of representative values, e.g., how and when to obtain the value, are hidden from environment condition designers and agent designers. Platform developers implement such details on skeleton code generated by the code generator. For example, the following is a fragment of Java source code generated from the `LocalWorkload` environment condition in Figure 18. By implementing the `getRepValue` method, e.g., returns a request rate, CPU load, or the summation of them, a representative value of an environment condition is retrieved and evaluated against conditions specified by each category. The environment configuration facility is implemented on Eclipse Graphical Modeling Framework (GMF)[5]. The transformations from visual models and Java source code are implemented with a model-code transformation engine in openArchitectureware[6].

```java
class LocalWorkload extends EnvCondition{
  enum Category{ HIGH, LOW };
  public Category evaluate(){
    double repValue = getRepValue();
    if( repValue > 200 ){ return Category.HIGH; }
    else{ return Category.LOW; }
  }
  private double getRepValue(){
    // TODO: platform developers add code here
  }
}
```

Figure 19 shows the EE configuration facility appears as one of windows in BEYONDwork, located below the BS configuration facility (Figure 20). Each column in the table represents an environment condition configured in the environment configuration facility (Figure 18) and each row represents a detector. The facility allows agent designers to add and remove detectors, and configure detectors by selecting the categories of each environment condition. For example, in Figure 18, the `NumOfAgents` environment condition is configured to have three categories, `MANY`, `MID` and `FEW`, and cells in the corresponding column in Figure 19 allows agent designers to select its value from `MANY`, `MID` and `FEW`. From a set of detectors, BEYONDwork automatically create a decision tree and deploys it in agents (see Section 4.2).

Figure 20 and 21 show the visual modeling and textual programming environments in the BS configuration facility, respectively. As Figure 20 illustrates, the visual language visualizes an antibody as a rounded rectangle. Each rectangle consists of three compartments: (1) the name and the initial concentration of an antibody,

---

[5] www.eclipse.org/gmf/

[6] www.openarchitectureware.org

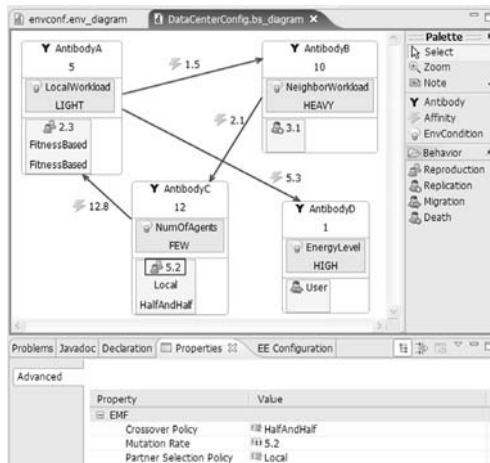**Fig. 19.** BEYONDwork EE Configuration Facility

Fig. 20. Visual Modeling Environment in BEYONDwork BS Configuration Facility

(2) an environment condition to which an antibody reacts, and (3) an agent behavior and its properties. For example, in Figure 20, `AntibodyA`'s initial concentration value is 5, and it represents the reproduction behavior. The behavior is invoked when `LocalWorkload` is high. A stimulation/suppression relationship between antibodies is visualized as an solid arrow between rounded rectangles. Each arrow has value, which represents affinity value of a relationship. As Figure 20 demonstrates, the visual language supports all the concepts in antibody configurations as built-in model elements, and agent designers can configure antibodies (agent behavior policies) in an intuitive and rapid manner. The BS configuration facility is implemented on GMF and openArchitectureWare as well as the environment configuration facility.

In the textual language (Figure 21), each antibody is defined with the built-in keyword `antibody`. The program in Figure 21 and the model in Figure 20 define the semantically same antibody configuration. As Figure 21 shows, the textual programming environment in BEYONDwork shows built-in keywords in boldface, automatically performs a syntax check, and reports syntax errors while antibody designers configure antibodies. In Figure 21, a syntax error is reported as a cross mark. (The textual language does not support keyword `energylevel` but `EnergyLevel` because of the environment conditions defined in Figure 18.) The textual programming environment in the BS configuration facility is implemented on Eclipse. The

Fig. 21. Textual Programming Environment in BEYONDwork BS Configuration Facility

transformations from textual programs and Java source code are implemented with a model-code transformation engine in openArchitectureware.

The following is a fragment of Java source code generated from the textual program in Figure 21.

```java
void setupAntibodiesOfINet(){
  Antibody antibodyA =
   new Antibody( "AntibodyA", 5, LocalWorkload.HIGH,
    new Reproduction(
     2.3, CROSSOVER.FITNESSBASED, PARTNER.FITNESSBASED ) );
  Antibody antibodyD =
   new Antibody( "AntibodyD", 1, EnergyLevel.HIGH,
    new Migration( DirectionPolicy.USER ) );

  ImmuneNetwork inet = getImmuneNetwork();
  inet.add( antibodyA );
  inet.add( antibobyD );
  antibodyA.addAffinity( antibodyD, 5.3 );
}
```

The BS configuration facility allows agent designers to not only configure an antibody configuration (behavior policy) from scratch, but also investigate and fine tune existing antibody configurations in running agents. In iNet, antibody configurations are evolved automatically via genetic operations (see Section 4.2). The BS configuration facility helps agent designers to understand evolved antibody configurations by showing it in a visual manner, and experienced agent designers can fine-tune them by hand.

