Biologically-inspired Adaptation of Autonomic Network Applications*

Junichi Suzuki  
jxs@cs.umb.edu  
Department of Computer Science  
University of Massachusetts, Boston  
Boston, MA, 02125-3393

Abstract

This paper describes and empirically evaluates a new biologically-inspired adaptation mechanism for network applications in the NetSphere architecture. The NetSphere architecture is inspired by the observation that the biological systems (e.g. bee colonies) have already developed mechanisms necessary to achieve future network requirements such as autonomy and adaptability. In the NetSphere architecture, a network application is implemented as a group of distributed and autonomous agents (analogous to a bee colony consisting of multiple bees). Each agent implements a functional service related to the application and follows simple behaviors similar to biological entities such as reproduction, replication, migration and environment sensing.

The proposed agent adaptation mechanism runs on the middleware platform for the NetSphere architecture, called the NetSphere platform. Designed after the mechanism behind how the immune system produces specific antibodies against an antigen invasion, the proposed adaptation mechanism allows agents to autonomously monitor their surrounding environmental conditions (e.g. traffic volume and resource availability) and adaptively perform their behaviors (e.g. reproduction and migration) suitable for the current environmental conditions. The empirical evaluation shows that the proposed mechanism achieves autonomous adaptability of network applications and the NetSphere platform is efficient and reusable to host autonomous adaptive network applications.

* Research supported by OGIS International, Inc. and Electric Power Development Co., Ltd.
1. INTRODUCTION

As computing devices and networks are becoming more powerful and ubiquitous, the networking landscape is evolving into new paradigms, such as autonomic networks [1], pervasive networks [2], grid networks [3] and space networks [4]. In these networking paradigms, future network applications will be orders of magnitude more complex and larger than current ones, and they are expected to be more autonomous, scalable and adaptive to dynamic network environments [5]. As inspiration for a new network application design paradigm, the author of the paper observes that various biological systems have already developed the mechanisms necessary to achieve the key requirements of future network applications such as autonomy, scalability and adaptability. For example, a bee colony scales to support a huge number of bees, and autonomously adapts to a wide variety of weather and food conditions. The author of the paper believes if network applications are modeled after certain biological concepts and mechanisms, they may be able to meet these requirements of future network applications.

The NetSphere architecture applies key concepts and mechanisms in biological systems to design network applications\(^1\). One of the key concepts in biological systems is emergence. In biological systems, beneficial system properties (e.g. adaptability) often emerge through the simple and autonomous interactions among diverse biological entities. The NetSphere architecture applies the concept of emergence by implementing network applications as a group of distributed, autonomous and diverse agents. This is analogous to a bee colony (a network application) consisting of multiple bees (agents). Each agent implements a functional service related to the application and follows simple behaviors similar to biological entities, such as reproduction, death, migration and environment sensing.

---

\(^1\) The NetSphere architecture is an extension to the Bio-Networking Architecture [6, 7, 8, 9, 10]. The Bio-Networking Architecture was first proposed in [7], later adopted by NTT Cooperation [10], and also adopted by the Object Management Group as a part of its standard specification [11].
Similar to entities in the biological world, agents in the NetSphere architecture are designed to provide a sufficient degree of diversity. Different agents may implement different services. For instance, an agent may implement an airline reservation service, while another agent may implement a hotel reservation service. An agent may implement a web service and contain web pages. Different agents may implement different behavior policies. For instance, an agent may have a migration policy of moving towards a user, while another agent may have a migration policy of moving towards a network node where resource availability is higher.

Similar to an entity in the biological world, each agent in the NetSphere architecture may store and expend energy for living. Each agent may gain energy in exchange for performing a service, and they may pay energy to receive a service from other agents and to use network and computing resources. The abundance or scarcity of stored energy may affect various behaviors of an agent. For example, an abundance of stored energy is an indication of higher demand for the agent; thus the agent may be designed to favor reproduction in response to higher levels of stored energy. A scarcity of stored energy (an indication of lack of demand or ineffective behaviors) may eventually cause the agent’s death.

This paper addresses autonomous adaptability of agents (i.e. network applications) in the NetSphere architecture. Autonomous adaptability is an ability of network applications (i.e. agents) to allow intelligently adapt to dynamic changes in the network without any administrative interventions from and to human users.

This paper describes and empirically evaluates an autonomous adaptation mechanism for network applications (i.e. agents) in the NetSphere architecture. The proposed adaptation mechanism is designed and executed on the runtime platform for the NetSphere architecture,
called the NetSphere platform. The NetSphere platform runs on a virtual machine\(^2\), and agents in the NetSphere architecture run atop the NetSphere platform (Figure 1). The platform provides reusable software components for developing, deploying and executing autonomous adaptive agents. The components abstract low-level operating and networking details (e.g. network I/O and concurrency control for executing agents), and implement high-level runtime services that agents use to perform their services and behaviors. The components are designed based on several biological mechanisms (e.g. migration, reproduction, energy exchange and immune response) so that agents satisfy the requirement of autonomous adaptability.

