Exploring Biologically-inspired Evolvable Network Applications with the BEYOND Architecture

(Invited Paper)

Chonho Lee, Hiroshi Wada and Junichi Suzuki
Department of Computer Science
University of Massachusetts, Boston
Boston, MA 02125, USA
Email: {chonho, shu, jxs}@cs.umb.edu

***Abstract*— The BEYOND architecture applies biological principles and mechanisms to design evolvable network applications that autonomously adapt to dynamic environmental changes in the network. This paper describes two key components in BEYOND: (1) an evolutionary adaptation engine, called iNet, for network applications and (2) an application development environment, called BEYONDwork, for the adaptation engine. iNet is designed after the mechanisms behind how the immune system works. It models a set of environment conditions (e.g., network traffic) as an antigen and a behavior of network applications (e.g., migration and reproduction) as an antibody. iNet allows each network application to autonomously sense its surrounding environment conditions (i.e., antigens) and adaptively invoke a behavior (i.e., antibody) suitable for the conditions. The configuration of antibodies evolves via genetic operations such as mutation and crossover. BEYONDwork provides visual and textual languages to configure antigens and antibodies in iNet. The languages increase the ease of specifying and modifying iNet configurations. Simulation results show that iNet allows network applications designed with BEYONDwork to evolve themselves to adapt to changing network environments.**

I. Introduction

Large-scale network applications such as grid computing applications and data center applications face several challenges, particularly autonomy and adaptability, as they have been increasing in complexity and scale [1]–[5]. 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.

As inspiration for a new design paradigm for network applications, the authors of the paper observe that various biological systems have developed the mechanisms necessary to meet the above requirements (i.e., autonomy and adaptability) [6]. For example, bees act autonomously, influenced by local conditions and local interactions with other bees. A bee colony adapts to dynamic environmental conditions. When the amount of honey in a hive is low, many bees leave the hive to gather nectar from flowers. When the hive is full of honey, bees rest in the hive. Based on this observation, the authors of the paper believe that, if network applications are designed after certain biological principles and mechanisms, they may be able to increase their autonomy and adaptability.

The proposed architecture, called BEYOND[1], applies 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 migration, replication, reproduction and death.

This paper focuses on two key components in BEYOND: (1) an evolutionary adaptation engine for agents and (2) an application development environment for the adaptation engine. The proposed adaptation engine, called iNet, is designed after the mechanisms behind how the immune system specifically produce antibodies to eliminate antigens (e.g., viruses) 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 the configuration of iNet (antibodies) defines the agent's behavior policy (i.e., when to invoke which behavior). iNet allows each agent to autonomously sense its surrounding environment conditions (i.e., an antigen) for evaluating whether it adapts well to the sensed conditions, and if it does not, adaptively invoke a behavior (i.e., an 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 population; agents can improve their throughput performance. Also, agents may invoke the migration behavior to move toward 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 performance.

iNet also allows each agent to dynamically evolve its own antibody configuration (i.e., behavior policy) so that the configuration becomes fine-tuned to environment conditions in the network. This evolution process occurs via genetic operations such as mutation and crossover, which alter antibody configurations during agent replications and reproductions. Agent

[1]Biologically-Enhanced sYstem architecture beyond Ordinary Network Design

evolution frees application developers (agent developers) from anticipating all possible environmental changes and tuning their agents' antibody configurations (behavior policies) to the changes at design time, thereby significantly simplifying the implementation and configuration of agents.

The second focus of this paper is an application development environment for iNet, called BEYONDwork. It provides visual and textual languages to configure antibodies of each agent in an intuitive and easy-to-understand manner. BEYONDwork 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 in BEYOND. This code generation enables rapid development and configuration of agents, thereby improving the productivity of agent developers.

