A Middleware Platform for a Biologically Inspired Network Architecture Supporting Autonomous and Adaptive Applications

Junichi Suzuki, *Member, IEEE,* and Tatsuya Suda, *Fellow, IEEE*

***Abstract*—This paper describes and empirically evaluates the middleware platform of a new network architecture called the Bio-Networking Architecture. The Bio-Networking 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, scalability, adaptability, and simplicity. In the Bio-Networking Architecture, a network application is implemented as a group of distributed, autonomous and diverse objects called cyber-entities (CEs) (analogous to a bee colony consisting of multiple bees). Each CE implements a functional service related to the application and follows simple behaviors similar to biological entities (e.g., reproduction and migration). In the Bio-Networking Architecture, beneficial application characteristics (e.g., autonomy, scalability, adaptability, and simplicity) arise from the autonomous interaction of CEs. The middleware platform in the Bio-Networking Architecture, the bionet platform, provides reusable software components for developing, deploying, and executing CEs. The components abstract low-level operating and networking details, and implement high-level runtime services that CEs use to perform their services and behaviors. The components in the bionet platform are designed based on several biological concepts (e.g., energy exchange and pheromone emission). This paper describes key designs of the bionet platform and empirically demonstrates that the bionet platform is efficient, scalable, reusable, and significantly simplifies development of network applications.**

***Index Terms*—Autonomic computing middleware, autonomous and adaptive network applications, biologically inspired network architecture for distributed computing.**

I. Introduction

FUTURE network applications are expected to be autonomous, scalable, adaptive to dynamic network environments, and to be simple to develop and deploy. In order to realize future network applications with such desirable characteristics, the authors of this paper observe that various biological systems have already developed the mechanisms necessary to achieve the key requirements of future network applications such as autonomy, scalability, adaptability, and simplicity. The authors of the paper believe if network applications are modeled after certain biological concepts and mechanisms, they may be able to meet these requirements of future network applications.

The Bio-Networking Architecture [1]–[5] 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 Bio-Networking Architecture applies the concept of emergence by implementing network applications as a group of distributed, autonomous, and diverse objects called cyber-entities (CEs). This is analogous to a bee colony (a network application) consisting of multiple bees (CEs). Each CE implements a functional service related to the application and follows simple behaviors similar to biological entities, such as reproduction, death, migration, and environment sensing.

Similar to entities in the biological world, CEs in the Bio-Networking Architecture are designed to provide a sufficient degree of diversity. Different CEs may implement different services. For instance, a CE may implement an airline reservation service, while another CE may implement a hotel reservation service. A CE may implement a Web service and contain Web pages. Different CEs may implement different behavior policies. For instance, a CE may have a migration policy of moving toward a user, while another CE may have a migration policy of moving toward a node, where resource availability is higher.

Similar to an entity in the biological world, each CE in the Bio-Networking Architecture may store and expend *energy* for living. CEs may gain energy in exchange for performing a service, and they may pay energy to receive a service from other CEs and to use network and computing resources. The abundance or scarcity of stored energy may affect various behaviors of a CE. For example, an abundance of stored energy is an indication of higher demand for the CE; thus, the CE 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 CE's death.

Manuscript received December 1, 2003; revised May 15, 2004. This work was supported in part by the National Science Foundation (NSF) under Grant ANI-0083074 and Grant ANI-9903427, in part by the Defense Advanced Research Projects Agency (DARPA) under Grant MDA972-99-1-0007, in part by the Air Force Office of Scientific Research (AFOSR) under Grant MURI F49620-00-1-0330, and in part by grants from the California MICRO and CoRe Programs, Hitachi, Hitachi America, Novell, Nippon Telegraph and Telephone (NTT), NTT Docomo, Fujitsu, NS Solutions Corporation, and Denso IT Laboratory.

J. Suzuki is with the Department of Computer Science, University of Massachusetts, Boston, MA 02125-3393 USA (e-mail: jxs@cs.umb.edu).

T. Suda is with the School of Information and Computer Science, University of California, Irvine, CA 92697-3425 USA (e-mail: suda@ics.uci.edu).

Digital Object Identifier 10.1109/JSAC.2004.839388

[1]The Bio-Networking Architecture was first proposed in [2], later adopted by NTT [3], and also adopted by the Object Management Group (OMG) as a part of its standard specification for super distributed objects (SDO) [6].

In the Bio-Networking Architecture, application functionality emerges from the collaborative execution of services carried by CEs, and beneficial application characteristics (e.g., autonomy, scalability, adaptability, and simplicity) arise from the simple and diverse behaviors among CEs and from the autonomous interaction of individual CEs.

This paper describes and empirically evaluates the design of the middleware platform in the Bio-Networking Architecture, called the bionet platform [4]. The bionet platform runs on a virtual machine, and CEs run atop the bionet platform. It provides reusable software components for developing, deploying and executing CEs. The components abstract low-level operating and networking details (e.g., network I/O and concurrency control for executing CEs), and implement high-level runtime services that CEs use to perform their services and behaviors. The components are designed based on several biological mechanisms (e.g., migration, replication, reproduction, energy exchange, and pheromone emission) so that CEs satisfy future network requirements. Empirical measurements show that the bionet platform is efficient, scalable, reusable, and significantly simplifies development of network applications.