The proposed adaptation mechanism is designed on a key component in the NetSphere platform, iNet [12] (Figure 1), a behavior selection framework for network applications (i.e. agents). Modeled after the mechanism behind how the immune system produces specific antibodies against an antigen invasion, iNet allows agents to autonomously monitor their surrounding environmental conditions (e.g. traffic volume) and adaptively perform their behaviors (e.g. migration, replication and reproduction) suitable for the current environmental conditions. Empirical measurement results show that the proposed mechanism achieves autonomous adaptability of network applications and the NetSphere platform is efficient and reusable to host autonomous adaptive network applications.

\(^2\) Java virtual machine (JVM) is currently used to host the NetSphere platform on a network node.
This paper is organized as follows. Section 2 overviews key design principles of the NetSphere architecture. Based on the design principles described in Section 2, Section 3 describes the design of agents in the NetSphere architecture. Section 4 presents the design of key components in the NetSphere platform and describes how the platform components can contribute to improve autonomous adaptability of agents. Section 5 shows the results of empirical measurements to evaluate efficiency of the NetSphere platform and adaptability of agents implemented on the NetSphere platform. Sections 6 and 7 conclude with comparison with existing related work and some discussion on future work.

2. DESIGN PRINCIPLES OF THE NETSPHERE ARCHITECTURE

In the NetSphere architecture, agents are designed based on the three principles described below in order to interact and collectively provide network applications that are autonomous, adaptive, scalable and simple.

(1) Decentralization: Agents in the NetSphere architecture are decentralized. There are no central entities to control and coordinate agents (i.e. no directory servers and no resource managers). Decentralization allows network applications to be scalable and simple by
avoiding a single point of performance bottleneck and failure [13, 14] and by avoiding any central coordination in developing and deploying agents [14].

(2) Autonomy: Agents in the NetSphere architecture are autonomous. Agents monitor their local network environments, and based on the monitored environmental conditions, they autonomously interact without any intervention from human users or from other controlling entities.

(3) Adaptability: Agents in the NetSphere architecture are adaptive to dynamically changing environmental conditions (e.g. user demands, user locations and resource availability) over the short-term and long-term. The short-term adaptation is achieved through designing agent behavior policies to consider local environmental conditions [15]. For instance, agents may implement a migration policy of moving towards a platform\(^3\) that accepts a large number of user requests for their services. This results in the adaptation of agent locations, and agents concentrate around the users who request their services.

The long-term adaptation is achieved through applying biological evolutionary process. It occurs as a result of the natural selection (using energy) from diverse behavioral policies of agents. Diverse behavioral policies of agents may be created manually by human agent developers or created through crossover and mutation during replication and reproduction of agents. Through natural selection using energy\(^4\), beneficial behavior policies are retained while detrimental behavior policies become dormant or extinct over many successive generations, and agents specialize and improve themselves according to long-term environmental changes.

3. AGENTS IN THE NETSPHERE ARCHITECTURE

\(^3\) The term “platform” means the NetSphere platform in this paper.

\(^4\) As described in Section 1, an agent may store and expend energy for living. Agents with beneficial behavior policies will acquire more energy and reproduce more often than agents with detrimental behavior policies. Agents with detrimental behavior policies will eventually become extinct due to lack of energy.
Each agent in the NetSphere architecture is implemented as a Java object and runs on a NetSphere platform (Figure 1). The NetSphere platform is implemented in Java\(^5\), and each platform runs on a Java virtual machine (JVM) (Figure 1).

Each agent consists of three parts: attributes, body and behaviors (Figure 2). 

*Attributes* carry descriptive information regarding the agent (e.g. agent ID and description of a service it provides). The *body* implements a service that the agent provides and contains materials relevant to the service (e.g. application data and user profiles). For instance, an agent may implement control software for a device in its body, while another agent may implement a hotel reservation service in its body. An agent that implements a web service may contain web pages in its body. *Behaviors* implement non-service related actions that are inherent to all agents. Examples of behavior include migration, reproduction and energy exchange.

![Diagram](image_url)

**Figure 2. Design of an Agent in the NetSphere Architecture.**

(1) **Agent Attributes.** The current design of the NetSphere platform defines four mandatory attributes that every agent maintains: (1) Agent GUID (globally unique ID), (2) reference (or pointer) to the agent, (3) description of the service that the agent provides, and (4) price (in

\(^5\) The current code base of the NetSphere platform contains approximately 37,300 lines of Java code.
energy units) of the service that the agent provides. A GUID is a 32-digits string data created from the information provided by the platform where the agent was originally created (i.e. the IP address of the platform, JVM identity hash code\(^6\) of the GUID generator on the platform, the time when the agent was created on the platform, and a random number\(^7\) generated by the JVM that the platform runs on). An agent’s GUID is unique and does not change throughout the lifetime of an agent. An agent’s reference is a pointer that other agents use to send messages to the agent. It encapsulates the IP address and port number of the platform where the agent currently resides on. When an agent migrates, it obtains a new reference at the platform it migrates to. An agent’s reference is represented as a stringfied CORBA object reference [16]. The description of a service is the name of the service that an agent provides, and the price of a service represents the amount of energy required to receive the service that the agent provides. In addition to four mandatory attributes, the current design of the NetSphere platform allows agents to specify optional attributes.