II. Design Principles

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

- **Decentralization:** In various biological systems (e.g., bee colony), there are no central leader entities to control or coordinate individual entities in order to increase scalability and survivability. Similarly, in BEYOND, there are no central entities to control and coordinate agents 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 environmental 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 [6]. In BEYOND, agents only interact with nearby agents. They behave against dynamic environment conditions such as user demands and resource availability. Through collective behaviors and interactions of individual agents, desirable system characteristics such as adaptability (e.g., load balancing and resource efficiency) emerge in a swarm of agents.
- **Lifecycle:** Biological entities strive to seek and consume food for living. Similarly, 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., network bandwidth and memory space). The abundance or scarcity of stored energy in agents affects their lifecycle. For example, an abundance of stored energy indicates higher demand to an agent; thus, the agent may be designed to favor reproduction or replication to increase its availability. A scarcity of stored energy (i.e., an indication of lack of demand) causes death of the agent.
- **Evolution:** In addition to individual adaptation, in which individual biological entities behave according to environmental changes, they evolve as a species to increase the fitness to the environment across generations. In BEYOND, as described above, individual agents adapt to environmental changes in the network by invoking their behaviors. In addition, agents evolve their antibody configurations as a species (group) 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 (i.e., behavior policies). This is generated via mutation and crossover during agent replications and reproductions. Natural selection retains the agents that adapt well to environment conditions (i.e., the agents that have beneficial/effective behavior policies suitable for the environment conditions) and eliminate the agents that does not adapt to the conditions (i.e., the agents that have detrimental/ineffective behavior policies).

III. 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 agent 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).
- **Replications:** 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, as crossover, antibody configurations (behavior policies) from both parents. Each of two parents 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.

Each agent expends energy to invoke behaviors (i.e., behavior cost) except death behavior.

IV. Design of iNet Adaptation Engine

This section overviews how the natural immune system works (Section IV-A), describes how iNet is designed after the natural immune system (Section IV-B).

A. 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 and trained through the negative selection process. In this process, thymus removes T-cells that 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 activate the second immune response: 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 [10]. This immune network is formed with stimulation and suppression relationships among antibodies. With these relationships, antibodies dynamically change their population and network structure. For example, the population of specific antibodies rapidly increases following the recognition of an antigen and, after eliminating the antigen, decreases again. Through this self-regulation mechanism, the adaptive immune response is an emergent product of interactions among antibodies.

In order to react a variety of antigens, the immune system needs to be able to generate a massive number 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 [11].

B. 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.

1) 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.

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

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 feature table (Figure 2)[2].

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 feature table and classifies an antigen into self or non-self[3]. Figure 3 shows an example decision tree. Each node in the tree specifies which feature (environment condition) is considered. Based on the feature values in a given antigen, the EE facility travels through tree branches. If the EE facility classifies the antigen to non-self, it activates the BS facility.

2) 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

[2]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.

[3]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 [12]. The efficiency of classification is one of the most important requirements in iNet because each agent periodically senses and classifies its surrounding environment conditions.

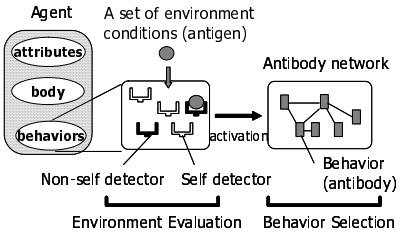

Fig. 1. The iNet Architecture

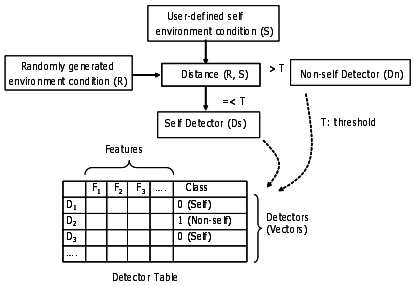

Fig. 2. Initialization Step in the EE Facility

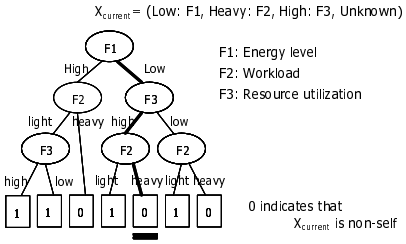

Fig. 3. An Example Decision Tree

their concentration values, and suppression relationships contribute to decrease it. Each relationship has an affinity value, which indicates the degree of stimulation or suppression.

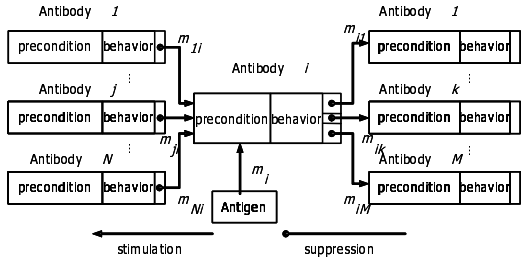

Fig. 4. A Generalized Antibody Network

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.

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

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

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 [13]. 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 represents 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.