Without the BS configuration facility, agent designers need to know the details on how to implement agents in Java (e.g., how to define new agents, where to implement antibody configuration code, and which iNet API to use.) For example, agent designers need to define a new class extending the `Agent` class provided by a simulator in BEYOND. Also, as the above code fragment shows, they need to write

the `setupAntibodiesOfINet()` method using iNet API in order to configure the agent's antibodies. The visual and textual languages hide these implementation details and allow agent designers to focus on the design of antibody configurations. In addition, compared with the Java code shown above, a model or program in the BS configuration facility is easier to read and understand.

7 Related Work

This paper describes several extensions to the prior work on iNet [19, 21]. [19] does not investigate the iNet evolutionary mechanism. Thus, agent designers needed to manually and carefully configure antibodies in their agents at design time. In contrast, the iNet evolutionary mechanism allows agents to autonomously adjust their antibody configurations at runtime; it does not require manual antibody configurations. [21] describes preliminary simulation results of the iNet evolutionary mechanism; however, it does not investigate the languages in BEYONDwork as well as the self-regulatory mechanism in the iNet EE facility.

The Bio-Networking Architecture [22] is similar to BEYOND in that it applies biological principles and mechanisms to allow network applications to autonomously adapt to dynamic environmental changes in the network. However, its adaptation engine is different from iNet. While iNet is designed after immune responses, [22] employs a simple weighted sum calculation for behavior selection. Although [22] has an evolutionary mechanism that dynamically adjusts weight values in the weighted sum calculation, agent designers still need to manually define a weighted sum equation for each behavior and configure a threshold value for each weighted sum equation. In contrast, iNet requires no manual configuration work for agent designers.

Artificial immune systems have been proposed and used in various application domains such as anomaly detection [23] and pattern recognition [24]. [23] focuses on the generation of detectors for self/non-self classification and improves the negative selection process of the artificial immune system. [24] focuses on the accuracy for the matchmaking of an antigen and antibody. Unlike those work, this paper proposes an artificial immune system to improve autonomous adaptability of network applications. To the best of our knowledge, this work is the first attempt to apply an artificial immune system to this domain.

In addition, some research work [25] using artificial immune systems extend their work with the concept of danger signals. [25] proposes the mechanism to detect misbehaving nodes as antigens based on event sequences of routing process in ad hoc network. Danger signals contribute to reduce the number of false positives (i.e., the system evaluates a correctly working node as a misbehaving node) by dynamically updating the definition of normal event sequences (self). On the other hand, iNet self-regulation process allows agents to respond false positives as well as false negatives (i.e. the system cannot catch unknown non-self antigens).

BEYONDwork provides visual and textual languages to configure iNet, i.e., configuring environment conditions, detectors and behavior policies. The work of the languages in BEYONDwork is parallel to the existing research on domain specific

languages (DSLs) [26]. The languages are considered as DSLs focusing on directly capturing the concepts and mechanisms specific to a particular problem domain. There are several DSLs to model biological systems such as biochemical networks for simulating and understanding biological systems (e.g., [27, 28]). However, the objective of the languages in BEYONDwork is different from theirs; languages in BEYONDwork aim to model biological (immunological) mechanisms for building autonomous and adaptive network applications. This work is the first attempt to investigate a DSL for biologically-inspired networking.

8 Conclusion

This paper describes the BEYOND architecture, which applies biological principles and mechanisms to design evolvable network applications that autonomously adapt to dynamic environmental changes in the network. This paper proposes two key components in BEYOND: (1) a self-regulatory and evolutionary adaptation mechanism for agents, called iNet, and (2) an agent development environment, called BEYONDwork. iNet allows each agent to autonomously sense its surrounding environment conditions (i.e., antigens) and adaptively invoke a behavior (i.e., antibody) suitable for the conditions. iNet also allows each agent to detect its own deficiencies to recognize antigens and regulate its policies for antigen recognition. Agents evolve their antibodies so that they adapt to unexpected environmental changes. Simulation results show that agents adapt to changing network environments by self-regulating their antigen recognition and evolving their antibodies through generations. In addition, BEYONDwork provides visual and textual languages to configure iNet in an intuitive and easy-to-understand manner. It accepts the visual models and textual programs built with the proposed languages, and transforms them to Java code that are compilable and runnable on a simulator for BEYOND. This code generation enables rapid development configuration of agents, thereby improving the productivity of agent developers.

