Chapter 1

A Biologically-inspired QoS-aware Architecture for Scalable, Adaptive and Survivable Network Systems

Paskorn Champrasert and Junichi Suzuki

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

Large-scale network systems, such as grid/cloud computing systems, are increasingly expected to be autonomous, scalable, adaptive to dynamic network environments, survivable against partial system failures and simple to implement and maintain. Based on the observation that various biological systems have overcome these requirements, the proposed architecture, SymbioticSphere, applies biological principles and mechanisms to design network systems (i.e., application services and middleware platforms). SymbioticSphere follows key biological principles such as decentralization, evolution, emergence, diversity and symbiosis. Each application service and middleware platform is modeled as a biological entity, analogous to an individual bee in a bee colony, and implements biological mechanisms such as energy exchange, migration, replication, reproduction and death. Each agent/platform possesses behavior policies, as genes, each of which determines when to and how to invoke a particular behavior. Agents and platforms are designed to evolve and adjust their genes (behavior policies) through generations and autonomously improve their scalability, adaptability and survivability. Through this evolution process, agents/platforms strive to satisfy given constraints for quality of service (QoS) such as response time, throughput and workload distribution. This chapter describes the design of SymbioticSphere and evaluates how the biologically-inspired mechanisms in SymbioticSphere impact the autonomy, adaptability, scalability, survivability and simplicity of network systems.

1.1. Introduction

The scale, dynamics, heterogeneity and complexity of network systems have been growing at an enormous rate. The Software Engineering Institute

(SEI) of Carnegie Mellon University and the Department of Defence expect that the growth rate will keep increasing in the future.[1] The authors of the chapter believe that, in the very near future, the capability of network systems will exceed the capacity of humans to design, configure, monitor and understand them. Therefore, network systems need to address new challenges that traditional systems have not considered well; e.g., *autonomy*–the ability to operate with minimal human intervention; *scalability*–the ability to scale to, for example, a large number of users and a large volume of workload; *adaptability*–the ability to adapt to dynamic changes in network conditions (e.g., resource availability and network traffic); *survivability*–the ability to retain operation and performance despite partial system failures (e.g., network host failures); and *simplicity* of implementation and maintenance.

In order to meet these challenges in network systems, the authors of the chapter observe that various biological systems have already developed the mechanisms necessary to overcome those challenges.[2,3] For example, a bee colony is able to scale to a huge number of bees because all activities of the colony are carried out without centralized control. Bees act autonomously, influenced by local environmental conditions and local interactions with nearby 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, most bees remain in the hive and rest. A bee colony can survive massive attacks by predators because it does not depend on any single bee, even on the queen bee. The structure and behavior of each bee are very simple; however, a group of bees autonomously emerges these desirable system characteristics such as adaptability and survivability through collective behaviors and interactions among ants. Based on this observation, the authors of the chapter believe that, if network systems are designed after certain biological principles and mechanisms, they may be able to meet the aforementioned challenges in network systems (i.e., autonomy, scalability, adaptability, survivability and simplicity). Therefore, the proposed architecture, called SymbioticSphere, applies key biological principles and mechanisms to design network systems.

SymbioticSphere consists of two major system components: *application service* and *middleware platform*. Each of them is modeled as a biological entity, analogous to an individual bee in a bee colony. They are designed to follow several biological principles such as decentralization, evolution, emergence, diversity and symbiosis. An application service is implemented as an autonomous software agent. Each agent implements a functional service (e.g., web service) and follows biological behaviors such as energy exchange, migration, replication, reproduction, death and environment sensing. A middleware platform runs on a network host and operates agents. Each platform provides a set of runtime services that agents use to perform their services and behaviors, and implements biological behaviors such as energy exchange, replication, reproduction, death and environment sensing.

Each agent/platform possesses *behavior policies*, each of which determines when to and how to invoke a particular behavior. A behavior policy is encoded as a *gene*. In SymbioticSphere, evolution occurs on behavior policies via genetic operations such as mutation and crossover, which alter behavior policies when agents/platforms replicate themselves or reproduce their offspring. This evolution process is intended to increase the adaptability, scalability and survivability of agents/platforms by allowing them to adjust their behavior policies to dynamic network conditions across generations. Agents/platforms evolve to satisfy given constraints for quality of service (QoS) such as response time, throughput and workload distribution. Each constraint is defined as the upper or lower bound of a QoS measure. Evolution frees network system developers from anticipating all possible network conditions and tuning their agents and platforms to the conditions at design time. Instead, agents and platforms evolve and autonomously adjust themselves to network environments at runtime. This can significantly simplify to implement and maintain agents/platforms.

This chapter describes the design of SymbioticSphere and evaluates the biologically-inspired mechanisms in SymbioticSphere, through simulation experiments, in terms of autonomy, scalability, adaptability and survivability. Simulation results show that agents and platforms evolve in order to autonomously scale to network size and demand volume and adapt to dynamic changes in network conditions (e.g., user locations, network traffic and resource availability). Agents/platforms also evolve to autonomously survive partial system failures such as host failures in order to retain their availability and performance. Moreover, it is verified that agents and platforms satisfy given QoS constraints through evolution.

This chapter is organized as follows. Section 1.2 overviews key biological principles applied to SymbioticSphere. Section 1.3 describes the design of agents and platforms in SymbioticSphere. Section 1.4 discusses a series of simulation results to evaluate SymbioticSphere. Sections 1.5 and 1.6 conclude this chapter with some discussion on related and future work.

1.2. Design Principles in SymbioticSphere

SymbioticSphere applies the following biological principles to design agents and platforms.

- **Decentralization:** In various biological systems (e.g., bee colony), there are no central entities to control or coordinate individual entities for increasing scalability and survivability. Similarly, in SymbioticSphere, there are no central entities to control or coordinate agents/platforms so that they can be scalable and survivable by avoiding a single point of performance bottlenecks[4] and failures.[5]
- **Autonomy:** Inspired by biological entities (e.g., bees), agents and platforms sense their local network conditions, and based on the conditions, they behave and interact with each other without any intervention from/to other agents, platforms and human users.
- **Emergence:** In biological systems, collective (group) behaviors emerge from local interactions of autonomous entities.[3] In SymbioticSphere, agents/platforms only interact with nearby agents/platforms. They behave according to dynamic network conditions such as user demands and resource availability. For example, an agent may invoke the migration behavior to move toward a platform that forwards a large number of request messages for its services. Also, a platform may replicate itself on a neighboring network host where resource availability is high. Through collective behaviors and interactions of individual agents and platforms, desirable system characteristics such as scalability, adaptability and survivability emerge in a group of agents and platforms. Note that they are not present in any single agent/platform.
- **Redundancy:** Biological entities (e.g, bees) die due to, for example, advanced age and consumption by predators. However, biological systems (e.g., bee colony) can survive because the death is compensated by the birth of new entities. In SymbioticSphere, agents/platforms replicate themselves and reproduce their offspring to retain redundancy. This redundancy enhances their survivability; they can continuously provide their services in case that many agents/platforms are lost due to network failures.
- **Lifecycle and Food Chain:** Biological entities strive to seek and consume food for living. In SymbioticSphere, 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 resources such

as memory space (Figure 1.1). Each platform gains energy in exchange for providing resources to agents, and continuously evaporates energy (Figure 1.1). The abundance or scarcity of stored energy in agents/platforms affects their lifecycle. For example, an abundance of stored energy indicates high demand to an agent/platform; thus, the agent/platform may be designed to favor reproduction or replication to increase its availability and redundancy. A scarcity of stored energy indicates a lack of demand; it causes the agent/platform's death.

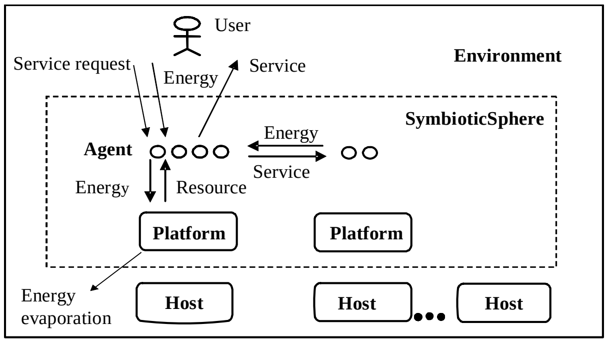

Fig. 1.1. Energy Exchange in SymbioticSphere

Also, in ecosystem, the energy accumulated from food is transferred between different species to balance their populations. For example, producers (e.g., shrubs) convert the Sun light energy to chemical energy. The chemical energy is transferred to consumers (e.g., hares) as consumers consume producers[2] (Figure 1.2). In order to balance the populations of agents and platforms, the energy exchange in SymbioticSphere is designed after ecological food chain among different species. SymbioticSphere models agents and platforms as different biological species; it models human users as the Sun, which have an unlimited amount of energy, agents as producers, and platforms as consumers.

- **Diversity:** Biological entities are slightly different with each other in each species. This can contribute to survivability against environmental changes.[6] In SymbioticSphere, agents/platforms retain *behavioral diversity*. Behavioral diversity means that different agents/platforms have different behavior policies. For example, in response to abundance of energy, an agent may migrate toward a user for reducing response time, while another agent may reproduce offspring with a mate for increasing agent availability/redundancy. Behavioral diversity is generated via

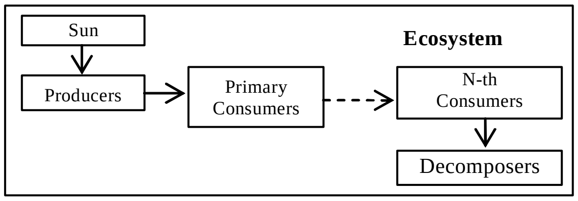

Fig. 1.2. Energy Flow in the Ecosystem

genetic operations (i.e., mutation and crossover) during replication and reproduction.