This paper is organized as follows. Section II overviews key design principles of the Bio-Networking Architecture. Based on the design principles described in Section II, Section III identifies functionalities that the bionet platform performs. Section III also describes the design of the CEs and bionet platform. Section IV shows the results of empirical measurements to evaluate the bionet platform and applications implemented on the bionet platforms. Section V concludes with some discussion on future work.

II. Design Principles of the Bio-Networking Architecture

In the Bio-Networking Architecture, CEs are designed based on the three principles described below in order to interact and collectively provide network applications that are autonomous, scalable, adaptive, and simple.

1) Decentralization: CEs are decentralized. There are no central entities to control and coordinate CEs (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 [7], [8] and by avoiding any central coordination in developing and deploying CEs [8].

2) Autonomy: CEs are autonomous. CEs 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: CEs 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 CE behavior policies to consider local environmental conditions [2]. For instance, CEs may implement a migration policy of moving toward a bionet platform that forward a large number of user requests for their services. This results in the adaptation of CE locations, and CEs concentrate around the users who request their services. The long-term adaptation occurs as a result of the natural selection (using energy) from diverse behavioral policies of CEs. Diverse behavioral policies of CEs may be created manually by human CE developers or created through crossover and mutation during replication and reproduction of CEs. Through natural selection using energy,[2] beneficial behavior policies are retained, while detrimental behavior policies become dormant or extinct over many successive generations, and the CEs specialize and improve themselves according to long-term environmental changes [5].

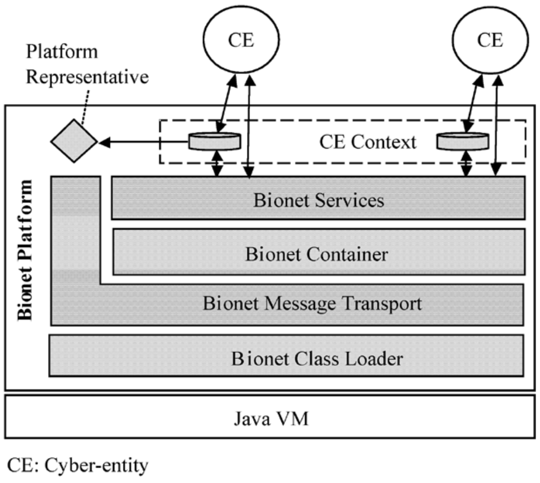

Fig. 1. Bionet platform architecture.

III. CEs and the Bionet Platform

The bionet platform provides an execution environment for CEs. 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 CEs). The second type of components, *runtime components*, provides runtime services that CEs use to perform their services and behaviors. The bionet platform is implemented in Java,[3] and each bionet platform runs on a Java virtual machine (JVM) (Fig. 1). Each CE is implemented as a Java object and runs on a bionet platform (Fig. 1).

A. Cyber-Entities (CEs)

A CE consists of three main parts: *attributes, body*, and *behaviors* (Fig. 2). *Attributes* carry descriptive information regarding the CE (e.g., CE ID and description of a service it provides). The *body* implements a service that the CE provides and contains materials relevant to the service (e.g., data, application code, or user profiles). For instance, the CE may implement control software for a device in its body, while another CE may implement a hotel reservation service in its body. A CE that implements a Web service may contain Web

[2]As described in Section I, a CE may store and expend *energy* for living. CEs with beneficial behavior policies will acquire more energy and reproduce more often than CEs with detrimental behavior policies. CEs with detrimental behavior policies will eventually become extinct due to lack of energy.

[3]The current code base of the bionet platform contains approximately 30 600 semicolons.

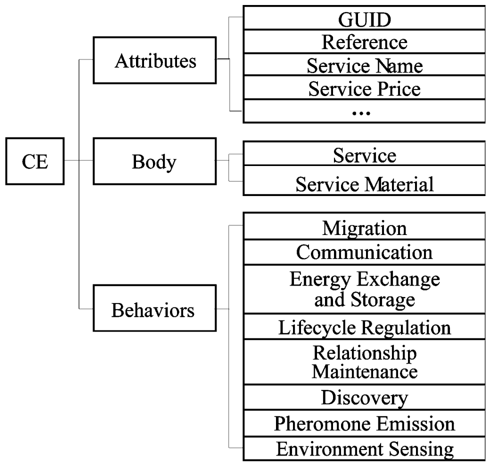

Fig. 2. Design of a CE.

pages in its body. CE *behaviors* implement nonservice related actions that are inherent to all CEs. Examples of behavior include migration, reproduction, and energy exchange.