Attributes are implemented as name-value pairs defined with the OMG constraint language [17]. Table 1 shows an example of mandatory attributes of an agent that provides a web service at the price of 100 energy units.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUID</td>
<td>'sti3sdr98rd56fn...'</td>
</tr>
<tr>
<td>ref</td>
<td>'IOR:daforimklcmd...'</td>
</tr>
<tr>
<td>serviceType</td>
<td>'HTTP/1.1'</td>
</tr>
<tr>
<td>serviceCost</td>
<td>100.0</td>
</tr>
</tbody>
</table>

Table 1. Example Agent Attributes.

\(^6\) This hash code is obtained by calling `System.identityHashCode()`.

\(^7\) Random numbers are generated with `java.util.Random` (default option because of its efficiency) or `java.security.SecureRandom`. 
(2) **Agent Body.** The body implements the service that an agent provides and contains materials relevant to the service (e.g. application data and user profiles). Implementation of an agent body is left to the developer of the agent.

(3) **Agent Behaviors.** Agents are autonomous, and follow simple biological behaviors. Some example behaviors are explained below.

- **Migration:** Agents may migrate from one platform to another.
- **Communication.** Agents may communicate with other agents for the purposes of, for instance, requesting a service or exchanging energy.
- **Energy exchange and storage:** Agents may receive and store energy in exchange for providing services to other agents. Agents also expend energy. For instance, agents may pay energy units for services that they receive from other agents. In addition, when an agent uses resources on a platform (e.g. CPU and memory), it may pay energy units to the platform.
- **Lifecycle regulation:** Agents may regulate their lifecycles. Agents may make a copy of themselves (replication), possibly with mutation of the replica’s behavioral policy. Two parent agents may create a child agent (reproduction) possibly with crossover and mutation of the child’s behavioral policy. Agents also may die (death) as a result of lack of energy. If energy expenditure of an agent is not balanced with the energy units it receives from providing services to other agents, it will not be able to pay for the resources it needs, i.e., it dies from lack of energy. Agents with wasteful behavioral policies (e.g. replicating or migrating too often) will have a higher chance of dying from lack of energy.
- **Relationship maintenance:** Agents may establish and maintain relationships with other agents. A relationship contains information regarding the partner agent, for instance, the attributes of the partner agent. Relationships are autonomously maintained by the
participant agents. Such relationships may have a variety of uses, including creating applications from a group of agents or performing discovery to search for agents.

- **Discovery**: Agents may seek for other agents of certain attributes by forwarding queries to agents that they have relationships to.

- **Pheromone emission**: Agents may emit and leave a pheromone (or a trace) behind on a platform when they migrate to another platform. This is to indicate their presence to other agents. A pheromone contains the emitter’s GUID and a reference to the platform that the emitter migrated to. Pheromones are emitted with certain strength and may decay over time. Pheromones may have a variety of uses, including improving the performance of discovery.

- **Environment sensing**: Agents may sense their local environment. For instance, an agent may sense the local environment to learn which agents are in the environment and what services they provide. An agent may also sense pheromones (e.g. which agents left pheromones on local and neighboring platforms) and resources (e.g. CPU processing power and memory space available on local and neighboring platforms).

The NetSphere platform implements behaviors explained above. Each behavior is implemented by one or more software components provided by the NetSphere platform. When a behavior is invoked, a corresponding software component (or components) is called.

### 4. Behavior Selection Engine for Agents

Each agent has its own behavior selection engine (Figure 2). A behavior selection engine allows an agent to monitor its surrounding environmental condition (e.g. resource availability on the local platform), identify behaviors suitable for the monitored environmental condition, and decide which behavior to invoke so that the agent adapt itself to the current environmental condition. Agent developers construct behavior selection engines
for their agents with the iNet framework, which is a software component provided by the NetSphere platform (Figure 1). The design of behavior selection engine is based on the mechanism behind how the immune system produces specific antibodies against an antigen invasion.

4.1. NATURAL IMMUNE SYSTEM

The immune system discriminates foreign molecules (i.e. non-self) from the body’s own cells and proteins (i.e. self). Once non-self is recognized, the immune system enlists the participation of a variety of cells and molecules to mount an appropriate response in order to eliminate or neutralize it. The immune response involves lymphocytes and antibodies. Lymphocytes are one of many types of white blood cells, and two major population of lymphocytes are B lymphocytes (or B cells) and T lymphocytes (or T cells). B cells have receptors on their surface, which can recognize antigens invading a human body, e.g. viruses, and then produce antibodies specific to the recognized antigen. The key portion of antigen that is recognized by antibodies is called epitope, which is the antigen determinant (Figure 3). Paratope is the portion of antibody that corresponds to a specific type of antigens. Once an antibody combines an antigen via their epitope and paratope, the antibody starts to eliminate the antigen. Each type of antibody has its own antigenic determinant, called idiotope. This means an antibody is recognized as an antigen by other antibodies.
Based on this fact, Jerne proposed the concept of the immune network, or idiotypic network [18], which states antibodies and lymphocytes are not isolated, but they are communicating with each other (Figure 3). The idiotope of an antibody is recognized by another antibody as an antigen. This network is formed based on idiotope recognition with the stimulation and suppression chains among antibodies (Figure 3). Thus, the immune response eliminating foreign antigens is offered by the entire immune system (or, at least, more than one antibody) in a collective manner, although the dominant role may be played by a single antibody whose paratope fits best with the epitope of a specific invading antigen. The immune network also helps to keep the quantitative balance of antibodies. Through stimulation and suppression interactions, the populations of specific antibodies rapidly increase following the recognition of any foreign antigen and, after eliminating the antigen, decrease again. Performed based on this self-regulation mechanism, the immune response has an emergent property through many interactions among antibodies. The structure of immune network is not fixed, but varies continuously.