- **Evolution:** Biological entities evolve so that the entities that fit better to the environment become more abundant.[7] In SymbioticSphere, agents and platforms evolve their genes (i.e., behavior policies) by generating behavioral diversity and executing natural selection. Natural selection is governed with agents' and platforms' energy levels. It retains the agents/platforms whose energy levels are high (i.e., the agents/platforms that have effective behavior policies, such as moving toward a user to gain more energy) and eliminates the agents/platforms whose energy levels are low (i.e., the agents/platforms that have ineffective behavior policies, such as moving too often). Through generations, effective behavior policies become abundant while ineffective ones become dormant or extinct. This allows agents/platforms to adjust their behavior policies to improve their scalability, adaptability and survivability.
- **Symbiosis:** Although competition for food and terrain always occurs in the biological world, several species establish symbiotic relationships to avoid excessive competition and cooperate to survive.[8] In SymbioticSphere, agents and platforms evolve to cooperate in certain circumstances in order to pursue their mutual benefits and improve their scalability, adaptability and survivability.

1.3. SymbioticSphere

In SymbioticSphere, each agent runs (or lives) on a platform. A platform is an execution environment (or middleware) for agents. Each platform implements runtime services that agents use to perform their services and behaviors. Each platform can operate multiple agents, and each network host operates at most one platform.

1.3.1. *Agents*

Each agent consists of three parts: *attributes*, *body* and *behaviors.* Attributes carry descriptive information regarding an agent, such as agent ID, energy level, description of a service the agent provides and cost of a service (in energy unit) that the agent provides.

A body implements a service that an agent provides, and contains materials relevant to the service (e.g., application data and user profiles). For example, an agent may implement a web service and contain web pages. Another agent may implement a physical model for scientific simulations and contain parameter settings for the physical model.

Behaviors implement actions inherent to all agents.

- *Migration*: Agents may migrate from one platform to another.
- *Replication*: Agents may make a copy of themselves. A replicated (child) agent is placed on the platform that its parent agent resides on. It inherits the half of the parent's energy level.
- *Reproduction*: Agents may make their offspring with their mates. A reproduced (child) agent is placed on the platform that its parent agent[a] resides on. It inherits the half of the parent's energy level.
- *Death*: Agents may die due to energy starvation. If the energy expenditure of an agent is not balanced with its energy gain from users and other agents, it cannot pay for the resources it requires. Agents have high chances of dying from lack of energy, if they provide unwanted services and/or have wasteful behavioral policies (e.g., replicating too often). When an agent dies, an underlying platform removes it and releases the resources it consumes.

1.3.2. *Platforms*

Each platform consists of *attributes*, *behaviors*, and *runtime services.* Attributes carry descriptive information regarding a platform, such as platform ID, energy level and *health level.* Health level indicates how healthy an underlying host is. It is defined as a function of resource availability on, age of and freshness of a host. Resource availability indicates how much resources (e.g., memory space) are available for a platform and agents on the host. Age indicates how long a host has been alive. It represents how much stable the host is. Freshness indicates how recently a host joined the network. Once a host joins the network, its freshness gradually decreases

[a]The parent agent is an agent that invokes the reproduction behavior.

from the maximum value. When a host resumes from a failure, its freshness starts with the value that the host had when it went down. Using age and freshness can distinguish unstable hosts and new hosts. Unstable hosts tend to have low freshness and low age, and new hosts tend to have high freshness and low age (Table 1.1).

Table 1.1. Freshness and age

Host type	**Freshness**	**Age**
Unstable host	Low	Low
New host	High	Low
Stable host	Low	High

Health level affects how platforms and agents invoke behaviors. For example, higher health level indicates higher stability of and/or higher resource availability on a host that a platform resides on. Thus, the platform may replicate itself on a neighboring host if the host is healthier than the local host. This results in the adaptation of platform locations. Platforms strive to concentrate around stable and resource-rich hosts. Also, lower health level indicates that a platform runs on a host that is unstable and/or poor in resources. Thus, agents may leave the platform and migrate to a healthier (i.e., more stable and/or resource-rich) hosts. This results in the adaptation of agent locations. Agents strive to concentrate around stable and/or resource-rich hosts. In this case, the platforms on unstable and/or resource-poor hosts will eventually die due to energy starvation because agents do not run on the platforms and transfer energy to them. This results in the adaptation of platform population. Platforms avoid running on the hosts that are unstable and/or poor in resources.

Behaviors are the actions inherent to all platforms.

- *Replication:* Platforms may make a copy of themselves. A replicated (child) platform is placed on a neighboring host that does not run a platform. (Two or more platforms are not allowed to run on each host.) It inherits the half of the parents energy level.
- *Reproduction:* Platforms may make their offspring with their mates. A reproduced (child) platform is placed on a neighboring host that does not run a platform. It inherits the half of the parent's[b] energy level.
- *Death:* Platforms may die due to energy starvation. A dying platform uninstalls itself and releases the resources it consumes. When a platform

[b]The parent platform is a platform that invokes the reproduction behavior.

dies, agents running on it are killed.

Runtime Services are middleware services that agents and platforms use to perform their behaviors. In order to maximize decentralization and autonomy of agents/platforms, they only use the local runtime services. They are not allowed to invoke any runtime services on a remote platform.

1.3.3. ***Behavior Policies***

Each agent/platform possesses policies for its behaviors. A behavior policy defines when to and how to invoke a particular behavior. Each behavior policy consists of factors (F_i), which indicate environment conditions (e.g., network traffic) or agent/platform/host status (e.g., energy level and health level). Each factor is associated with a weight (W_i). Each agent/platform decides to invoke a behavior when the weighted sum of the behavior's factor values ($\sum F_i * W_i$) exceeds a threshold.

1.3.3.1. *Agent Behavior Policies*

This chapter focuses on four agent behaviors described in Section 1.3.1 (i.e., migration, replication, reproduction and death). The behavior policy for agent migration includes the following four factors.

(1) *Energy Level*: Agent energy level, which encourages agents to move in response to high energy level.
(2) *Health Level Ratio*: The ratio of health level on a neighboring platform to the local platform, which encourages agents to move to healthier platforms. This ratio is calculated with three health level properties ($HLP_i, 1 \leq i \leq 3$)): resource availability, freshness and age (Equation 1.1).

$$Helth\ Level\ Ratio = \sum_{i=1}^{3} \frac{HLP_i^{neighbor} - HLP_i^{local}}{HLP_i^{local}} \tag{1.1}$$

$HLP_i^{neighbor}$ and HLP_i^{local} denote a health level property on a neighboring platform and local platform, respectively.
(3) *Service Request Ratio*: The ratio of the number of incoming service requests on a neighboring platform to the local platform. This factor encourages agents to move toward human users.

(4) *Migration Interval*: Time interval to perform migration, which discourages agents to migrate too often.

If there are multiple neighboring platforms that an agent can migrate to, the agent calculates the weighted sum of the above factors for each of them, and moves to a platform that generates the highest weighted sum.

The behavior policy for agent reproduction and replication includes the following two factors.

(1) *Energy Level*: Agent energy level, which encourages agents to reproduce their offspring in response to their high energy levels.
(2) *Request Queue Length*: The length of a queue, which the local platform stores incoming service requests to. This factor encourages agents to reproduce their offspring in response to high demands for their services.

When the weighted sum of the above factors exceeds a threshold, an agent seeks a mate from the local and neighboring platforms. If a mate is found, the agent invokes the reproduction behavior. Otherwise, the agent invokes the replication behavior. Section 1.3.5 describes how an agent seeks its mate for reproduction.

The behavior policy for agent death includes the following two factors:

(1) *Energy Level*: Agent energy level. Agents die when they run out of their energy.
(2) *Energy Loss Rate*: The rate of energy loss, calculated with Equation 1.2. E_t and E_{t-1} denote the energy levels in the current and previous time instants. Agents die in response to sharp drops in demands for their services.

$$Energy\ Loss\ Rate = \frac{E_{t-1} - E_t}{E_{t-1}} \tag{1.2}$$

1.3.3.2. *Platform Behavior Policies*

This chapter focuses on three platform behaviors described in Section 1.3.2 (i.e., reproduction, replication and death).

The behavior policy for platform reproduction and replication includes the following three factors.

(1) *Energy Level:* Platform energy level, which encourages platforms to reproduce their offspring in response to their high energy levels.

(2) *Health Level Ratio:* The ratio of health level on a neighboring host to the local host. This factor encourages platforms to reproduce their offspring on the hosts that generate higher values with Equation 1.1.
(3) *The Number of Agents:* The number of agents working on each platform. This factor encourages platforms to reproduce their offspring in response to high agent population on them.

When the weighted sum of the above factors exceeds a threshold, a platform seeks a mate from its neighboring hosts. If a mate is found, the platform invokes the reproduction behavior. Otherwise, it invokes the replication behavior. Section 1.3.5 describes how a platform finds its mate for reproduction. If there are multiple neighboring hosts that a platform can place its child platform on, it places the child on a host whose health ratio is highest among others.

The behavior policy of platform death includes the following two factors.

(1) *Energy Level*: Platform energy level. Platforms die when they run out of their energy.
(2) *Energy Loss Rate*: The rate of energy loss, calculated with Equation 1.2. Platforms die in response to sharp drops in demands for their resources.

Each agent/platform expends energy to invoke behaviors (i.e., behavior invocation cost) except the death behavior. When the energy level of an agent/platform exceeds the cost of a behavior, it decides whether it performs the behavior by calculating a weighted sum described above.