1) CE Attributes: The current design of the bionet platform defines four mandatory attributes that every CE maintains: 1) globally unique ID (CE GUID); 2) reference (or pointer) to the CE; 3) description of the service that the CE provides; and 4) price (in energy units) of the service that the CE provides. A GUID is a 32-digits string data created from the information provided by the platform where the CE was originally created (i.e., the Internet protocol (IP) address of the platform, JVM identity hash code[4] of the GUID generator on the platform, the time when the CE was created on the platform, and a random number[5] generated by the JVM that the platform runs on). A CE's GUID is unique and does not change throughout the lifetime of a CE. A CE's reference is a pointer that other CEs use to send messages to the CE. It encapsulates the IP address and port number of the platform where the CE currently resides on. When a CE migrates, it obtains a new reference at the platform it migrates to. A CE's reference is represented as a stringfied CORBA object [9]. The description of a service is the name of the service that a CE provides, and the price of a service represents the amount of energy required to receive the service that the CE provides. In addition to four mandatory attributes, the current design of the bionet platform allows CEs to specify optional attributes.

Attributes are implemented as name-value pairs defined with the OMG constraint language [10]. Table I shows an example of mandatory attributes of a CE that provides a Web service at the price of 100 energy units.

2) CE Body: The body implements the service that a CE provides and contains materials relevant to the service (e.g., data, application code or user profiles). Implementation of a CE body is left to the developer of the CE.

3) CE Behaviors: CEs are autonomous and follow simple biological behaviors. Some example behaviors are explained next.

[4]This hash code is obtained by calling `System.identityHashCode()`.

[5]Random numbers are generated with `java.util.Random` (default option because of its efficiency) or `java.security.SecureRandom`.

TABLE I
EXAMPLE OF CE ATTRIBUTES

Attribute Name = Attribute Value
GUID = 'sti3sdr98rd56fn...'
ref = 'IOR:daforimklcmd...'
serviceType = 'HTTP/1.1'
serviceCost = 100.0

- *Migration:* CEs may migrate from one bionet platform to another.
- *Communication.* CEs may communicate with other CEs for the purposes of, for instance, requesting a service, forwarding a discovery query, or exchanging energy.
- *Energy exchange and storage*: CEs may receive and store energy in exchange for providing services to other CEs. CEs also expend energy. For instance, CEs may pay energy units for services that they receive from other CEs. In addition, when a CE uses resources on a bionet platform (e.g., CPU and memory), it may pay energy units to the platform.
- *Lifecycle regulation:* CEs may regulate their lifecycles. CEs may make a copy of themselves (replication), possibly with mutation of the replica's behavioral policy. Two parent CEs may create a child CE (reproduction) possibly with crossover and mutation of the child's behavioral policy. CEs also may die (death) as a result of lack of energy. If energy expenditure of a CE is not balanced with the energy units it receives from providing services to other CEs, it will not be able to pay for the resources it needs, i.e., it dies from lack of energy. CEs with wasteful behavioral policies (e.g., replicating or migrating too often) will have a higher chance of dying from lack of energy.
- *Relationship maintenance:* CEs may establish and maintain relationships with other CEs. A relationship contains information regarding the partner CE, for instance, the attributes of the partner CE. Relationships are autonomously maintained by the participant CEs. Such relationships may have a variety of uses, including creating applications from a group of CEs or performing discovery to search for CEs.
- *Discovery:* CEs may seek for other CEs of certain attributes by forwarding queries to CEs that they have relationships to.
- *Pheromone emission:* CEs may emit and leave a pheromone (or a trace) behind on a bionet platform when they migrate to another platform. This is to indicate their presence to other CEs. 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.* CEs may sense their local environment. For instance, a CE may sense the local environment to learn which CEs are in the environment and what services they provide. A CE may also sense pheromones (e.g., which CEs left pheromones on remote bionet