4.2. Artificial Immune System in the NetSphere Architecture

In the NetSphere architecture, a behavior selection engine in an agent is designed
based on the concept of the immune network. It contains a network of antibodies (i.e. each agent contains its own immune network) (Figure 4). An antibody represents an agent behavior, and an antigen represents an environment condition. Each antibody (i.e. behavior) has its own concentration value, which represents the antibody’s population. Antibodies (i.e. behaviors) are linked with each other using stimulation and suppression relationships. When an antigen is stimulated by another one, the stimulation contributes to increase the concentration value of the stimulated antibody. On the contrary, when an antigen is suppressed by another one, the suppression contributes to decrease the concentration value of the suppressed antibody. Each stimulation and suppression relationship has its own strength value, or affinity value. The affinity value indicates the degree of stimulation or suppression.

![Behavior Selection Engine and Immune Network](image)

Figure 4. Behavior Selection Engine and Immune Network.

Figure 5 shows the structure of an antibody in the NetSphere architecture. The behavior compartment contains the ID of an agent behavior that the antibody represents. The paratope compartment contains a precondition under which the antibody (i.e. agent behavior) is selected. The idiotope compartment contains the relationships to other antibodies that the antibody stimulates.
In the NetSphere architecture, each agent periodically monitors its surrounding environment conditions with the software components provided by the NetSphere platform, produces antigens that represent the monitored environmental conditions, and supply the antigens into the agent’s immune network (i.e. behavior selection engine). A behavior selection process (i.e. immune response) begins when a supplied antigen matches with an antibody’s paratope. In a behavior selection process, a behavior selection engine (i.e. immune network) calculates concentration values of all the antibodies (i.e. agent behaviors), prioritizes them based on their concentrations, and selects one antibody that is most suitable to a given antigen(s) (i.e. environmental conditions). An agent performs the selected behavior using the software components provided by the NetSphere platform.

Figure 5. Antibody Structure

Figure 6 shows a generalized immune network in the NetSphere architecture. The antibody $i$ stimulates $M$ antibodies and suppresses $N$ antibodies. $m_{ji}$ and $m_{ik}$ denote affinity relationships to other antibodies (behaviors).
values between antibody \( j \) and \( i \), and between antibody \( i \) and \( k \), respectively. \( m_i \) 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.

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

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

In the equation (1), the first and second terms in a big bracket of the right hand side denote the stimulation and suppression from other antibodies. The affinity values between antibodies (i.e. \( m_{ji} \) and \( m_{ik} \)) are positive between 0 and 1. \( m_i \) is 1 when antibody \( i \) is stimulated directly by an antigen, otherwise 0. \( d \) denotes the dissipation factor representing the natural death of an antibody. This value is 0.1. The initial concentration value for every antibody, i.e. \( a_i(0) \), is 0.01. The equation (2) 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 [12, 15]. If no antibody exceeds a predefined threshold (0.7) 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 current concentrations (i.e. roulette-wheel selection).

The NetSphere architecture currently supports the paratopes and agent behavior IDs depicted in Table 2, in order to create various antibodies. A pair of paratope major ID and minor ID represents an environmental condition. An agent behavior major ID indicates an agent behavior, and an agent behavior minor ID represents a behavior policy. For instance, MIGRATION (behavior major ID) means agent’s migration behavior. HIGHER_RA
(behavior minor ID) indicates a migration policy of moving towards a platform where resource availability is higher, and HIGHER_TRAFFIC (behavior minor ID) indicates a migration policy of moving towards a platform that accepts higher network traffic. REPRODUCTION (behavior major ID) represents agent’s reproduction behavior. RANDOM (behavior minor ID) indicates a reproduction policy of creating a child agent with randomly chosen partner agent, and HIGHER_ENERGY (behavior minor ID) indicates a reproduction policy of creating a child agent with a partner agent whose energy level is higher. COMM-SERV (behavior major ID) represents agent’s communication behavior for providing a service to other agents.