1.3.4. *Energy Exchange*

As described in Section 1.2, SymbioticSphere models agents and platforms as different biological species and follows ecological concepts to design energy exchange among agents, platforms and human users. Following the energy flow in ecosystem (Figure 1.2), SymbioticSphere models users as the Sun, agents as producers, and platforms as (primary) consumers. Similar to the Sun, users have an unlimited amount of energy. They expend energy units for services provided by agents. Agents gain energy from users and expend energy to consume resources provided by platforms. They expend 10% of the current energy level to platforms[c]. Platforms gain energy

[c]This 10% rule is known in ecology[2] and applied to the energy exchange in SymbioticSphere.

from agents, and periodically evaporate 10% of the current energy level.

Agents dynamically change the frequency to transfer energy to platforms, depending on the rate of incoming service requests from users. When agents receive and process more service requests from users, they consume more resources. Thus, agents transfer energy units (i.e., 10% of the current energy level) to platforms more often. On the contrary, they reduce their energy transfer rate in response to a lower rate of incoming service requests.

In order to dynamically change energy transfer rate, each agent keeps an interval time between arrivals of an incoming service request and a previous request. It records the average, shortest and maximum intervals of previous requests (T_a, T_s and T_m, respectively). Figure 1.3 shows how often each agent transfers energy to an underlying platform. First, an agent waits for T_s and transfer energy to an underlying platform. Then, the agent examines whether a new service request(s) has arrived during the previous T_s interval. If arrived, the agent updates T_a, T_s and T_m, waits for T_a, and transfer energy. Otherwise, it waits for T_a and transfers energy. Similarly, each agent repeats energy transfers in T_a, T_s and T_m intervals (Figure 1.3).

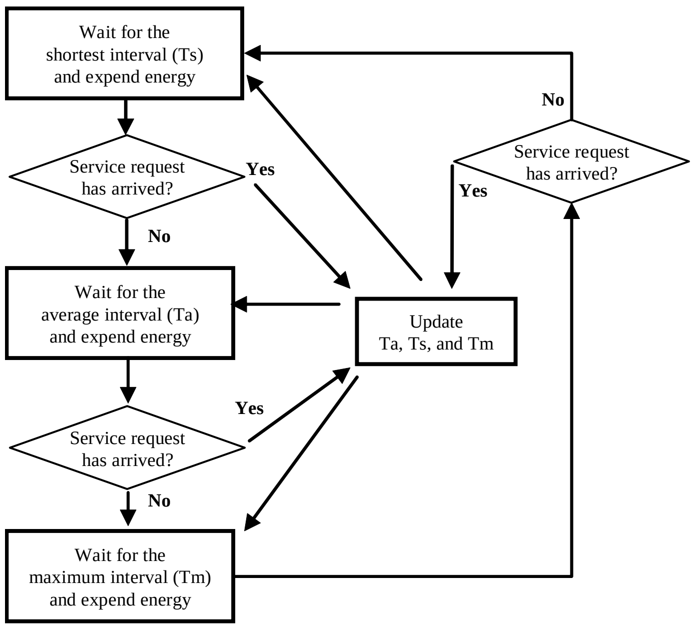

Fig. 1.3. Energy Exchange in SymbioticSphere

T_a is the simple average calculated with the intervals of previous N requests. The shortest and maximum intervals play a role to adjust energy transfer rate according to dynamic changes in request rate. T_s and T_m values are periodically reset (every M service requests).

Platforms dynamically change the frequency to evaporate energy (10% of the current energy level), depending on the rate of incoming energy transfers from agents. The more often they gain energy from agents, the more often they evaporate energy. Each platform changes its energy evaporation rate in the same way as each agent changes its energy expenditure rate; each platform follows the mechanism shown in Figure 1.3.

1.3.5. *Constraint-aware Evolution*

The weight and threshold values in behavior policies have significant impacts on the adaptability, scalability and survivability of agents and platforms. However, it is hard to anticipate all possible network conditions and find an appropriate set of weight and threshold values for the conditions. As shown in Section 1.3.3, there are 18 weight and threshold variables in total (11 for agent behavior policies and 7 for platform behavior policies). Assuming that 10 different values can be assigned to each variable, there are 10^{18} possible combinations of weight and threshold values.

Instead of manual assignments, SymbioticSphere allows agents and platforms to autonomously find appropriate weight and threshold values through evolution, thereby adapting themselves to dynamic network conditions. Behavior policies are encoded as genes of agents and platforms. Each gene contains one or more weight values and a threshold value for a particular behavior. Figures 1.4 and 1.5 show the gene structure for agent and platform behavior policies, respectively. For example, for the agent reproduction behavior, a gene consists of three gene elements: (1) W^a_{r1}, a weight value for the energy level factor; (2) W^a_{r2}, a weight value for the factor of request queue length and (3) T^a_r, a threshold value (Figure 1.4)

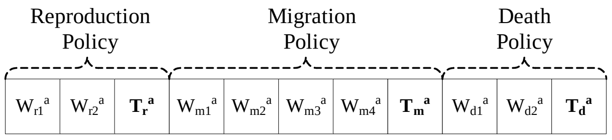

Fig. 1.4. Gene Structure for Agent Behavior Policies

The genes of agents and platforms are altered via genetic operations

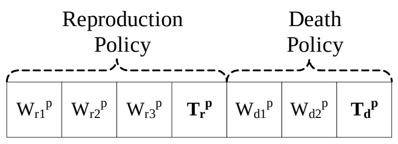

Fig. 1.5. Gene Structure for Platform Behavior Policies

(genetic crossover and mutation) when they perform the reproduction or replication behaviors. As described in Section 1.3.3, each agent/platform selects a mate when it performs the reproduction behavior. A mate is selected by ranking agents/platforms running on the local and neighboring hosts. For this ranking process, SymbioticSphere leverages a *constraint-based domination ranking* mechanism.

Agents and platforms are ranked with the notion of *constraint domination*, which considers two factors: optimization objectives and QoS constraint violation. SymbioticSphere considers the following three objectives. For all objectives, the higher, the better.

(1) *Energy level*
(2) *The total number of behavior invocations*
(3) *Health level of the underlying host*

Using these objectives, domination is determined among agents/platforms. Figure 1.6 shows an example to examine domination among four agents (Agent A to D). For simplicity, this figure shows only two objectives: energy level and the number of behavior invocations. Agents are plotted on a two dimensional space whose axes represent the two objectives. In this example, Agent A is the best in both objectives; it is said to *dominate* the other three agents. In other words, Agent A is *non-dominated.* Agent B is dominated by Agent A; however, it dominates the other two agents (Agent C and D). Agent C and D do not dominate with each other because one of them does not outperform the other in both objectives, and vise versa.

The second factor in the agent/platform ranking process is constraint violation on QoS such as response time, throughput and workload distribution. Each constraint is defined as the upper or lower bound of a QoS measure. For example, a constraint may specify that response time must be lower than 1 second. When an agent/platform satisfies all of given constraints, it is said to be *feasible.* Otherwise, it is said to be *infeasible.*

With the above two factors examined, an agent/platform i is said to *constraint-dominate* another agent/platform j if any of the following con-

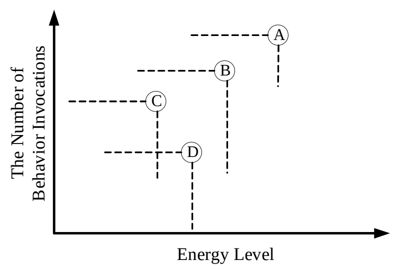

Fig. 1.6. An Example of Agent Domination

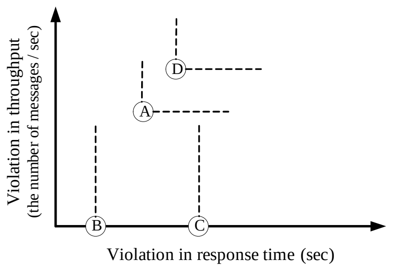
Violation in throughput (the number of messages / sec)
Violation in response time (sec)
D
A
B
C

Fig. 1.7. An Example of Constraint Violation

ditions are true:

(1) i is feasible, and j is not.
(2) Both i and j are feasible; however, i dominates j.
(3) Both i and j are infeasible; however, i has a smaller degree of constraint violation than j.

Figure 1.7 shows an example to evaluate the degree of constraint violations by four agents. (Agent A, B, C and D are all infeasible.) The X and Y axes represent the difference between an actual QoS measure and a QoS constraint in response time and throughput, respectively. Agent A and D violate the constraints for response time and throughput. Agent B and C violate a constraint for response time, but satisfy a constraint for throughput. In this example, Agent B constraint-dominates the other three agents because its violation is minimum in both of two QoS measures. Agents A and C do not constraint-dominate with each other because one of them cannot yield lower violation in both QoS measures, and vice versa. (Agent A yields lower violation in response time but higher violation in throughput than Agent B.) Agent D is constraint-dominated by Agents A and C.

Agents/platforms are ranked with constraint-domination among them. Non-constraint-dominated agents/platforms are given the lowest rank, Rank 1. When an agent/platform is constraint-dominated by a Rank N agent/platform, its rank is $N + 1$.

Figure 1.8 shows pseudo code that shows the evolution process in SymbioticSphere. Table 1.2 shows a set of variables and functions used in the pseudo code.