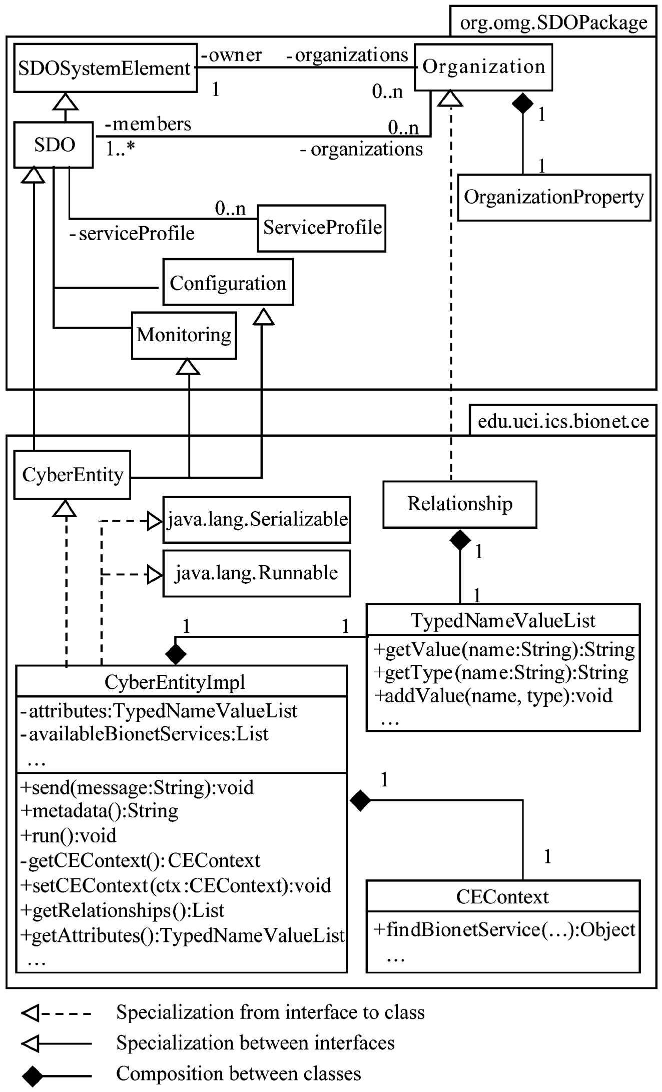
org.omg.SDOPackage
SDOSystemElement
-owner
-organizations
Organization
1
0..n
-members
0..n
SDO
1..*
- organizations
1
1
OrganizationProperty
0..n
ServiceProfile
- serviceProfile
Configuration
Monitoring
edu.uci.ics.bionet.ce
CyberEntity
java.lang.Serializable
Relationship
java.lang.Runnable
1
1
1
1
TypedNameValueList
+getValue(name:String):String
+getType(name:String):String
+addValue(name, type):void
...
CyberEntityImpl
-attributes:TypedNameValueList
-availableBionetServices:List
...
+send(message:String):void
+metadata():String
+run():void
-getCEContext(): CEContext
+setCEContext(ctx:CEContext):void
+getRelationships():List
+getAttributes():TypedNameValueList
...
1
1
CEContext
+findBionetService(...):Object
...
Specialization from interface to class
Specialization between interfaces
Composition between classes

Fig. 3. Class diagram around `CyberEntityImpl`.

platforms) and resources (e.g., CPU processing power and memory space available on remote bionet platforms).

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

B. Supporting Components in the Bionet Platform

As described earlier in this section, the *supporting components* in the bionet platform abstract low-level operating and networking details. The package `edu.uci.ics.bionet.ce` on Fig. 3 shows some of the key supporting components in the bionet platform. `CyberEntityImpl` is the base class for CEs. Developers of CEs define their own CEs by extending this class as subclasses of this class. It provides a set of operations and variables that are common among all the CEs. The operations and variables are used to implement attributes, body, and behaviors of CEs in the following manner.

Each attribute of a CE is implemented as a typed pair of a name and a value, and attributes of a CE are implemented as a list of the typed pairs in the class `TypedNameValueList`. `CyberEntityImpl` has a variable `attributes`, which is typed in `TypedNameValueList`, to maintain attributes of a CE. The operations of `TypedNameValueList` allow CEs to define, modify, and obtain their attributes. Implementation of a CE body is left up to the developer of the body. The bionet platform only assumes that it is implemented as one or more arbitrary operations in a subclass of `CyberEntityImpl`. 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 service. Behaviors of a CE are implemented by the runtime components of the bionet platform. `CyberEntityImpl` includes a variable `availableBionetServices`, which is a list of references to the runtime components available on the bionet platform. In invoking a behavior, a CE examines `availableBionet-Services` and obtains references to the runtime components that implement the behavior.

The key designs of the supporting components in the bionet platform form a foundation of a standard specification at the SDOs special interest group (SIG) of the OMG [6]. The SDO SIG standardizes a uniform object model for supporting heterogeneous hardware devices and software services in highly distributed environments. The package `org.omg.SDOPackage` on Fig. 3 shows some of the components defined in the OMG SDO specification. The interface `SDO` is a uniform representation of heterogeneous hardware devices and software services. Multiple `SDO`s may form relationships between themselves using the interface `Organization` in order to, for example, create a group of `SDO`s and forward discovery queries among group member `SDO`s. The class `ServiceProfile` is used to define properties of `SDO`'s function (e.g., identifier and name of `SDO`'s function). The interface `Configuration` is to configure (i.e., define and modify) the properties stored in `ServiceProfile`, and the interface `Monitoring` is to monitor and obtain the properties stored in `ServicePro-file`.

The components in the OMG SDO specification are implemented by the supporting components in the bionet platform (Fig. 3). For example, `CyberEntityImpl` (in the bionet platform) implements `SDO`. `TypedNameValueList` (in the bionet platform) implements `ServiceProfile`, `Config-uration`, and `Monitoring`. The bionet platform serves as a reference implementation of the OMG SDO specification.

C. Runtime Components in the Bionet Platform

1) Architecture of the Bionet Platform: The *runtime components* in the bionet platform provide runtime services that CEs use to perform their services and behaviors. In order to maximize the degree of decentralization and autonomy of CEs, CEs only use the runtime components on the platform they reside. CEs do not invoke any runtime components running on a remote bionet platform. In addition, there are no runtime components that control or coordinate other runtime components.