<table>
<thead>
<tr>
<th>Paratope</th>
<th>Agent behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCAL RESOURCE AVAILABILITY (RA)</td>
<td>HIGH, LOW</td>
</tr>
<tr>
<td>REMOTE RESOURCE AVAILABILITY (RA)</td>
<td>HIGH, LOW</td>
</tr>
<tr>
<td>LOCAL TRAFFIC</td>
<td>HEAVY, LIGHT</td>
</tr>
<tr>
<td>REMOTE TRAFFIC</td>
<td>HEAVY, LIGHT</td>
</tr>
<tr>
<td>NUM LOCAL AGENTS</td>
<td>LARGE, SMALL</td>
</tr>
<tr>
<td>ENERGY LEVEL</td>
<td>HIGH, LOW</td>
</tr>
<tr>
<td>USER LOCATION</td>
<td>FAR, CLOSE</td>
</tr>
</tbody>
</table>

**Table 2. Paratopes and Agent Behaviors Supported in Antibodies.**

Figure 7 shows an example immune network. It contains four antibodies representing two behaviors: communication and migration (with two different migration policies). Antibody 1 represents that migration behavior is stimulated when resource availability is low on the local platform. Antibody 1 suppresses antibody 3 when it is stimulated (i.e. when resource availability is low on the local platform). Now, suppose that (1) local resource availability is low on the local platform, (2) network traffic is low on the local platform, and (3) user location is close. In this case, three antigens stimulate antibodies 1, 2 and 4
simultaneously. The populations of these antibodies increase, and it is likely that antibody 2’s concentration value becomes highest because antibody 2 suppresses antibody 4, which suppresses antibody 1. As a result, antibody 2 (i.e. migration behavior) would be selected by behavior selection engine.

![Example Immune Network](image)

**Figure 7. Example Immune Network**

5. The NetSphere Platform

The NetSphere platform provides an execution environment for agents (Figure 1). It consists of two types of software components. The first type of components, *supporting components*, abstracts low-level operating and networking details (e.g. network I/O and concurrency control for executing agents). The second type of components, *runtime components*, provides runtime services that agents use to perform their services and behaviors.

5.1 Supporting Components in the NetSphere Platform

Figure 8 shows some of the key supporting components in the NetSphere platform. `AgentImpl` in the package `edu.umb.cs.netsphere.agent` is the base class for agents. Developers of agents define their own agents by extending this class as subclasses of
this class. It provides a set of operations and variables that are common among all the agents. The operations and variables are used to implement attributes, body and behaviors of agents.

Each attribute of an agent is implemented as a typed pair of a name and a value, and attributes of an agent are implemented as a list of the typed pairs in the class `TypedNameValueList`. `AgentImpl` has a variable `attributes`, which is typed in `TypedNameValueList`, to maintain attributes of an agent. The operations of `TypedNameValueList` allow agents to define, modify and obtain their attributes. Implementation of an agent body is left up to the developer of the body. The NetSphere platform only assumes that it is implemented as one or more arbitrary operations in a subclass of `AgentImpl`. The operations that implement a body are called by the `run()` operation derived from the interface `Runnable` upon an arrival of a request for the corresponding

![Figure 8. Supporting Components in the NetSphere Platform.](image-url)
Behaviors of an agent are implemented by the runtime components of the NetSphere platform. **AgentImpl includes a variable availableRuntimeServices**, which is a list of references to the runtime components available on the NetSphere platform. In invoking a behavior, an agent examines **availableRuntimeServices** and obtains references to the runtime components that implement the behavior.

The package `edu.umb.cs.netsphere.inet` shows key components in the iNet framework. The framework provides a set of building blocks to construct and operate immune networks described in Section 4. **Antibody** and **Antigen** defines common variables and operations for antibodies and antigens, respectively. For instance, **Antibody** provides variables to maintain paratope major and minor IDs. **Relationship** represents a stimulation and suppression relationship between antibodies, and provides operations to configure affinity values. **BehaviorSelectionEngine** keeps references to antibodies. It also provides operations to calculate concentration values of antibodies and select one antibody that is most suitable to given environmental conditions. **BehaviorSelectionEngine** is directly accessible from **AgentImpl** in the package `edu.umb.cs.netsphere.agent`.

### 5.1 Runtime Components in the NetSphere Platform

The **runtime components** in the NetSphere platform provide runtime services that agents use to perform their services and behaviors. In order to maximize the degree of decentralization and autonomy of agents, agents only use the runtime components on the platform they reside. Agents do not invoke any runtime components running on a remote platform.
This section describes five key runtime components in the NetSphere platform. Agents use these runtime components along with behavior selection engine in order to adapt themselves to environmental conditions.

(1) **NetSphere Class Loader:** The *NetSphere Class Loader* dynamically loads an agent class definition into JVM when an agent is newly created on a platform or when a new agent migrates from another platform. The NetShere Class Loader is a customized class loader that extends JVM’s default class loader.

(2) **Migration Service:** The NetSphere platform provides the Migration Service, which implements the functionalities necessary to support the migration behavior of agents. The current implementation only supports weak migration [19], where an agent migrates only with its data state\(^8\). When an agent migrates, the Migration Service on the platform where the agent resides transmits the class name, class definition and runtime data state of the agent to the Migration Service on a destination platform. The class definition and data state are serialized at an origin platform and de-serialized on a destination by using the Java serialization mechanism. A destination-side Migration Service loads the received class definition into JVM using the NetSphere Class Loader, and then instantiates an agent with the received data state.