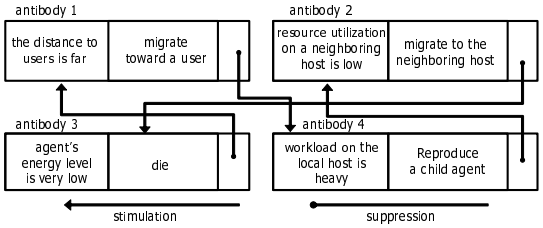

Fig. 5. An Example Antibody Network

3) Evolution of Antibodies: As Section IV-A 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 a genotype (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 (antibody configurations) through generations. Fitness is calculated as a weighted sum of fitness factors (f_i):

$$Fitness = \sum w_i \cdot f_i \quad (5)$$

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

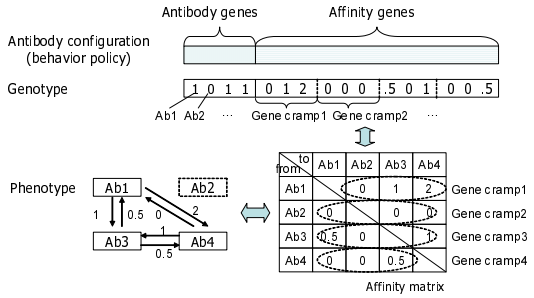

Fig. 6. Agent Genotype and Phenotype

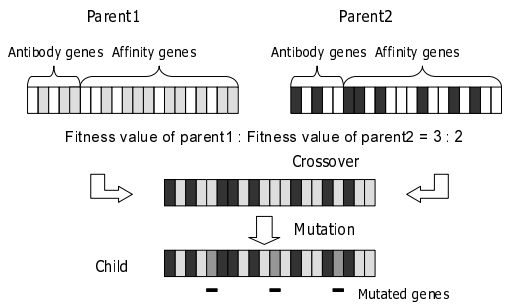

Fig. 7. An Example of Genetic Operation

- **Response time** (f_1), the response time of an agent for users. R is the time for each agent to process a single user request.

$$f_1 = \frac{R}{Response\ time} \tag{6}$$

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

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

- **Energy utility** (f_3), indicates the rate of agents' energy expenditure to their energy gain.

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

- **Resource efficiency** (f_4), indicates how much resources agents consume against the workload they face.

$$f_4 = 1 - \frac{Resource\ utilization\ on\ the\ local\ host}{Workload\ on\ the\ local\ host} \tag{9}$$

Resource utilization is measured by the rate of the resource consumption of agents on the local host to the total amount of resources available on the local host. Workload is the rate of the total number of user requests the local host receives to the number of user requests agents can process in a unit time on the local host.

- **Age** (f_5) denotes the lifetime of an agent, L is the actual age of an agent, divided by a costant S.

$$f_5 = \frac{1}{1 - exp(-L + 0.5)},\ where\ L = \frac{Age\ of\ an\ agent}{S} \tag{10}$$

- **Load balancing** (f_6), indicates how agents distribute their workload among them. N denotes the number of neighboring hosts, and M denotes the maximum number of agents that can run on the local host.

$$f_6 = 1 - \frac{LB}{M},\ where\ LB = \#\ of\ agents\ on\ the\ local\ host - \frac{\#\ of\ agents\ on\ the\ local\ and\ negihboringhosts}{N} \tag{11}$$

When an agent invokes the reproduction behavior, it searches the candidates of mating partner agents whose fitness values are higher than the agent's fitness value. The candidate mating partners are searched on the local host. If the agent cannot find such a partner, it replicates itself. This mating partner selection contributes to increase the population of agents that provide services in higher demand and maintains higher fitness values than other agents.

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 parent agent 1 and 2. Thus, parent agent 1 contributes 60% of its genes to a child agent, and parent agent 2 contributes the rest. In replication, a parent agent contribute its whole genes to a child agent. Both in reproduction and replication, mutation may occur on the genes of a child agent in a certain probability.

V. BEYOND Development Environment

BEYONDwork is an application development environment for iNet. It provides a visual modeling language and a textual programming language, collectively called iNet Configuration Language (ICL). 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 (antibody configuration) of each agent through the use of either language.

Figure 8 shows the agent configuration process with BEYONDwork. BEYONDwork consists of two facilities: agent configuration facility and code generator. The agent configuration facility allows agent designers to configure their agents' behavior policies with the visual or textual ICL. Once a behavior policy is complete in the form of visual models or textual programs, the code generator transforms the behavior policy to compilable source code by following a transformation rule between ICL and source code. 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. Agent designers do not have to write different ICL models/programs for the same agent running on different deployment environments. This flexible code generation feature improves the productivity of agent designers. Currently, BEYONDwork supports Java code generation for a simulator in BEYOND.