In reproduction, crossover occurs. A parent and its mate contribute their genes and combine them for a child's genes. The child's gene element

Table 1.2. Variables and Functions used in Figure 1.8

Variable or function	Description
$Agents$	A set of agents at the current simulation cycle.
$Platforms$	A set of platforms at the current simulation cycle.
$parent$	An agent ($parent^a$) or a platform ($parent^p$) that invokes the reproduction behavior.
$mate$	An agent ($mate^a$) or a platform ($mate^p$) that is selected as a mate.
$child$	An agent ($child^a$) or a platform ($child^p$) that is reproduced or replicated.
$child[i]$	The i-th gene element of a child agent/platform ($0 \leq i \leq N$)
$mutationRate$	The probability to perform mutation. 0.1 is currently used.
$FINDMATE(parent)$	Returns a mate for a given parent agent/platform. Returns $\oslash$ if no agents/platforms exist on the local and direct neighbor (one-hop away) hosts. Returns a Rank 1 agent/platform on those hosts if the parent is feasible. Otherwise, returns a feasible agent/platform, on those hosts, which performs best in the QoS whose constraint(s) the parent violates.
$FITNESS(parent)$	Returns the number of agents/platforms that the parent constraint-dominates on the local and direct neighbor hosts.
$U(0,1)$	Generates a random number between 0 to 1 based on the uniform distribution.
$N(\mu, \sigma)$	Generates a random number based on a normal distribution whose average is μ and standard deviation is σ. Currently, 0 and 0.3 are used for μ and σ.

value is in between its parent's and a mate's. It is shifted to closer to its parent's and a mate's, depending on their fitness values. After crossover, mutation may occur on the child's genes. Each gene element is randomly altered based on a uniform distribution function ($U(0,1)$). See Figure 1.9 for an example of crossover and mutation. In replication, a parent copies its genes to its child. Then, mutation may occur on the child's genes in the same way as the mutation in reproduction.

1.4. Evaluation

This section shows a series of simulation results to evaluate the biologically-inspired mechanisms in SymbioticSphere. The objectives of simulations are to examine how the biologically-inspired mechanisms impact the adaptability, scalability and survivability of network systems. Simulations were carried out with the SymbioticSphere simulator, which implements the biologically-inspired mechanisms described in Section 1.3[d].

[d]The current code base of the SymbioticSphere simulator contains 15,200 lines of Java code. It is freely available at http://dssg.cs.umb.edu/projects/SymbioticSphere/.

```
main
 while not the last cycle of a simulation run
   do
     for each $parent^a \in Agents$
       do
         if $parent^a$ invokes the reproduction behavior
           then
             $mate^a \leftarrow$ FindMate($parent^a$)
             if $mate^a \neq \oslash$
               then $child^a \leftarrow$ Reproduce($parent^a, mate^a$)
               else $child^a \leftarrow$ Replicate($parent^a$)
     for each $parent^p \in Platforms$
       do
         if $parent^p$ invokes the reproduction behavior
           then
             $mate^p \leftarrow$ FindMate($parent^p$)
             if $mate^p \neq \oslash$
               then $child^p \leftarrow$ Reproduce($parent^p, mate^p$)
               else $child^p \leftarrow$ Replicate($parent^p$)

procedure Reproduction($parent, mate$)
 $child \leftarrow$ CrossOver($parent, mate$)
 $child \leftarrow$ Mutate($child$)
 return ($child$)

procedure Replication($parent$)
 $child \leftarrow$ Mutate($parent$)
 return ($child$)

procedure CrossOver($parent, mate$)
 for $i \leftarrow 1$ to $N$
   do
     $center_i = (parent[i] + mate[i])/2$
     $offset = \frac{(\text{Fitness}(mate) - \text{Fitness}(parent))}{\text{Fitness}(parent) + \text{Fitness}(mate)} * \frac{|mate[i] - parent[i]|}{2}$
     if $mate[i] \geq parent[i]$
       then $child[i] = center_i + offset$
       else $child[i] = center_i - offset$
 return ($child$)

procedure Mutate($parent$)
 for $i \leftarrow 1$ to $N$
   do
     if $U(0,1) \leq mutationRate$
       then $child[i] = parent[i](1 + N(\mu, \sigma))$
 return ($child$)
```

Fig. 1.8. Evolution Process in SymbioticSphere

1.4.1. *Simulation Configurations*

This section describes the implementation and configuration of the SymbioticSphere simulator. The implementation and configuration are commonly used in all the simulations.

A simulated network system is designed as a server farm. Each agent implements a web service in its body. Its behavior policies are randomly configured at the beginning of each simulation. Figure 1.10 shows a simu-

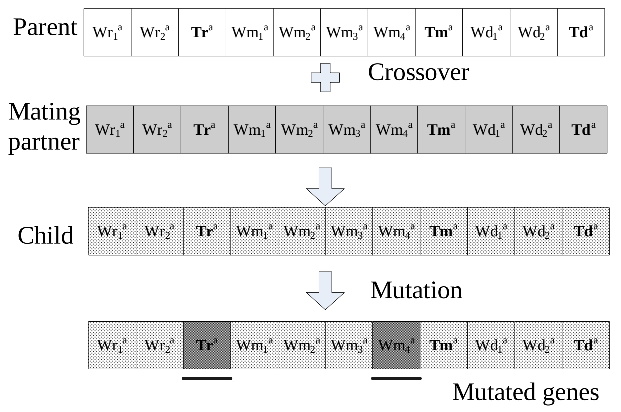

Fig. 1.9. Example Genetic Operations

lated network. It consists of hosts connected in an N x N grid topology, and 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. Each host has 256 MB or 320 MB of memory space[e] Out of the memory space, an operating system consumes 128 MB, and Java virtual machine consumes 64 MB. The remaining space is available for a platform and agents on each host. Each agent and platform consumes 5 MB and 20 MB, respectively. This assumption is obtained from prior empirical experiments.[9]

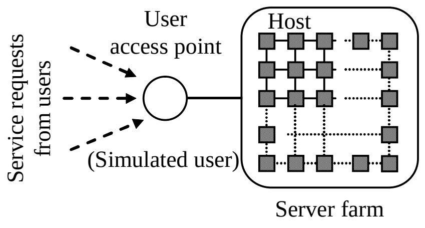

Fig. 1.10. Simulated Network

Each host operates in the active or inactive state. When a platform works on a host, the host is active and consumes 60 W power. The host goes to the inactive state when a platform dies on it. An inactive host consumes 5 W power. This assumption on power consumption is obtained

[e]Currently, memory availability represents resource availability on each platform/host.

from Reference 10. A host becomes active from the inactive state using the Wake On LAN (WOL) technology.[11] When a platform places its offspring on an inactive host, it sends a WOL packet to the host to activate it.

Figure 1.11 shows pseudo code to run users, agents and platforms in each simulation run. A single execution of this while loop corresponds to one simulation cycle.

```
main
 while not the last cycle of a simulation run
  do
    for each user
      do  Send service requests to agents according to a
           configured service request rate.
    for each agent
      do  if a service request(s) received
            then Process the request(s) and gain energy.
          Determine whether or not to invoke the reproduction,
           replication, migration and death behaviors.
          Transfer energy to the local platform.
    for each platform
      do  Gain energy from the local agents
          Determine whether or not to invoke the reproduction,
           replication and death behaviors.
          Update health level.
          Evaporate energy.
```

Fig. 1.11. Pseudo Code to Run Users, Agents and Platforms in a Simulation

When a user issues a service request, the service request is passed to the local platform on which the user resides, and the platform performs a discovery process to search a target agent that can process the issued service request. The platform (discovery originator) forwards a discovery message to its neighboring platforms, asking whether they host a target agent. If a neighboring platform hosts a target agent, it returns a discovery response to the discovery originator. Otherwise, it forwards the discovery message again to its neighboring platforms. Figure 1.12 shows this decentralized agent discovery process. Note that there is no centralized directory to keep track of agent locations.

Through the above discovery process, a user finds a set of platforms hosting the target agents that can process his/her service request. The user chooses the platform closest to him/her and transmits his/her service request to the platform. When the service request arrives the platform, the platform inserts the request into its request queue. (Each platform maintains a request queue for queuing incoming service requests and dispatching them to agents.) Each platform inspects the number of queued

```
while not the last cycle of a simulation run
do
    if a discovery message(s) arrived
    then
        for each discovery message
        do
            if the discovery message matches one of the local agents
            then
                Return a discovery response to the
                discovery originator.
            else
                Forward the discovery message to
                neighboring platforms.
```

Fig. 1.12. Pseudo Code for Agents Discovery Process

service requests and the number of the local agents running on it in each simulation cycle. If the number of queued requests exceeds the number of service requests that the local agents can process in one simulation cycle, the platform transfers the queued requests to neighboring platforms hosting the (idle) agents that can process the requests. This propagation continues until the number of queued requests becomes smaller than the number of service requests that the local agents can process in one simulation cycle. Figure 1.13 shows this request propagation process.

```
while not the last cycle of a simulation run
do
    if # of queued requests > # of requests that the local agents can process
    in one simulation cycle
    then
        if there are one or more neighboring platform(s) that can process
        queued requests
        then Transfer queued requests to those platforms in round robin.
    else Dispatch queued requests to the local agents.
```

Fig. 1.13. Pseudo Code for Service Request Propagation

In this simulation study, SymbioticSphere is compared with the Bio-Networking Architecture (BNA).[9,12,13] In BNA, agents are modeled after biological entities and designed to evolve and adapt dynamic network conditions in a decentralized manner. See also Section 1.5.

1.4.2. *Evaluation of Energy Exchange*

This section evaluates SymbioticSphere's energy exchange mechanism described in Section 1.3.4. In this evaluation, an agent is deployed on a platform to accept service requests from the user. The agent does not invoke any behaviors to focus on the evaluation of energy exchange. T_s and T_m values are periodically reset every 50 service requests.

Figure 1.14 shows how the user changes service request rate over time. In order to evaluate the energy exchange in SymbioticSphere, two agents are implemented and compared. The first agent implements the energy exchange mechanism described in Section 1.3.4 It uses T_a, T_s and T_m intervals. The second agent uses T_a only.