(3) **Communication Service.** The NetSphere platform provides the Communication Service, which implements the functionalities necessary to support the communication behavior of agents. The Communication Service handles marshalling messages, establishing and maintaining network connections, transmitting messages, un-marshalling messages, and managing threads to accept incoming messages. The current implementation uses the CORBA IIOP [16] as a message transport protocol on TCP.

(4) **Energy Management Service.** The NetSphere platform provides the Energy Management Service, which implements the functionalities necessary to support the energy exchange and

\(^8\) It does not support strong migration, where an agent migrates with both of its data and execution state [19].
storage behavior of agents. The Energy Management Service maintains a table, called the energy table, which contains pairs of agent’s GUID and energy level of each agent on the same platform. Using the energy table, the Energy Management Service allows an agent to pay energy units to other agents for the service it receives and to the platform for the resources (e.g. CPU and memory) it utilizes. Upon receiving a service, the Energy Management Service decreases the energy level of an agent that received a service by the price of the service, and contacts the Energy Management Service on the remote platform to increase the energy level of an agent that provided the service. In paying for platform resources that an agent utilizes, the Energy Management Service periodically decreases the energy level of an agent by the unit price of resources it utilizes.

(5) Lifecycle Regulation Service. As described in Section 3, agents may replicate, reproduce or die as a part of the lifecycle regulation behavior. The platform provides the Lifecycle Regulation Service, which implements the functionalities necessary to support the lifecycle regulation behavior of agents (i.e. to initialize, replicate, reproduce and destroy agents).

In order to initialize agents, the Lifecycle Regulation Service provides the initialization operation. This initialization operation is called when an agent is newly instantiated (either due to a creation of a new agent, replication or reproduction of an existing agent, or on the arrival of an agent from another platform). The initialization operation assigns a GUID to the agent, configures the agent’s behavior selection engine, registers the agent to the Communication Service, registers the agent to the energy table in the Energy Management Service, and starts running the initialized agent.

In order to replicate agents, the Lifecycle Regulation Service provides the replication operation. This replication operation makes a copy (child) of a parent agent using the Java

---

9 This step of assigning a GUID is not necessary for a migrated agent. An agent migrated from another platform already has GUID.
serialization mechanism\textsuperscript{10}, possibly executing a mutation on the child agent’s behavioral policy, and calls the initialization operation of the Lifecycle Regulation Service. When an agent reproduces a child agent with another agent, the agent calls the reproduction operation provided by the Lifecycle Regulation Service. The reproduction operation makes a copy (child) of the agent that called the operation, executes a crossover to inherit the behavioral policies of parent agents\textsuperscript{11}, possibly executing a mutation on the child agent’s behavioral policy, and calls the initialization operation of the Lifecycle Regulation Service.

In order to destroy an agent, the Lifecycle Regulation Service provides the destruction operation. The destruction operation frees the resources (e.g. memory and threads) that a dying agent utilizes, removes an entry for the dying agent from the energy table in the Energy Management Service, and un-registers the dying agent from the Communication Service. In the current implementation of the NetSphere platform, the destruction operation is called only by the Energy Management Service when the energy level of an agent becomes zero. No agents are allowed to call this operation to destruct other agents.

\textbf{(6) Environment Sensing Service.} As described in Section 3, agents may detect various environmental conditions through the environment sensing behavior. In order to support the environment sensing behavior, the platform provides the Environment Sensing Service. It allows each agent to sense (a) the agents running on the same and remote platforms, (b) resource availability on the same and remote platforms, and (c) network traffic load and traffic patterns on the same and remote platforms.

The Environment Sensing Service allows an agent to sense other agents on the same and remote platforms. The service maintains a list of references to the agents that are on the local platform, and returns the reference list when invoked by an agent. In order to sense

\textsuperscript{10} When replicating an agent, the parent agent’s antibodies (i.e. behavior selection engine) is copied, or inherited, to a child agent.

\textsuperscript{11} When reproducing a child agent from two parents agents, the child agent inherits two different set of antibodies (i.e. behavior selection engine) from parent agents.
agents running on remote platforms, the Environment Sensing Service contacts the Environment Sensing Service on a remote platform and obtains a list of the agents on the remote platform.

The Environment Sensing Service also allows an agent to sense resource availability on the same and remote platforms. The service monitors resources such as CPU and memory available on the same platform, and maintains the type, amount and unit price of each resource (in energy units). CPU availability is calculated by measuring the current CPU utilization\(^\text{12}\). Memory availability is obtained by executing a garbage collection and measuring the amount of free memory in JVM. In order to sense the resource availability on remote platforms, an agent asks the Environment Sensing Service to contact the other Environment Sensing Services running on remote platforms.

With the Environment Sensing Service, an agent can sense the network traffic load and traffic patterns on the same and remote platforms. It monitors network traffic load on the same platform by counting the number and size of incoming messages. It also monitors network traffic patterns on the same platform by recording the sources of incoming messages. The Environment Sensing Service also finds the sender agent of an incoming message by obtaining a reference to the sender agent through parsing the incoming message. In order to sense the network traffic load and traffic patterns on remote platforms, an agent asks the Environment Sensing Service to contact the Environment Sensing Services on remote platforms.