The current design of the bionet platform defines six runtime components as shown in Fig. 1. The *bionet class loader* dynamically loads a CE class definition into JVM when a CE is newly created on a bionet platform or when a new CE migrates from another bionet platform.[6] The *bionet message transport* performs functionalities required for communication between different CEs and between different bionet platforms, such as marshalling and transmitting of messages. The *bionet container* maintains a reference table to the CEs running on a bionet platform, and it uses the table to dispatch incoming messages to CEs. The *bionet services* implement CE behaviors. A *platform representative* contains the information (e.g., address) on a bionet platform. It is used by CEs and runtime components to reference a bionet platform. For example, the pheromone that a CE emits when migrating to another bionet platform contains a platform representative of the CE's destination platform. A *CE context* is an entry point for a CE to access underlying runtime components (e.g., bionet services). It examines whether a runtime component requested by a CE is available on a bionet platform, and if it is, it returns a reference to the requested component to the CE. A CE context is created and associated with a CE by a bionet service (the lifecycle management service to be explained below) when the CE is newly instantiated on a bionet platform (either due to a creation of a new CE, replication or reproduction of an existing CE, or on the arrival of a CE from another bionet platform).

2) Bionet Services: The bionet platform provides nine bionet services. Table II summarizes the nine bionet services. These nine bionet services along with the runtime components that the bionet platform provides (described earlier in Section III-C) implement eight CE behaviors described in Section III-A. The implementation of each CE behavior is described below[7] in detail.

a) Migration Behavior: The bionet platform provides the migration service (a bionet service), which implements the functionalities necessary to support the migration behavior of CEs. The current implementation only supports weak migration [13], where a CE migrates only with its data state.[8] When a CE migrates, the migration service on the bionet platform where the CE resides transmits the class name, class definition, and runtime data state of the CE to the migration service on a destination bionet platform. The class definition and data state are serialized at an origin bionet platform and deserialized on a destination by using the Java serialization mechanism. A destination-side migration service loads the received class definition into JVM using the bionet class loader (a runtime component), and then instantiates a CE with the received data state.

b) Communication Behavior: The bionet platform provides the bionet message transport (a runtime component), which implements the functionalities necessary to support the communication behavior of CEs.[9] The bionet message transport handles marshalling messages, establishing and maintaining network connections, transmitting messages, unmarshalling messages, and managing threads to accept incoming messages. The current implementation uses the CORBA IIOP [9] as a message transport protocol on transmission control protocol (TCP).

c) Energy Exchange and Storage Behavior: The bionet platform provides the energy management service (a bionet service), which implements the functionalities necessary to support the energy exchange and storage behavior of CEs. The energy management service maintains a table, called the energy table, which contains pairs of CE's GUID and energy level of each CE on the same bionet platform. Using the energy table, the energy management service allows a CE to pay energy units to other CEs for the service it receives and to the bionet platform for the resources (e.g., CPU and memory) it utilizes. Upon receiving a service, the energy management service decreases the energy level of a CE 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 a CE that provided the service. In paying for platform resources that a CE utilizes, the energy management service periodically decreases the energy level of a CE by the unit price of resources it utilizes.

TABLE II
LIST OF THE BIONET SERVICES

Name	Supported CE Behavior	Functionality
Migration	Migration	allows CEs to migrate from a platform to a platform.
Energy Management	Energy Exchange and Storage	allows CEs to exchange and store energy.
Lifecycle Management	Lifecycle Regulation	allows CEs to initialize, replicate, reproduce and die.
Relationship Management	Relationship Maintenance	allows CEs to establish, examine, update and eliminate relationships.
Social Networking	Discovery	allows CEs to discover other CEs with certain attributes.
Pheromone Emission	Pheromone Emission	allows CEs to emit pheromones on a bionet platform.
CE Sensing	Environment Sensing	allows CEs to sense other CEs on the same or remote platforms.
Pheromone Sensing	Environment Sensing	allows CEs to sense pheromones on the same or remote platforms.
Resource Sensing	Environment Sensing	allows CEs to sense resource availability on the same or remote platforms.

[6]The bionet class loader is a customized class loader that extends JVM's (default) system class loader.

[7]The bionet platform has some commonality with existing mobile agent platforms such as Aglets [11], AgentSpace [12], and Hive [8]. For instance, both the bionet platform and existing mobile agent platforms support mobility of agents (i.e., CEs) and facilitate abstraction of low-level operating details and communication between agents. Unlike existing mobile agent platforms, the bionet platform applies biological concepts and supports services such as energy management, pheromone emission, and distributed discovery that are not supported in existing mobile agent platforms. In addition, unlike existing mobile agent platforms, the bionet platform is fully decentralized and does not require any central entities such as directory servers.

[8]It does not support strong migration, where a CE migrates with both of its data and execution state [13].

[9]In the current implementation of the bionet platform, the communication behavior of CEs is implemented by the bionet message transport, which is not a Bionet Service (see Fig. 1). The bionet message transport is designed as a runtime component separate from the bionet services. This is because the bionet message transport is used not only by CEs but also by the bionet services. For example, during a CE's migration, migration services on two bionet platforms communicate with each other using the bionet message transport.

d) Lifecycle Regulation Behavior: As described in Section III-A, CEs may replicate, reproduce or die as a part of the lifecycle regulation behavior. The bionet platform provides the lifecycle management service (a bionet service), which implements the functionalities necessary to support the lifecycle regulation behavior of CEs (i.e., to initialize, replicate, reproduce, and destroy CEs).