Figure 9 and 10 show the visual modeling and textual programming environments in BEYONDwork, respectively. As Figure 9 illustrates, the visual ICL visualizes an antibody as a rounded rectangle. Each rectanble consists of three compartments: (1) the name and the initial concentration of an antibody, (2) an environment condition to which an antibody reacts, and (3) an agent behavior and its properties.

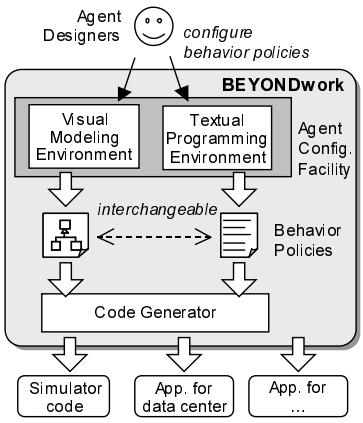

Fig. 8. Agent Configuration with BEYONDwork

For example, in Figure 9, `AntibodyAA`'s initial concentration value is 5, and it represents the reproduction behavior. The behavior is invoked when workload is high. An 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 9 demonstrates, the visual ICL 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 visual modeling environment of BEYONDwork is implemented on Eclipse Graphical Modeling Framework (GMF)[4]. The transformations from ICL models and Java source code are implemented with a model-code transformation engine in openArchitectureware[5].

In the textual ICL (Figure 10), each antibody is defined with the built-in keyword `antibody`. The program in Figure 10 and the model in Figure 9 define the semantically same antibody configuration. As Figure 10 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 10, a syntax error is reported as a wavy underline. (The textual ICL does not support keyword `resources` but `resource`.) The textual programming environment in BEYONDwork is implemented on Eclipse. The transformations from ICL 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 ICL program in Figure 10.

```
void setupAntibodiesOfINet(){
  Antibody aa =
   new Antibody( "AntibodyAA", 5, "workload", "high",
    new Reproduction( 2.3, "fitnessbased", "fitnessbased" ) );
  Antibody bb =
   new Antibody( "AntibodyBB", 10, "resource", "high",
    new Mutation( 3 ) );

  ImmuneNetwork inet = getImmuneNetwork();
  inet.add( aa );
  inet.add( bb );
  aa.addAffinity( bb, 1.5 );
}
```