6. EMPIRICAL EVALUATION

This section empirically evaluates the simplicity of developing autonomous adaptive

\(^{12}\) Since measurement of CPU utilization is not available through the standard Java APIs, CPU utilization is measured with a non-Java library implemented with C and Java Native Interface.
agents that operate the behavior selection engine proposed in this paper. It also empirically examines how the behavior selection engine is efficient and how it contributes to the adaptability of agents.

6.1 Configurations for Empirical Evaluation

In the empirical evaluation presented in this section, measurements were obtained assuming varying numbers of agents (from 1 through approx 100 agents) and NetSphere platforms (from 1 through 16 platforms). A maximum of 8 Windows XP PCs are used in the empirical evaluation, each running the Java 2 standard edition JVMs (version 1.4.2_04 from Sun Microsystems). These 8 PCs were divided into 4 groups of 2 PCs in each group, depending on their CPU speed and memory size, as shown in Table 3. This section shows four separate measurements using different configurations. The first three measurements use a PC of the group D configuration shown in Table 3. The last measurement uses 8 PCs (2 PCs from each of the 4 configurations shown in Table 3). These PCs were connected through 100Mbps Ethernet.

<table>
<thead>
<tr>
<th>Group</th>
<th>CPU</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Intel Celeron 2.0 GHz</td>
<td>512 MB</td>
</tr>
<tr>
<td>B</td>
<td>Intel Celeron 2.4 GHz</td>
<td>640 MB</td>
</tr>
<tr>
<td>B</td>
<td>Intel Pentium 4 2.8 GHz</td>
<td>1000 MB</td>
</tr>
<tr>
<td>D</td>
<td>Intel Pentium 4 3.0 MHz</td>
<td>1000 MB</td>
</tr>
</tbody>
</table>

Table 3. Configurations of PCs used in Empirical Evaluation.

6.2 Application Development Using Autonomous Adaptive Agents

In order to examine how the NetSphere platform reduces the complexity of developing network applications, web services are implemented using agents. The attributes of a web service agent are empty. The body of a web service agent contains a set of files, accepts HTTP request messages from users, and returns the requested files to the users. The behaviors part of a web service agent contains a behavior selection engine that maintains 50
antibodies. The 50 antibodies are built by randomly combining 7 behavior policies and 14 environmental conditions described in Table 2.

Table 4 shows the empirical evaluation of the web service application. It shows that the iNet framework, one of the supporting components in the NetSphere platform, simplifies building and configuring agent’s behavior selection engine. Other supporting components also help to reduce the lines of code and development time. It is fairly simple and easy to implement network applications on the NetSphere platform.

<table>
<thead>
<tr>
<th># of Classes</th>
<th>Agent attributes</th>
<th>Agent body</th>
<th>Agent behaviors</th>
<th>Total lines of code</th>
<th>Development Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>198</td>
<td>2 hours</td>
</tr>
</tbody>
</table>

Table 4. Application Implemented using the NetSphere Platform.

6.3 OVERHEAD OF DEPLOYING AUTONOMOUS ADAPTIVE AGENTS

In order to evaluate the efficiency of deploying an agent on a platform, Table 5 shows the time required for the NetSphere platform to execute key steps of agent deployment. The measurement examined the following key deployment steps; instantiating an agent, configuring the initial status of behavior selection engine, and initializing an instantiated agent with the Lifecycle Regulation Service. In the step to instantiate an agent, the measurement examined two cases; the case where a human developer manually instantiates a new agent, and the case where a parent agent replicates (makes a copy of) itself. In the former case, an agent configures its behavior selection engine with 50 antibodies by randomly combining 7 behavior policies and 14 environmental conditions described in Table 2. In the latter case, a behavior selection engine that contains 50 antibodies is inherited from a parent agent to child agent.
Table 5 shows that the overhead of deploying an agent is small and that the Lifecycle Regulation Service is efficient. The overhead difference between the two cases to instantiate an agent is due to the time to make a copy of a parent agent in the replication process.

<table>
<thead>
<tr>
<th>Deployment Steps</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instantiate an agent</td>
<td></td>
</tr>
<tr>
<td>Manually create a new agent</td>
<td>25.38 msec</td>
</tr>
<tr>
<td>Replicate a parent agent with the Lifecycle Regulation Service</td>
<td>114.12 msec</td>
</tr>
<tr>
<td>Configure an agent’s behavior selection engine with 50 antibodies</td>
<td>40.62 msec</td>
</tr>
<tr>
<td>Initialize an agent with the Lifecycle Regulation Service</td>
<td>186.98 msec</td>
</tr>
<tr>
<td>Total</td>
<td></td>
</tr>
<tr>
<td>Deploy a new agent</td>
<td>252.98 msec</td>
</tr>
<tr>
<td>Deploy a replicated agent</td>
<td>341.72 msec</td>
</tr>
</tbody>
</table>

Table 5. Overhead of Deploying Autonomous Adaptive Agents.