Figure 1.15 shows how much energy the two agents transfer to their local platforms. It demonstrates that the SymbioticSphere's energy exchange mechanism allows an agent to change its energy expenditure rate against dynamic changes in energy intake (i.e., service request rate). With T_a, T_s and T_m, an agent's energy expenditure better follows the changes in energy intake than using T_a only.

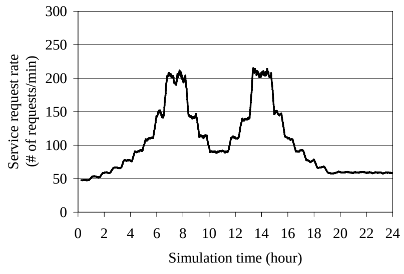

Fig. 1.14. Service Request Rate

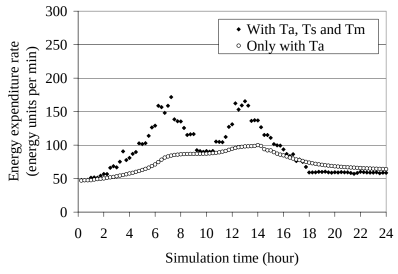

Fig. 1.15. Energy Expenditure Rate

1.4.3. *Evaluation of Adaptability*

This section evaluates how agents and platforms adapt to dynamic network conditions. In this evaluation, adaptability is defined as *service adaptation* and *resource adaptation*. Service adaptation is a set of activities to adaptively improve the quality and availability of services provided by agents. The quality of services is measured as response time and throughput of agents for processing service requests from users. Service availability is measured as the number of agents. Resource adaptation is a set of activities to adaptively improve resource availability and resource efficiency. Resource availability is measured as the number of platforms that make resources available for agents. Resource efficiency indicates how many service requests can be processed per resource utilization of agents and platforms.

A simulated network is a 7x7 network with 49 network hosts, each of which has 256 MB memory space. At the beginning of each simulation, an agent and a platform are deployed on each node (i.e., 49 platforms and 49 agents on 49 network hosts). Figure 1.16 shows how the user changes service request rate over time. This is obtained from a workload trace of the

1998 Olympic official website.[14] The peak demand is 9,600 requests/min. Each simulation was carried out for 10 days in simulation time by repeating the daily workload trace 10 times.

Figure 1.17 shows how service availability (i.e., the number of agents) changes against dynamic service request rate. At the beginning of a simulation, the number of agents fluctuates in SymbioticSphere and BNA because agent behavior policies are initially random. However, as evolution continues over time, agents adapt their population to the changes in service request rate. When service request rate becomes high, agents gain more energy from a user and replicate themselves more often. In contrast, when service request rate becomes low, some agents die due to energy starvation because they cannot balance their energy gain and expenditure. This result demonstrates that the biologically-inspired mechanisms in SymbioticSphere allow agents to evolve their behavior policies and adaptively adjust their availability as a group. Figure 1.17 also shows that agent population are more sensitive to workload changes in SymbioticSphere than BNA.

Figure 1.18 shows how resource availability (i.e., the number of platforms) changes against dynamic service request rate. At the beginning of a simulation, the number of platforms fluctuates in SymbioticSphere because platform behavior policies are initially random. However, as evolution continues over time, platforms adapt their population to the changes in service request rate. When service request rate becomes high, agents gain more energy and transfer more energy to platforms. In response to abundance of stored energy, platforms replicate or reproduce offspring more often. In contrast, when service request rate becomes low, some platforms die due to energy starvation because they cannot gain enough energy from agents to keep their population. Figure 1.18 demonstrates that the biologically-inspired mechanisms in SymbioticSphere allow platforms to evolve their behavior policies and adaptively adjust resource availability as a group. In BNA, platforms are not designed as biological entities; the number of platforms does not change dynamically, but remains at 49.

Figure 1.19 shows the average response time for agents to process one service request from the user. This includes the request transmission latency between the user and an agent and the processing overhead for an agent to process a service request. In SymbioticSphere, the average response time is maintained very low; less than 5 seconds, because agents increase their population to process more service requests (see Figure 1.17) and migrate toward the user. Figure 1.19 demonstrates that the biologically-inspired mechanisms in SymbioticSphere allow agents to autonomously improve and maintain response time. In BNA, agents fail to consistently maintain low response time in several days because they do not adapt their population well to the changes in service request rate (Figure 1.17). Agent evolution process operates better in SymbioticSphere than BNA. See Section 1.5 for the differences in the evolution process in the two architectures.

Figure 1.20 shows the throughput of agents. Throughput is measured as the ratio of the number of service requests agents process to the total number of service requests the user issues. In SymbioticSphere, throughput is maintained very high; i.e., higher than 98%, because agents adapt their population to dynamic service request rate (Figure 1.17). Figure 1.20 demonstrates that the biologically-inspired mechanisms in SymbioticSphere allow agents to autonomously improve and maintain throughput. In BNA, agents fail to consistently maintain high throughput in several days. For example, throughput drops to 84% in Day 6. This is because agents do not adapt their population well to the changes in service request rate. Agent evolution process operates better in SymbioticSphere than BNA. See Section 1.5 for the differences in the evolution process in the two architectures.

Figure 1.21 shows how resource efficiency changes. Resource efficiency is measured with the following equation:

$$Resource\ efficiency = \frac{\text{The total number of service requests processed by agents}}{\text{The total amount resources consumed by agents and platforms}} \tag{1.3}$$

SymbioticSphere always yields higher resource efficiency than BNA because, in BNA, platforms do not adapt their population to dynamic service request rate; 49 platforms are active at all times (Figure 1.18). In SymbioticSphere, both agents and platforms autonomously adapt their populations to dynamic service request rate (Figures 1.17 and 1.18); thus, resource efficiency is always higher in SymbioticSphere. Figure 1.21 demonstrates that the biologically-inspired mechanisms in SymbioticSphere allows agents and platforms to retain high resource efficiency.

Figure 1.22 shows the total power consumption in SymbioticSphere and BNA. In SymbioticSphere, each host operates in active or inactive state depending on whether a platform works on the host (Section 1.4.1). Since platforms adapt their population to dynamic service request rate (Figure 1.18), hosts change their states between active and inactive based on the changes in service request rate. More hosts become active in response to higher service request rate, and more hosts become inactive in response to lower service request rate. In BNA, all hosts are always active because platforms never die. As a result, SymbioticSphere consumes a lower amount of power(443.1 kWh) than BNA does (705.6 kWh). SymbioticSphere saves approximately 40% power consumption compared with BNA. Figure 1.22 shows that the biologically-inspired mechanisms in SymbioticSphere allows platforms to improve power efficiency by adapting their population.

Figure 1.23 shows workload distribution over available platforms in SymbioticSphere. It is measured as Load Balancing Index (LBI) with Equation 1.4. A lower LBI indicates a higher workload distribution.

$$Load\ Balancing\ Index = \sqrt{\frac{\sum_i^N (X_i - \mu)^2}{N}} \tag{1.4}$$

where

$$\begin{aligned} X_i &= \frac{\text{The number of messages processed by agents running on platform } i}{\text{The amount of resources utilized by platform } i \text{ and agents running on platform } i} \\ \mu &= \text{The average of } X_i \\ &= \frac{\text{The total number of messages processed by all agents}}{\text{The total amount of resources utilized by all platforms and all agents}} \\ N &= \text{The number of available platforms} \end{aligned}$$

As Figure 1.23 shows, agents and platforms always strive to improve workload distribution through evolution. This is an example of *symbiotic emergence*, a unique property that SymbioticSphere exhibits. in SymbioticSphere. Agent migration behavior policy encourages agents to move towards platforms running on healthier hosts. Platform replication behavior policy encourages platforms to replicate themselves on healthier hosts. As a result, service requests are processed by agents that are spread over the platforms running on healthy hosts. This contributes to balance workload per platform, although agent migration policy and platform replication policy do not consider platform population nor workload distribution. This results in a mutual benefit for both agents and platforms. Platforms help agents decrease response time by making more resources available for them. Agents help platforms to keep their stability by distributing workload on them (i.e., by avoiding excessive resource utilization on them).

Figures 1.24 and 1.25 show the response time results with and without a QoS constraint. The constraint specifies 5 seconds as maximum (or worst) response time. Figure 1.24 shows that the constraint contributes to improve the average response time from 1.95 to 1.66 second. Figure 1.25 shows the number of constraint violations (i.e., the number of simulation cycles in which actual response time exceeds a given constraint). With

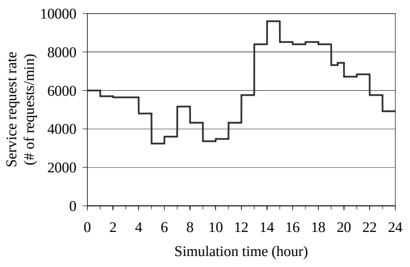

Fig. 1.16. Service Request Rate

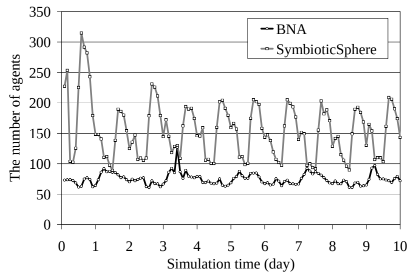

Fig. 1.17. The Number of Agents

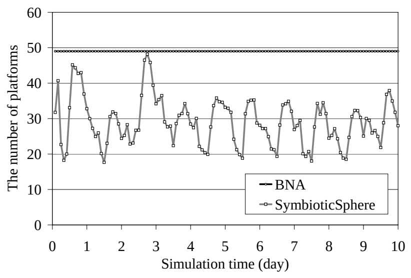

Fig. 1.18. The Number of Platforms

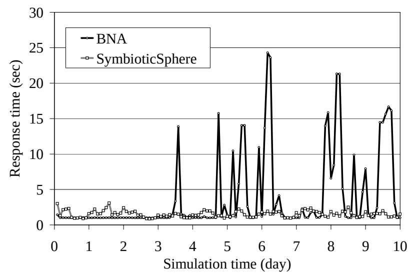

Fig. 1.19. Response Time

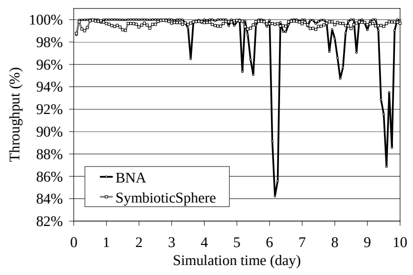

Fig. 1.20. Throughput

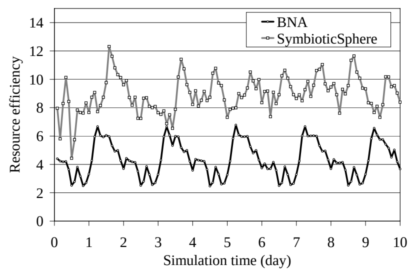

Fig. 1.21. Resource Efficiency

a constraint enabled, the number of constraint violations dramatically reduces by 90% (from 849 to 78). This is because it is unlikely agents are selected as mates when they violate a constraint. SymbioticSphere perform its evolution process to better satisfy a response time QoS constraint.