References

1. D. F. Carr, "How google works," *Baseline*, July 2006.
2. J. Markoff and S. Hansell, "Google's not-so-very-secret weapon," *International Herald Tribune*, June 2006.
3. P. Dini, W. Gentzsch, M. Potts, A. Clemm, M. Yousif, and A. Polze, "Internet, grid, self-adaptability and beyond: Are we ready?" *Proc. of IEEE International Workshop on Self-Adaptable and Autonomic Computing Systems*, August 2006.
4. R. Sterritt and D. Bustard, "Towards an autonomic computing environment," *Proc. of IEEE International Workshop on Database and Expert Systems Applications*, September 2003.
5. J. Rolia, S. Singhal, and R. Friedrich, "Adaptive internet data centers," *Proc. of International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet*, July 2000.

6. S. Ranjan, J. Rolia, E. Knightly, and H. Fu, "Qos-driven server migration for internet data centers," *Proc. of International Workshop on Quality of Service*, May 2002.

7. N. Minar, K. H. Kramer, and P. Maes, "Cooperating mobile agents for dynamic network routing," in *Software Agents for Future Communications Systems*, A. Hayzelden and J. Bigham, Eds.   Springer, June 1999.

8. R. Albert, H. Jeong, and A. Barabasi, "Error and attack tolerance of complex networks," *Nature*, July 2001.

9. G. Cabri, L. Leonardi, and F. Zambonelli, "Mobile-agent coordination models for internet applications," *IEEE Computer*, February 2000.

10. S. Camazin, J. L. Deneubourg, N. R. Franks, J. Sneyd, G. Theraula, and E. Bonabeau, *Self Organization in Biological Systems*.   Princeton University Press, May 2003.

11. N. K. Jerne, "Idiotypic networks and other preconceived ideas," *Immunological Review*, 1984.

12. C. Berek, "Somatic hypermutation and b-cell receptor selection as regulators of the immune response," *Transfusion Medicine and Hemotherapy*, 2005.

13. P. Matzinger, "The danger model: A renewed sense of self," *Science*, April 2002.

14. K. R. Jerome and L. Corey, "The danger within," *The New England Journal of Medicine*, 2004.

15. W. R. Heath and F. R. Carbonel, "Immunology: Dangerous liaisons," *Nature*, October 2003.

16. B. Goldman, "White paper: Heat shock proteins' vaccine potential from basic science breakthroughs to feasible personalized medicine," *Antigenic*, July 2002.

17. W. A. Fenton and A. L. Horwich, "Chaperonin-mediated protein folding: Fate of substrate polypeptide," *Quarterly Reviews of Biophysics*, May 2003.

18. T. Mitchell, *Machine Learning*.   McGraw-Hill, 1997.

19. C. Lee and J. Suzuki, "Biologically-inspired design of autonomous and adaptive grid services," *Proc. of International Conference on Autonomic and Autonomous Systems*, July 2006.

20. J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle, "Managing energy and server resources in hosting centers," *Proc. of the Eighteenth Symposium on Operating Systems Principles*, October 2001.

21. C. Lee and J. Suzuki, "An immunologically-inspired adaptation mechanism for evolvable network applications," *Proc. of the Fourth IEEE Consumer Communications and Networking Conference*, January 2007.

22. T. Nakano and T. Suda, "Self-organizing network services with evolutionary adaptation," *IEEE Transactions on Neural Networks*, September 2005.

23. F. A. Gonzalez and D. Dasgupta, "Anomaly detection using real-valued negative selection," *Genetic Programming and Evolvable Machines*, 2003.

24. L. N. de Castro and J. I. Timmis, "Artificial immune systems: A novel paradigm to pattern recognition," in *Artificial Neural Networks in Pattern Recognition*, J. M. Corchado, L. Alonso, and C. Fyfe, Eds.   University of Paisley, UK, 2002.

25. S. Sarafijanovic and J.-Y. L. Boudec, "An artificial immune system approach with secondary response for misbehavior detection in mobile ad-hoc networks," *IEEE Transactions on Neural Networks, Special Issue on Adaptive Learning Systems in Communication Network*, April 2005.

26. G. Cook, "Domain-specific modeling and model-driven architecture," in *The MDA journal: Model Driven Architecture Straight from the Masters*. Meghan-Kiffer Press, December 2004, ch. 5.

27. M. Hucka, A. Finney, B. Bornstein, S. Keating, B. Shapiro, J. Matthews, B. Kovitz, M. Schilstra, A. Funahashi, J. Doyle, and H. Kitano, "Evolving a lingua franca and associated software infrastructure for computational systems biology: The systems biology markup language (sbml) project," *Systems Biology Journal*, June 2004.
28. F. Kolpakov, "Biouml - framework for visual modeling and simulation biological systems," in *International Conference on Bioinformatics of Genome Regulation and Structure*, July 2002.