6.4 OVERHEAD OF BEHAVIOR SELECTION ENGINE

In order to evaluate the efficiency of behavior selection engine, Table 6 shows the overhead for a behavior selection engine to calculate concentration value of every antibody. The measured overhead is the time required for a behavior selection engine to repeat executing Equations (1) and (2) 200 times on every antibody. Although the measured overhead does not grow linearly, 9.95sec is acceptable in the case of 100 antibodies. Please note that a behavior selection engine can produce up to 96 antibodies using 7 behavior policies and 14 environmental conditions that the NetSphere architecture currently supports (Table 2).

<table>
<thead>
<tr>
<th># of antibodies</th>
<th>4</th>
<th>10</th>
<th>30</th>
<th>50</th>
<th>70</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calculation overhead (sec)</td>
<td>0.14</td>
<td>0.29</td>
<td>1.25</td>
<td>2.88</td>
<td>5.20</td>
<td>9.95</td>
</tr>
</tbody>
</table>

Table 6. Overhead of Behavior Selection Engine.

6.5 AUTONOMOUS ADAPTABILITY OF AGENTS

This measurement evaluates autonomous adaptability of agents. In this measurement, 16 NetSphere platforms are deployed on 8 PCs (i.e. 2 platforms per PC), and they are
connected with each other based on a grid topology (Figure 9). At the beginning of a measurement, a single web service agent (see Section 6.2) is randomly deployed on a platform. Each web service agent contains a behavior selection engine that is configured with 5 behavior policies (i.e. antibodies) and 7 environment conditions (i.e. paratopes) shown in Table 2. A workload generator generates HTTP request messages and randomly sends them to web service agents. It keeps track of the locations of web service agents. When a web service agent migrates, the agent notifies its new location to the workload generator\(^\text{13}\). The workload generator pays energy units to a web service when it receives a HTTP response message from the agent.

\[
\begin{array}{|c|c|}
\hline
\text{Paratope} & \text{Agent behavior} \\
\hline
\text{LOCAL RESOURCE AVAILABILITY (RA)} & \text{LOCAL TRAFFIC} & \text{NUM LOCAL AGENTS} & \text{ENERGY LEVEL} \\
\text{HIGH, LOW} & \text{HEAVY, LIGHT} & \text{LARGE, SMALL} & \text{HIGH, LOW} \\
\hline
\text{TOWARDS_USER} & \text{HIGHER_RA} & \text{N/A} & \text{N/A} \\
\text{MIGRATION} & \text{HIGHER_TRAFFIC} & & \\
\hline
\end{array}
\]

\textbf{Table 2. Paratopes and Agent Behaviors Supported in Antibodies.}

Figures 10 (1) shows the workload for web service agents, i.e. how many HTTP

\(^{13}\) In principle, agents are decentralized in the NetSphere architecture (i.e. no centralized directory). However, for simplicity, the workload generator in this measurement plays a role of a directory that maintains the locations of agents.
request messages are generated and sent to web service agents. The workload gradually grows in this measurement. Figure 10 (2) shows how the number of web service agents changes against the workload change. As web service agents replicate in response to enough energy gain from the workload generator, they autonomously change their population, up to 84 agents, so that they can process more HTTP request messages. Figure 10 (3) shows how many HTTP response messages web service agents send back to the workload generator. Since they migrate to platforms where resource availability is higher, web service agents change their locations so that they can process HTTP request messages more efficiently.

![Figure 10 (1). Generated workload](image1)

![Figure 10 (2). The number of agents.](image2)
7. RELATED WORK

The immunological concepts and mechanisms have been applied to various areas such as virus detection, intrusion detection, network protection, image processing and robot navigation [20, 21, 22]. However, there are few research efforts that apply the immune system’s adaptive property to network application design. The author of the paper believes that this work is the first attempt to apply the immune network’s mechanism to an autonomous adaptation strategy in network application components.

The NetSphere platform has some commonality with existing mobile agent platforms such as Aglets [23], AgentSpace [24] and Hive [25]. For instance, both the NetSphere platform and existing mobile agent platforms support mobility of agents and facilitate abstraction of low-level operating details and communication between agents. Unlike existing mobile agent platforms, the NetSphere platform provides a new biologically-inspired adaptation strategy for agents. In addition, unlike existing mobile agent platforms, the NetSphere platform also supports runtime services designed after biological concepts, such as energy management and lifecycle regulation.

8. CONCLUDING REMARKS
This paper describes and empirically evaluates a new biologically-inspired adaptation mechanism for network application components. With the proposed mechanism operated on the NetSphere platform, network applications satisfy the key requirements of future network applications such as autonomy and adaptability. The empirical evaluation shows that the proposed mechanism achieves autonomous adaptability of network applications and the NetSphere platform is efficient and reusable to host autonomous adaptive network applications.

An extended set of empirical measurements are being planned to investigate additional adaptability property of network applications (i.e. agents) in the NetSphere architecture. Further deployment of the NetSphere platform and agents on larger-scale environments (e.g. PlanetLab [26]) would identify the impact of the network size and realistic constraints on the autonomous adaptability of agents.

**Reference**