In order to initialize CEs, the lifecycle management service provides the initialization operation. This initialization operation is called when a CE is newly instantiated (either due to a creation of a new CE, replication or reproduction of an existing CE, or on the arrival of a CE from another bionet platform). The initialization operation creates a CE Context, associates the created CE context to the CE, assigns a GUID to the CE,[10] registers the CE to the bionet container, registers the CE to the energy table in the energy management service, and starts running the initialized CE.

In order to replicate CEs, the lifecycle management service provides the replication operation. This replication operation makes a copy (child) of a parent CE using the Java serialization mechanism, possibly executing a mutation on the child CE's behavioral policy, and calls the initialization operation of the lifecycle management service. When a CE reproduces a child CE with another CE, the CE calls the reproduction operation provided by the lifecycle management service. The reproduction operation makes a copy (child) of the CE that called the operation, executes a crossover to inherit the behavioral policies of parent CEs, possibly executing a mutation on the child CE's behavioral policy, and calls the initialization operation of the lifecycle management service. The behavioral policies of CEs evolve through mutation and crossover as described in Sections I and II.[11]

In order to destroy a CE, the lifecycle management service provides the destruction operation. The destruction operation frees the resources (e.g., memory and threads) that a dying CE utilizes, removes an entry for the dying CE from the energy table in the energy management service, and unregisters the dying CE from the bionet container. In the current implementation of the bionet platform, the destruction operation is called only by the energy management service when the energy level of a CE becomes zero. No CEs are allowed to call this operation to destruct other CEs.

e) Relationship Maintenance Behavior: The bionet platform provides the relationship management service (a bionet service), which implements the functionalities necessary to support the relationship maintenance behavior of CEs. The relationship management service allows CEs to establish, examine, update and eliminate their relationships. When a CE establishes a relationship with another CE, it invokes the relationship management service with its relationship partner CE's GUID and/or reference. The service then examines if the specified relationship partner CE exists, and if it does, obtains the relationship partner CE's attributes, and creates a relationship by assigning the obtained attributes to the created relationship.

[10]This step of assigning a GUID is not necessary for a migrated CE. A CE migrated from another bionet platform already has GUID.

[11]Note, however, that the current implementation of the bionet platform does not support evolution mechanisms using mutation and crossover yet. Please see [5] for more details regarding evolution of CEs.

When a relationship of a CE becomes invalid, for example due to migration of a relationship partner CE, the CE that finds the invalid relationship may invoke the relationship management service to update or destroy the relationship.

f) Discovery Behavior: The bionet platform provides the Social Networking Service (a bionet service), which implements the functionalities necessary to support the discovery behavior of CEs. The social networking service allows CEs to discover other CEs with certain attributes by forwarding queries through relationships among CEs. The social networking service defines and implements four key phases in discovery; *query initialization, query matching, query forwarding*, and *query hit backtracking*.

In *query initialization*, a CE (discovery originator CE), begins a discovery process by generating a query with the social networking service. Each query contains its GUID to distinguish it from other queries, a hops-to-live count to determine the scope of discovery, and search criteria that describe the CEs being sought. Search criteria in a query are written in the OMG constraint language [10]. The example below shows the search criteria to seek Web service CEs whose service price is less than 150 energy units

```
serviceType == 'HTTP/1.1'and serviceCost < 150.0.
```

The *query matching* is performed when a discovery originator CE initializes a query or a CE receives a query from another CE. The social networking service provides an evaluator object to examine whether the received query (i.e., the query's search criteria) matches a given CE. If the query matches, a query hit is generated and returned to the discovery originator CE. Otherwise, the query is forwarded to other CEs through relationships among CEs.

In *query forwarding*, queries are forwarded from a CE to another CE through their relationships, seeking the CEs that satisfy given search criteria. At each CE receiving a query, the social networking service decrements the hops-to-live value in a received query, and if the value becomes zero, the query is discarded. Otherwise, the query is forwarded to the relationship partner CEs. In forwarding a query, the social networking service maintains a record of the query's GUID, the CE from which the query is received and the CE to which the query is forwarded.

The *query hit backtracking* is performed when a query matches a CE. A query hit is generated and returned back to the discovery originator CE, following the reverse route of the query forwarding path that led to the CE being returned as a matching query hit.

g) Pheromone Emission Behavior: The bionet platform provides the pheromone emission service (a bionet service), which implements the functionalities necessary to support the pheromone emission behavior of CEs. This service allows a CE to emit and leave its pheromone (i.e., a trace) behind on a bionet platform when it migrates to another bionet platform. A pheromone contains the emitter's GUID and the platform representative of the bionet platform that the emitter migrated to. The pheromone emission service keeps a pheromone list that contains pheromones emitted on the platform that it runs on. In the current implementation of the bionet platform, the pheromone emission service deletes pheromones from its pheromone table after a certain time.

h) Environment Sensing Behavior: As described in Section III-A, CEs may detect various environmental conditions through the environment sensing behavior. In order to support the environment sensing behavior, the bionet platform allows each CE to sense: 1) the CEs running on the same and remote bionet platforms; 2) pheromones emitted on the same and remote bionet platforms by other CEs; 3) resource availability on the same and remote bionet platforms; and 4) network traffic load and traffic patterns on the same and remote bionet platforms. The bionet platform provides several bionet services, each of which implements sensing of each environmental condition described above.