Figures 1.26 and 1.27 show the LBI results with and without a QoS constraint. The constraint specifies 30% as the maximum (or worst) difference in LBI of an agent and another running on a neighboring platform.

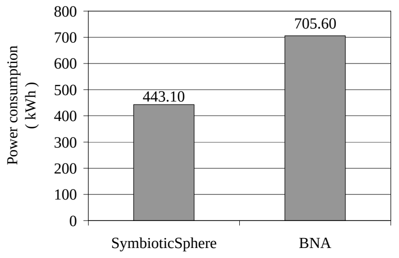

Fig. 1.22. Power Consumption

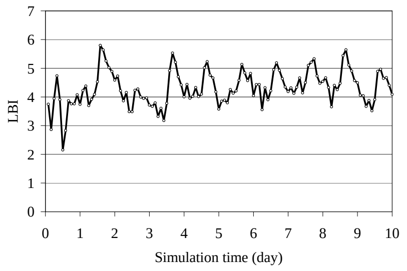

Fig. 1.23. Load Balancing Index (LBI)

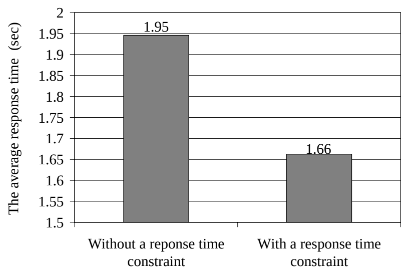

Fig. 1.24. The Average Response Time with and without a Constraint

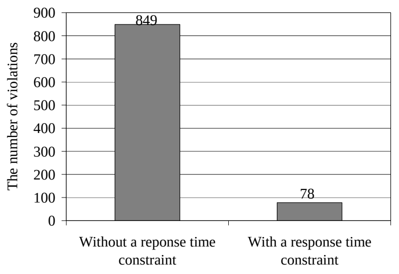

Fig. 1.25. The Number of Constraint Violations in Response Time

Figure 1.26 shows that the constraint contributes to improve LBI from 4.31 to 3.96. Figure 1.27 shows the number of constraint violations reduces by 70% (from 4841 to 1336). This is because it is unlikely agents are selected as mates when they violate a constraint. SymbioticSphere perform its evolution process to better satisfy an LBI QoS constraint.

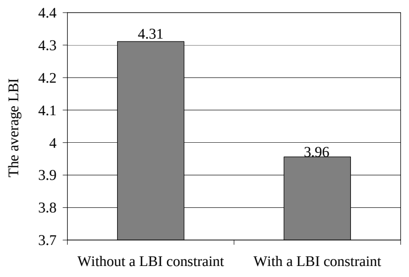

Fig. 1.26. The Average LBI with and without a Constraint

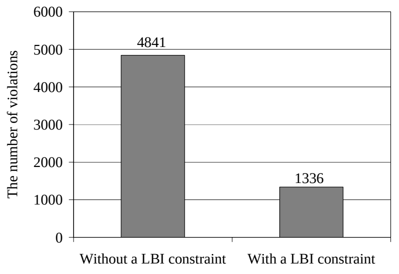

Fig. 1.27. The Number of Constraint Violations in LBI

Figures 1.28 and 1.29 show the throughput results with and without a

QoS constraint. The constraint specifies 3 messages/sec as the minimum (or worst) throughput. Figure 1.28 shows that the constraint contributes to improve the average throughput from 99.50% to 99.92%. Figure 1.29 shows the number of constraint violations reduces by 70% (from 6554 to 1992). This is because it is unlikely agents are selected as mates when they violate a constraint. SymbioticSphere perform its evolution process to better satisfy a throughput QoS constraint.

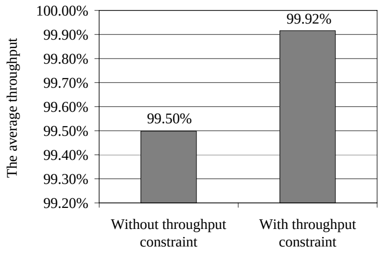

Fig. 1.28. The Average Throughput with and without a Constraint

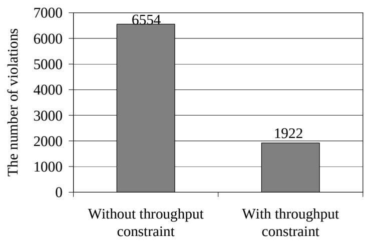

Fig. 1.29. The Number of Constraint Violations in Throughput

1.4.4. *Evaluation of Scalability*

This section evaluates how agents and platforms autonomously scale to demand volume and network size. In this simulation study, service request rate starts with 3,000 requests/min, spikes to 210,000 requests/min at 8:00, and drops to 3,000 requests/min at 16:30 (Figure 1.30). The peak demand and spike ratio (1:70) are obtained from a workload trace of the 1998 World Cup web site.[15] A simulated network is 7x7 (49 hosts) from 0:00 to 12:00 and 15x15 (225 hosts) from 12:00 to 24:00. This simulates that new hosts are added to a server farm at 12:00. 30% of the hosts (resource rich hosts) have 320 MB memory space. The other 70% are resource poor hosts, which have 256 MB memory space. Both types of hosts are placed randomly in a server farm. At the beginning of a simulation, a single agent and platform is deployed on each host (i.e., 49 agents and 49 platforms on 49 hosts). This simulation study uses the genes (i.e., behavior policies) obtained after a 10 days simulation in Section 1.4.3 as the initial genes for each agent and platform.

Figure 1.31 shows how the number of agents changes against the dynamic changes in service request rate and network size. At the beginning of a simulation, the number of agents is about 50 agents in SymbioticSphere and BNA because this number of agents is enough to process all service requests (3,000 requests/min). When service request rate spikes at 8:00, agents gain more energy from the user and reproduce/replicate offspring more often. From 10:00 to 12:00, agent population does not grow due to resource limitation on available hosts. (Agents and platforms cannot reproduce/replicate any more offspring because they have consumed all the resources available on 49 hosts.) When network size expands at 12:00, agents rapidly reproduce/replicate offspring on newly added hosts. When service request rate drops at 16:30, many agents die due to energy starvation. This result demonstrates that the biologically-inspired mechanisms in SymbioticSphere allow agents to adaptively adjust their availability to dynamic network conditions such as demand volume, spike ratio and network size. Figure 1.31 also shows that agent population follows workload changes better in SymbioticSphere than BNA.

Figure 1.32 shows how the number of platforms changes against the dynamic changes in service request rate and network size. In SymbioticSphere, at the beginning of a simulation, the number of platforms is about 7 platforms because this number of platforms is enough to run agents for processing service requests. When service request rate spikes at 8:00, agents gain more energy from the user and transfer more energy to underlying platforms. As a result, platforms also increase their population. From 10:00 to 12:00, platform population does not grow because platforms have already run on 49 hosts. When network size expands at 12:00, platforms rapidly reproduce offspring on newly added hosts. When service request rate drops at 16:30, most platforms die due to energy starvation. This result demonstrates that the biologically-inspired mechanisms in SymbioticSphere allow platforms to adaptively adjust their availability to dynamic network conditions such as demand volume, spike ratio and network size. In BNA, platforms are not designed as biological entities; they do not adaptively change their population.

Figure 1.33 show how the average response time changes. In SymbioticSphere, from 0:00 to 8:00, the average response time is maintained very low; less than 5 seconds. At 8:00, response time spikes because service request rate spikes. It starts decreasing at 12:00 when agents/platforms reproduce/replicate offspring on newly added hosts. It is approximately 12 seconds at 16:00. At 16:30, response time drops because service request rate drops. Figure 1.33 demonstrates that the biologically-inspired mechanisms in SymbioticSphere allow agents and platforms to strive to keep response time low despite demand surges. In BNA, agents do not adapt their population well to the changes in service request rate and network size (Figure 1.31); response time is rather high, about 24 seconds, at 16:00.