[4]www.eclipse.org/gmf/

[5]www.openarchitectureware.org

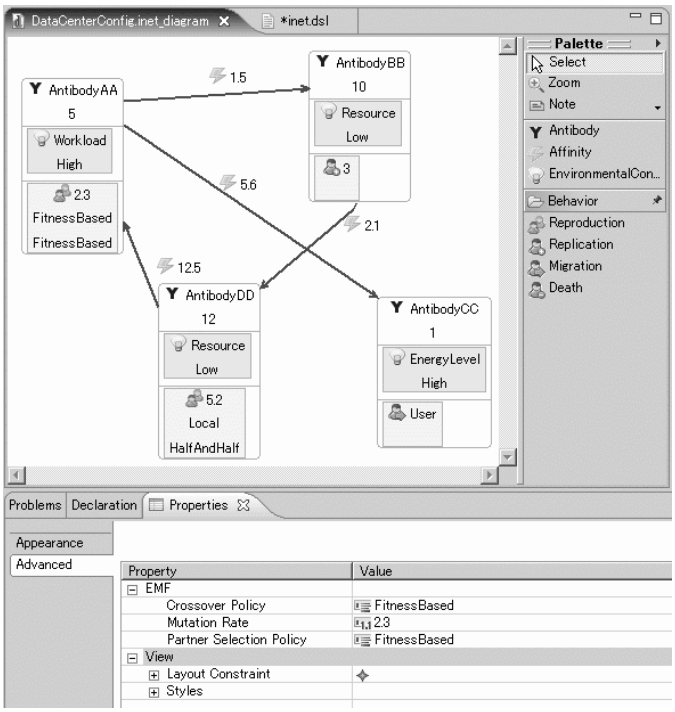

Fig. 9. BEYONDwork Visual Modeling Environment

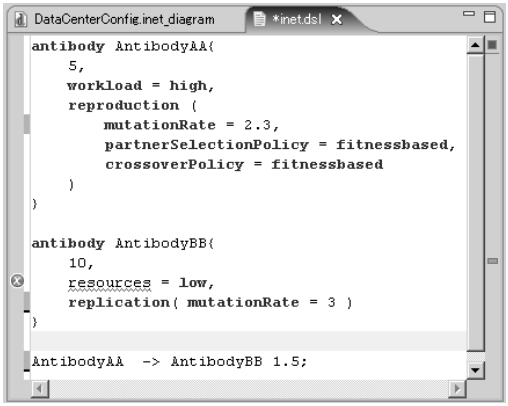

Fig. 10. BEYONDwork Textual Programming Environment

Without ICL, 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. ICL hides these implementation details and allows agent designers to focus on the design of antibody configurations. In addition, compared with the Java code shown above, a model or program in ICL is easier to read and understand.

VI. Simulation Results

This section presents several simulation results to evaluate the autonomous adaptability of agents designed with BEYONDwork. The simulations are carried out on the BEYOND simulator. Figure 11 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 a user issues a service request, request messages are broadcasted to search a target agent that can process the issued service requests.

This simulation generates a random workload for web service agents as described in figure 12. The workload trace is designed based on a daily request rate for the www.ibm.com site in February, 2001 [14]. The request rate peaks to about 5,500 requests per min in the morning and 9,000 requests per min in the evening. At the beginning, four agents whose antibody network is randomly configured are randomly deployed. In order to evaluate how the evolutionary process impacts the adaptability of agents, two different types of agents-agents with a reproduction behavior and without it, i.e. with evolution and without evolution-are compared.

Figure 13 shows how agents autonomously adapt their population to workload changes. When agents receive requests, they start to provide their service for users. Agents gain more energy from users and try to perform replication or reproduction behavior to increase their population. However, agents without evolution cannot perform a replication behavior appropriately; moreover, they may incorrectly invoke a death behavior despite receiving user requests. It follows that randomly configured agents do not have an antibody network suitable for the environment. On the other hand, agents with evolution successfully adjust the configuration of antibody network and increase their population. They reproduce children having the adaptive antibody network by which a replication behavior is appropriately selected according to the workload.

Figure 14 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 the agents accumulate enough energy from users and start to migrate towards users and replicating themselves, they rapidly decrease response time. For agents with evolution, when workload is generated, the response time spikes up to 10 seconds (at 3:00), but they decrease it to 2 seconds in 30 minutes. It follows that they reproduce children who successfully invoke migration and replication behaviors according to the workload. On the other hand, agents without evolution cannot reduce their response time because they did not properly migrate towards users and increase their population.

Figure 12 also shows how two different types of agents dynamically adapt their throughput to the workload changes. It is measured as the number of responses that users receives a minute from agents. Agents with evolution autonomously maintain high throughput by dynamically adjusting their locations and population through migration and reproduction behaviors while agents without evolution cannot achieve their throughput to workload becasue some agents did not migrate or replicate properly.

Figure 15 shows the average fitness value of agents (i.e., the degree of adaptation to the environment). While agents without evolution do not improve their fitness value, agents with evolution reproduce children who obtain the adaptive antibody network and dynamically improve their fitness value to about 0.6 0.7 by altering their antibody configuration.

Figure 16 shows the variance of agents' fitness values, how the fitness values are spread around the average. The variance for agents without evolution has not converged well while the variance for agents with evolution has gradually converged. The lower variance implies that all agents' fitness values are close each other. Together with the results in figure 15, figure 16 concludes that the optimal configuration (genes) of antibody network is successfully spread out to other surviving agents by evolutionary process; thus, agents adapts to the environment conditions well through generations.

VII. Related Work

This paper describes several extensions to the existing work on iNet [13], [15]. [13] 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 of agent designers. [15] describes preliminary simulation results of the iNet evolutionary mechanism; however, it does not investigate ICL as well as the EE facility in iNet.

The Bio-Networking Architecture [16] 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, [16] employs a simple weighted sum calculation for behavior selection. Although [16] 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.