The bionet platform provides the CE sensing service (a bionet service), which allows a CE to sense other CEs on the same and remote bionet platforms. The CE sensing service maintains a list of references to the CEs that are on the local bionet platform, and returns the reference list when invoked by a CE. In order to sense CEs running on remote bionet platforms, the CE sensing service contacts the CE sensing services on a remote bionet platform and obtains a list of the CEs on the remote bionet platform.

The bionet platform provides the pheromone sensing service (a bionet service), which allows a CE to sense the pheromones emitted on the same and remote bionet platforms. When called by a CE, the pheromone sensing service accesses a pheromone list maintained by the pheromone emission service on the same platform and returns the list to the CE. The pheromone sensing service can also find a specific pheromone with the GUID of a CE that left the pheromone. In order to sense the pheromones on remote bionet platforms, a CE asks the pheromone sensing service to contact other pheromone sensing services running on remote bionet platforms.

The bionet platform provides the resource sensing service (a bionet service), which allows a CE to sense resource availability on the same and remote bionet platforms. The resource sensing 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.[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 bionet platforms, a CE asks the resource sensing service to contact the other resource sensing services running on remote platforms.

The bionet container[13] (a runtime component) allows a CE to sense the network traffic load and traffic patterns on the same and remote bionet platforms. It monitors network traffic load on the same bionet platform by counting the number and size of incoming messages. It also monitors network traffic patterns on the same bionet platform by recording the sources of incoming messages. The bionet container also finds the sender CE of an incoming message by obtaining a reference to the sender CE through parsing the incoming message. In order to sense the network traffic load and traffic patterns on remote bionet platforms, a CE asks the bionet container to contact the bionet containers on remote bionet platforms.

TABLE III
CONFIGURATIONS OF PCs USED IN EMPIRICAL EVALUATION

Group	CPU	Memory
A	Intel Celeron 2.0 GHz	512 MB
B	Intel Pentium 4 1.8 GHz	512 MB
C	Intel Pentium 3 1.0 GHz	256 MB
D	Intel Pentium 3 768 MHz	256 MB

IV. EMPIRICAL EVALUATION

This section empirically evaluates the simplicity of developing network applications with CEs. It also empirically examines the efficiency and scalability of the bionet platform.

A. Configurations for Empirical Evaluation

In the empirical evaluation of the bionet platform presented in this section, various measurements were obtained assuming varying numbers of CEs (from 1 through 8000 CEs) and bionet platforms (from 1 through 16 bionet platforms). A maximum of eight Windows 2000 PCs are used in the empirical evaluation, each running the Java 2 standard edition JVMs (version 1.4.2_01 from Sun Microsystems). These eight PCs were divided into four groups of two PCs in each group, depending on their CPU speed and memory size, as shown in Table III. These PCs were connected through 100 Mb/s Ethernet.

B. Application Development Using CEs

In order to examine how the bionet platform reduces the complexity of developing network applications, three different network applications are implemented using CEs.[14]

In the first network application, Web services are implemented using CEs. The body of a Web service CE contains a set of files, accepts HTTP request messages from users, and returns the requested files to the users. In this Web service application, users are also implemented as CEs.

In the second network application, the peer-to-peer content discovery protocol in Gnutella [15] is implemented using CEs. Each CE represents a network node in Gnutella and contains a set of files in its body. The relationships between CEs correspond to the links between Gnutella nodes. Similar to searching for files in Gnutella by forwarding queries through the links among nodes, CE's search for files by forwarding queries through the relationships among CEs.

In the third network application, a new and improved version of Gnutella discovery protocol, GnutellaPlus, is implemented

[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.

[13]In the current implementation of the bionet platform, the bionet container is not a bionet service (see Fig. 1), It is designed as a runtime component separate from the bionet services. This is because the bionet container is used not only by CEs but also by the bionet services. For example, during a CE's migration, the bionet container (and the bionet message transport) are used by the migration services on two bionet platforms to communicate with each other.

[14]A software engineering discipline suggests investigating at least three applications on a framework in order to examine generality and reusability of the framework [14].

TABLE IV
APPLICATIONS IMPLEMENTED USING THE BIONET PLATFORM

Application	# of Classes	Lines of Code	Development Time
Web Server	3	114	2 hours
Gnutella	5	309	1/2 day
GnutellaPlus	5	324	1/2 day

using CEs. Unlike Gnutella, which searches for files with file names, GnutellaPlus allows CEs to specify name-value pairs as file search criteria. For example, with GnutellaPlus, CEs can search for files with their keywords, the date when they were created/revised or the names of their authors.