Figure 1.34 shows how throughput changes. In SymbioticSphere, by changing their populations, agents and platforms autonomously adapt throughput to dynamic changes in demand (at 8:00 and 16:30) and network size (at 12:00). From 0:00 to 8:00, throughput is maintained very high; i.e., more than 98%. At 8:00, throughput drops (lower than 20%) because the service request rate is very high and agents cannot process all service requests in a timely manner. Then, agents and platforms improve throughput by increasing their populations. Until 12:00, throughput cannot reach 100% due to resource limitation on available hosts. After 12:00, throughput improves to 100% because agents and platforms can replicate/reproduce offspring on newly added hosts. Figure 1.34 shows that the biologically-inspired mechanisms in SymbioticSphere allows agents and platforms to scale well to demand volume, spike ratio and network size. SymbioticSphere and BNA yield similar throughput from 0:00 to 12:00. From 12:00 to 16:30, BNA does not leverage newly added hosts well to improve throughput; throughput never reach 100%.

Figure 1.35 shows how resource utilization is distributed over available hosts in SymbioticSphere. It is measured as Resource Utilization balancing index (RUBI) with Equation 1.5. A lower RUBI indicates a higher distribution of resource utilization.

$$Resource\ Utilization\ Blancing\ Index = \sqrt{\frac{\sum_i^N (R_i - \mu)^2}{N}} \tag{1.5}$$

$$
\begin{aligned}
&where\\
R_i &= \frac{\text{The amount of resources a platform and agents utilize on the host } i}{\text{The total amount of resources the host } i \text{ has}}\\
\mu &= \text{The average of } R_i\\
&= \frac{\text{The total amount of resources utilized by all agents and platforms}}{\text{The total amount of resources available on all hosts}}\\
N &= \text{The number of available platforms}
\end{aligned}
$$

From 0:00 to 8:00, RUBI is relatively constant around 0.2. When service request rate spikes at 8:00, RUBI drops and constantly remains very low, around 0.02, because available resources are fully consumed on hosts. At 12:00, RUBI spikes because resources are not consumed first on newly added nodes. Agents and platforms spread over both resource rich and poor hosts to deal with a high service request rate. Then, they seek resource rich hosts through migration, reproduction and replication, thereby lowering RUBI. When service request rate drops at 16:30, RUBI spikes because many agents and platforms die and resource utilization is not evenly distributed over hosts. However, they decrease RUBI again by preferentially residing on resource rich hosts. Figure 1.35 shows that the biologically-inspired mechanisms in SymbioticSphere allows agents and platforms to adaptively balance their resource utilization over a large number of heterogeneous hosts.

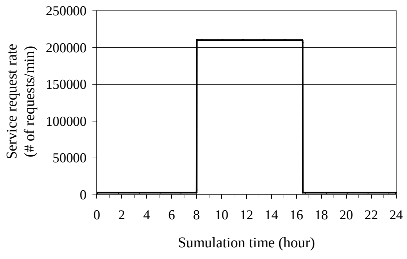

Fig. 1.30. Service Request Rate

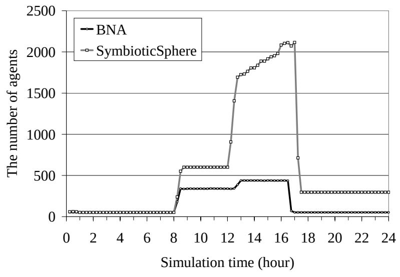

Fig. 1.31. The Number of Agents

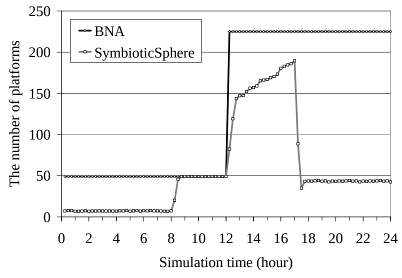

Fig. 1.32. The Number of Platforms

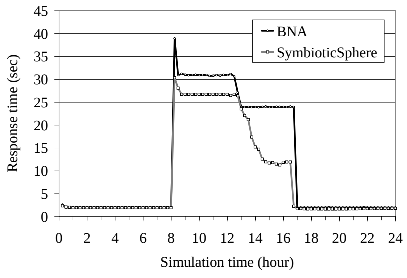

Fig. 1.33. Response Time

1.4.5. *Evaluation of Survivability*

This section evaluates how agents and platforms survive partial system failures due to, for example, errors by administrators and physical damages in server farm fabric This simulation study simulates node failures. Service request rate is constantly 7,200 requests/min, which is the peak in a workload trace of the IBM web site in 2001.[16] Randomly chosen 60% of hosts go down at 9:00 for 90 minutes. The size of a server farm is 7x7, 49 hosts, each of which has 256 MB memory space. At the beginning of a simulation,

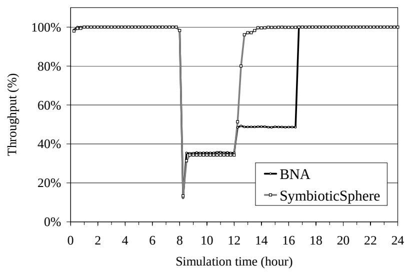

Fig. 1.34. Throughput

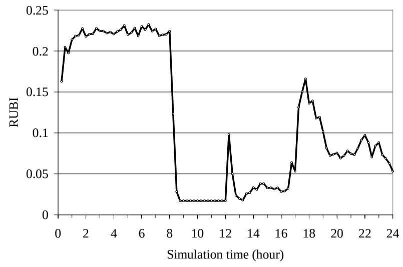

Fig. 1.35. Resource Utilization Balancing Index

an agent and a platform are deployed on each host. This simulation study uses the genes (i.e., behavior policies) obtained after a 10 days simulation in Section 1.4.3 as the initial genes for each agent and platform.

Figures 1.36 and 1.37 show how the number of agents and the number of platforms change over time, respectively. When a host goes down, agents and platforms crash and die on the host. This is why agent and platform populations drop at 9:00. In SymbioticSphere, after hosts fail, remaining agents and platforms increase their populations with replication and reproduction on available hosts. Around 9:45, agent and platform populations revert to the populations that agents and platforms had before host failures. When failed hosts resume, platforms reproduce/replicate offspring on those hosts. Some of agents migrate to the reproduced/replicated platforms in order to move towards the user and increase their population. The biologically-inspired mechanisms in SymbioticSphere allow agents and platforms to autonomously survive host failures by adjusting their populations. Figure 1.36 also demonstrates that agent population recovers faster in SymbioticSphere than BNA. This is because BNA does not consider workload distribution. All agents migrate towards the user; when hosts fail, most of them die. SymbioticSphere better considers survivability. In BNA, platforms are not designed as biological entities; the number of platforms is always equal to the number hosts (Figure 1.37).

Figure 1.38 shows the average response time. At 9:00, response time increases because all agents and platforms die on failed hosts. However, response time quickly recovers in SymbioticSphere by increasing agent and platform populations immediately (Figures 1.36 and 1.37). The biologically-inspired mechanisms in SymbioticSphere allow agents and platforms to autonomously survive host failures and strive to retain response time. In BNA, response time recovers when failed hosts resume at 10:30.

SymbioticSphere is more resilient against host failures than BNA.

Figure 1.39 shows the throughput of agents. Throughput decreases when hosts fail at 9:00; however, it decreases only by 10% in SymbioticSphere. Upon host failures, throughput quickly recovers by increasing agent and platform populations immediately (Figures 1.36 and 1.37). Figure 1.39 demonstrates that the biologically-inspired mechanisms in SymbioticSphere allow agents and platforms to autonomously retain throughput despite host failures. In BNA, throughput dramatically drops to 40% and does not recover until failed hosts resume.

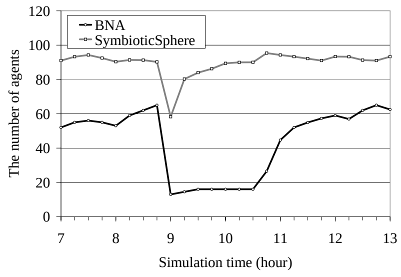

Fig. 1.36. The Number of Agents

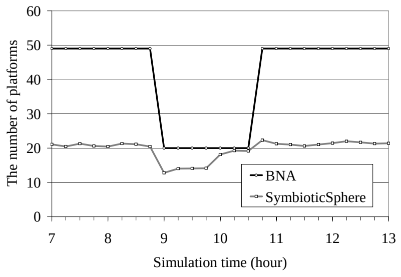

Fig. 1.37. The Number of Platforms

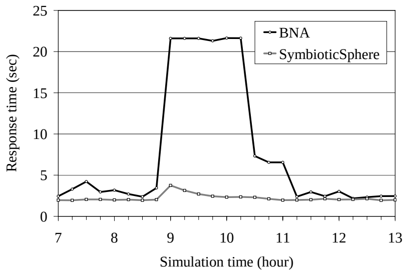

Fig. 1.38. Response Time

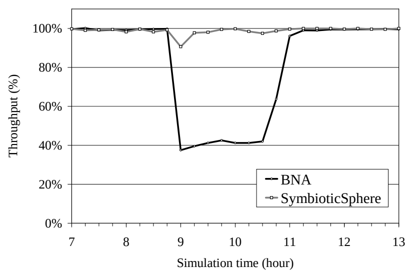

Fig. 1.39. Throughput

1.5. Related Work

SymbioticSphere is an extension to the Bio-Networking Architecture (BNA).[9,12,13] In BNA, biologically-inspired agents evolve and perform service adaptation in a decentralized manner. However, as demonstrated in Section 1.4, SymbioticSphere most always outperforms BNA. This is because the evolutionary process is designed more sophisticated in Symbiotic-

Sphere. For example, mate selection, mutation and crossover are carefully designed to better address real-value optimization to tune behavior policies. Also, in BNA, the threshold values of behavior policies are not included in genes.[12] This means that agent designers need to manually configure them through trial and errors. In contrast, no manual work is necessary to configure thresholds in SymbioticSphere because they are included in genes. In addition, BNA uses a fitness function to rank agents in mate selection. It aggregates multiple objectives as a weighted sum. Agent designers need to manually configure these weight values as well. In SymbioticSphere, no parameters exist for ranking agents/platforms because of a constraint-domination ranking mechanism. As a result, SymbioticSphere incurs much less configuration tasks/costs.