BEYONDwork provides visual and textual ICL to configure antibodies (behavior policies) for agents. The work of ICL is parallel to the existing research on domain specific languages (DSLs) [17]. ICL is considered as a DSL in that ICL focuses 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., [18],

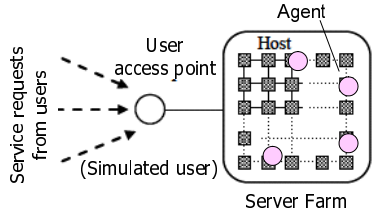

Fig. 11. A Simulated Network

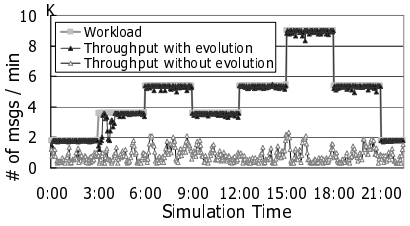

Fig. 12. Workload and Throughput

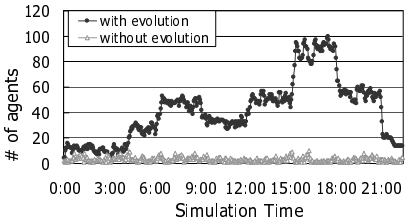

Fig. 13. Population of Agents

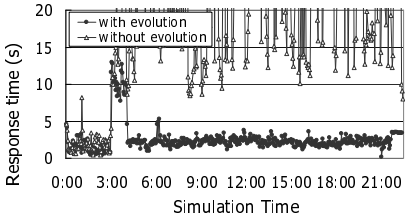

Fig. 14. Response Time

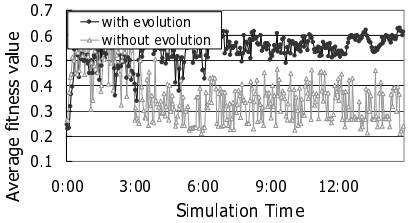

Fig. 15. Average Fitness Value of Agents

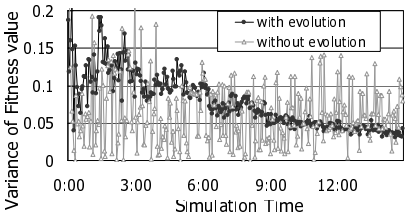

Fig. 16. Variance of Fitness Value

[19]). However, the objective of ICL is different from theirs; ICL aims 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.

VIII. 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 focuses on two key components in BEYOND: (1) an evolutionary adaptation engine, called iNet, for network applications and (2) an application development environment, called BEYONDwork, for the adaptation engine. iNet is designed after the mechanisms behind how the immune system works. It models a set of environment conditions (e.g., network traffic) as an antigen and a behavior of network applications (e.g., migration and reproduction) as an antibody. iNet allows each network application to autonomously sense its surrounding environment conditions (i.e., antigens) and adaptively invoke a behavior (i.e., antibody) suitable for the conditions. The configuration of antibodies evolves via genetic operations such as mutation and crossover. BEYONDwork provides visual and textual languages to configure antigens and antibodies in iNet. The languages increase the ease of specifying and modifying iNet configurations. Simulation results show that iNet allows network applications designed with BEYONDwork to evolve themselves to adapt to changing network environments.

References

[1] P. Dini, W. Gentzsch, M. Potts, A. Clemm, M. Yousif, and A. Polze, "Internet, grid, self-adaptability and beyond: Are we ready?" *IEEE Int'l Workshop on Self-Adaptable and Autonomic Computing Systems*, 2006.

[2] *Report of Workshop on New Visions for Large-scale Networks: Research and Applications*. Large Scale Networking (LSN) Coordinating Group of the Interagency Working Group (IWG) for Information Technology Research and Development (IT R&D), March 2001.

[3] R. Sterritt and D. Bustard, "Towards an autonomic computing environment," *IEEE Int'l Workshop on Database and Expert Systems Applications*, September 2003.

[4] J. Rolia, S. Singhal, and R. Friedrich, "Adaptive internet data centers," *International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet*, July 2000.

[5] S. Ranjan, J. Rolia, E. Knightly, and H. Fu, "Qos-driven server migration for internet data centers," *Int'l Workshop on Quality of Service*, 2002.

[6] 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.

[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*, Feb 2000.

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

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

[12] T. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

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

[14] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle, "Managing energy and server resources in hosting centers," *18th Symposium on Operating Systems Principles*, Oct 2001.

[15] C. Lee and J. Suzuki, "An immunologically-inspired adaptation mechanism for evolvable network applications," in *Proc. of the 4th IEEE Consumer Communications and Networking Conference*, January 2007.

[16] T. Nakano and T. Suda, "Self-organizing network services with evolutionary adaptation," *IEEE Trans. on Neural Networks*, September 2005.

[17] 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.

[18] 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.

[19] F. Kolpakov, "Biouml - framework for visual modeling and simulation biological systems," in *Proc. of Int'l Conference Bioinformatics of Genome Regulation and Structure*, July 2002.