Moreover, BNA does not perform resource adaptation because platforms are not designed as biological entities. In SymbioticSphere, both agents and platforms are designed as biological entities; they perform service adaptation and resource adaptation simultaneously.

Jack-in-the-Net (Ja-Net),[17,18] NetSphere[19] and BEYOND[20] are similar to SymbioticSphere in that they extend BNA. Ja-Net focuses on spontaneous creation of network applications, each of which consists of multiple types of agents. NetSphere and BEYOND propose artificial immune systems for agents to sense network conditions and adaptively select behaviors suitable to the conditions. In Ja-Net, NetSphere and BEYOND, platforms are not designed as biological entities; they do not address power efficiency, resource efficiency and resource utilization balancing of network systems.

Wakamiya et al. propose the concept of symbiosis between groups of peers (hosts) in peer-to-peer networks.[21] Peer groups symbiotically connect or disconnect with each other to improve the speed and quality of queries. A special type of peers implements the symbiotic behaviors for peer group connection/disconnection. Since the number of the symbiotic peers is statically fixed, they do not scale to network size and traffic volume. They also do not address power efficiency, resource efficiency and survivability of network systems. In SymbioticSphere, all agents and platforms are designed to interact in a symbiotic manner. They scale well to network size and traffic volume, and achieve power efficiency, resource efficiency and survivability.

The concept of energy in SymbioticSphere is similar to money in economy. MarketNet[22] and WALRAS[23] apply the concept of money to address market-based access control for network applications. Rather than access control, SymbioticSphere focuses on the adaptability, scalability and survivability of network systems.

Resource Broker is designed to dynamically adjust resource allocation for server clusters via centralized system monitor.[24] Resource Broker inspects the stability and resource availability of each host, and adjusts resource allocation for applications. In SymbioticSphere, agents and platforms adapt their populations and locations in a decentralized way. Also, Resource Broker does not consider the power efficiency of server clusters.

Shirose et al. propose a generic adaptation framework for grid computing systems.[25] It considers both service adaptation and resource adaptation. In this framework, centralized system components store the current environment conditions, and decide which adaptation strategy to execute. In contrast, SymbioticSphere does not assume any centralized system components. Each of agents and platforms collects and stores environment conditions and autonomously decides which behaviors to invoke.

Rainbow[26] investigates the adaptability of server clusters. A centralized system monitor periodically inspects the current environment conditions (e.g., workload placed on hosts) and performs an adaptation strategy (e.g., service migration and platform replication/removal). SymbioticSphere implements a wiser set of adaptation strategies such as agent replication/reproduction (service replication/reproduction) and agent death (service removal). SymbioticSphere also addresses survivability and power efficiency as well as adaptability with the same set of agent/platform behaviors. Rainbow does not consider power efficiency and survivability.

Adam et al. propose a decentralized design for server clusters to guarantee response time to users.[27] In SymbioticSphere, agents and platforms evolve to satisfy given QoS constraints including response time; however, they do not guarantee QoS measures because the dynamic improvement of those measures is an emergent result from collective behaviors and interactions of agents and platforms. As a result, agents and platforms can adapt to unexpected environment conditions (e.g. system failures) and survive them without changing any behaviors and their policies. Moreover, Adam et al. do not consider to satisfy other QoS constraints such as throughput as SymbioticSphere does. They also do not consider consider power efficiency and survivability of network systems as SymbioticSphere does.

1.6. Concluding Remarks

This chapter describes the architectural design of SymbioticSphere and describes how it implements key biological principles, concepts and mechanisms to design network systems. This chapter also describes how agents and platforms interact with each other to collectively exhibit the emergence of desirable system characteristics such as adaptability, scalability, and survivability. Simulation results show that agents and platforms scale well to network size and demand volume and autonomously adapt to dynamic changes in the network (e.g., user location, network traffic and resource availability). They also survive partial system failures such as host failures to retain their availability and performance.

Several extensions to SymbioticSphere are planned. In order to further explore the impacts of symbiosis between agents and platforms on their performance, it is planned to implement and evaluate new types of behaviors, called symbiotic behaviors, for agents and platforms. It is also planned to deploy and empirically evaluate SymbioticSphere on the real network such as PlanetLab[f].

References

1. L. Northrop, R. Kazman, M. Klein, D. Schmidt, et al. Ultra-Large Scale Systems: The Software Challenge of the Future. Technical report, Software Engineering Institute.
2. R. Alexander, *Energy for Animal Life.* (Oxford University Press, 1999).
3. S. Camazine, N. Franks, J. Sneyd, E. Bonabeau, J. Deneubourg, and G. Theraula, *Self-Organization in Biological Systems.* (Princeton University Press, 2003).
4. R. Albert, H. Jeong, and A. Barabasi, Error and attack tolerance of complex networks, *International Journal of Nature.* **406**(6794), 378–382, (2000).
5. N. Minar, K. Kramer, and P. Maes, *Cooperating Mobile Agents for Dynamic Network Routing*, In ed. A. Hayzelden, *Software Agents for Future Communications Systems*, chapter 12. Springer, (1999).
6. S. Forrest, A. Somayaji, and D. Ackley. Building diverse computer systems. In *Proceedings of the 6th Workshop on Hot Topics in Operating System*, pp. 67–72, (1997).
7. P. Stiling, *Ecology. theories and applications.* (Prentice-Hall, 2002).
8. L. Margulis, *Symbiotic Planet: A New Look at Evolution.* (Basic Books, 1998).
9. J. Suzuki and T. Suda, A middleware platform for a biologically inspired network architecture supporting autonomous and adaptive applications, *IEEE Journal on Selected Areas in Communications.* **23**(2), 249–260, (2005).
10. C. Gunaratne, K. Christensen, and B. Nordman, Managing energy consumption costs in desktop PCs and LAN switches with proxying, split TCP connections, and scaling of link speed, *International Journal of Network Management.* **15**(5), 297–310, (2005).

[f]http://www.planet-lab.org/

11. G. Gibson, Magic Packet Technology, *Advanced Micro Devices(AMD).* (1995).
12. T. Nakano and T. Suda, Self-organizing network services with evolutionary adaptation, *IEEE Transactions on Neural Networks.* **16**(5), 1269–1278, (2005).
13. T. Suda, T. Itao, and M. Matsuo. The Bio-Networking Architecture: The Biologically Inspired Approach to the Design of Scalable, Adaptive, and Survivable/Available Network Applications. In ed. K. Park, *The Internet as a Large-Scale Complex System.* Princeton University Press, (2005).
14. C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Ki-stler, and T. Keller, Energy Management for Commercial Servers, *IEEE Journal on Computer.* **36**(12), 39–47, (2003).
15. M. Arlitt and T. Jin, A Workload Characterization Study of the 1998 World Cup Web Site, *IEEE Journal on Network.* **14**(3), 30–37, (2000).
16. J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing energy and server resources in hosting centers. In *Proceedings of the 18th ACM Symposium on Operating System Principles*, pp. 103–116. ACM, (2001).
17. T. Itao, S. Tanaka, T. Suda, and T. Aoyama, A Framework for Adaptive UbiComp Applications Based on the Jack-in-the-Net Architecture, *International Journal of Wireless Networks.* **10**(3), 287–299, (2004).
18. T. Itao, T. Nakamura, M. Matsuo, T. Suda, and T. Aoyama. Adaptive Creation of Network Applications in the Jack-in-the-Net Architecture. In *Proceedings of the 2nd IFIP-TC6 Conference on Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; and Mobile and Wireless Communications*, pp. 129–140, (2002).
19. J. Suzuki, Biologically-inspired adaptation of autonomic network applications, *International Journal of Parallel, Emergent and Distributed Systems.* **20**(2), 127–146, (2005).
20. C. Lee, H. Wada, and J. Suzuki. Towards a biologically-inspired architecture for self-regulatory and evolvable network applications. In eds. F. Dressler and I. Carreras, *Advances in Biologically Inspired Information Systems: Models, Methods and Tools*, chapter 2, pp. 21–45. Springer (August, 2007).
21. N. Wakamiya and M. Murata. Toward Overlay Network Symbiosis. In *Proceedings of the 5th IEEE International Conference on Peer-to-Peer Computing*, pp. 154–155, (2005).
22. Y. Yemini, A. Dailianas, and D. Florissi. Marketnet: A market-based architecture for survivable large-scale information systems. In *Proceedings of the 4th ISSAT International Conference on Reliability and Quality in Design*, pp. 1–6, (1998).
23. M. Wellman, A Market-Oriented Programming Environment and its Application to Distributed Multicommodity Flow Problems, *Journal of Artificial Intelligence Research.* **1**, 1–23, (1993).
24. A. Othman, P. Dew, K. Djemamem, and I. Gourlay. Adaptive grid resource brokering. In *Proceedings of the 5th IEEE International Conference on Cluster Computing*, pp. 172–179, (2003).
25. K. Shirose, S. Matsuoka, H. Nakada, and H. Ogawa. Autonomous config-

uration of grid monitoring systems. In *Proceedings of the 4th International Symposium on Applications and the Internet Workshops*, pp. 651–657, (2004).

26. D. Garlan, S. Cheng, A. Huang, B. Schmerl, and P. Steenkiste, Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure, *IEEE Journal on Computer*. **37**(10), 46–54, (2004).
27. C. Adam and R. Stadler. Adaptable server clusters with QoS objectives. In *Proceedings of the 9th IFIP/IEEE International Symposium on Integrated Network Management*, pp. 149–162, (2005).