Table IV shows the empirical evaluation of the three network applications described above. It shows that it is fairly simple and easy to implement network applications on the bionet platform. The reusable components in the bionet platform help to reduce the lines of code and development time.

C. Empirical Evaluation of the Bionet Platform

This section describes empirical evaluation of the bionet platform and includes eight separate measurements using different configurations. The first five measurements use one or two PCs of the group A configuration shown in Table III. The last three measurements use eight PCs (two PCs from each of the four configurations shown in Table III).[15]

1) Overhead of Bionet Platform Initialization: In order to evaluate the overhead of initializing a bionet platform, Table V shows the bootstrap overhead (i.e., the time for the bionet platform to initialize each runtime component) and the bootstrap memory footprint (i.e., the amount of memory space each runtime component consumes when it is initialized) for each runtime component. Table V shows that a bionet platform initializes its runtime components efficiently with small memory footprint.

2) Overhead of CE Deployment: In order to evaluate the efficiency of deploying a CE on a bionet platform, Table VI shows the time required for the bionet platform to execute key steps of CE deployment. The measurement examined the following key deployment steps; instantiating a CE, initializing an instantiated CE with the Lifecycle Management Service, locating the CEs on the same bionet platform with the CE Sensing Service, and establishing relationships to the located CEs with the relationship management service. In the step to instantiate a CE, the measurement examined two cases; the case where a human developer manually instantiates a new CE, and the case where a parent CE replicates (makes a copy of) itself. Table VI shows that the overhead of deploying a CE is small and that the bionet services used to deploy a CE are efficient. The overhead difference between the two cases to instantiate a CE is due to the time to make a copy of a parent CE in the replication process.

[15]Please note that although there exists research to empirically evaluate the scalability of agent platforms (such as the bionet platform described in the paper), the number and the scope of such empirical evaluation is rather limited [12]. For instance, Auctionbot is concluded scalable through a measurement with only 90 agents [16], illustrating difficulty of empirically examining scalability of agent platforms. In this paper, the scalability of the bionet platform was examined with 8000 CEs and 16 bionet platforms.

TABLE V
BOOTSTRAP OVERHEAD AND MEMORY FOOTPRINT OF EACH PLATFORM COMPONENT

Platform Components		Bootstrap Overhead	Memory Footprint
Bionet Class Loader		9.11 msec	3.97 KB
Bionet Message Transport		22.98 msec	6.65 KB
Bionet Container		127.06 msec	8.88 KB
Bionet Services	Migration	33.13 msec	4.88 KB
	Energy Management	59.02 msec	8.12 KB
	Lifecycle Management	91.92 msec	44.07 KB
	Relationship Management	23.17 msec	4.48 KB
	Social Networking	69.85 msec	12.03 KB
	Pheromone Emission	26.68 msec	8.37 KB
	CE Sensing	56.43 msec	7.82 KB
	Pheromone Sensing	25.46 msec	7.92KB
	Resource Sensing	64.36 msec	42.12 KB
Platform Representative		82.31 msec	5.23 KB
Total		691.48 msec	162.64 KB

TABLE VI
OVERHEAD OF CE DEPLOYMENT

Deployment Steps		Overhead
Instantiate a CE	Manually create a new CE	14.94 msec
	Replicate a parent CE with the Lifecycle Mgt Service	104.58 msec
Initialize a CE with the Lifecycle Management Service		180.02 msec
Sense 100 CEs on the same platform using the CE Sensing Service		723.59 msec
Establish relationships to the 100 CEs using the Relationship Mgt Service		1,257.82 msec
Total	Deploy a new CE	2,176.37 msec
	Deploy a replicated CE	2,277.22 msec

3) Message Transmission Latency and Throughput: In order to examine the time required for a message to travel between two CEs on different bionet platforms (i.e., the message transmission latency) and the number of messages that such CEs can exchange per second (i.e., throughput), in the following measurements, a single CE (a sender CE) is deployed on a bionet platform, and the varying number of CEs (receiver CEs) are deployed on another bionet platform to receive messages from the sender CE. The two bionet platforms run on different PCs. The number of receiver CEs varies from 1 to 1000, and the sender CE randomly chooses one of the receiver CEs and sends an empty message to the receiver CE. The bionet message transport and the bionet container perform message transmission. On the sending bionet platform, the bionet message transport creates a message, establishes a TCP connection to the receiving bionet platform, and transmits a message on the connection. On the receiving bionet platform, the bionet message transport accepts

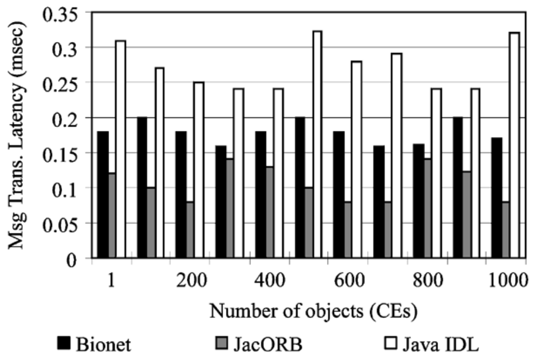

Fig. 4. Message transmission latency.

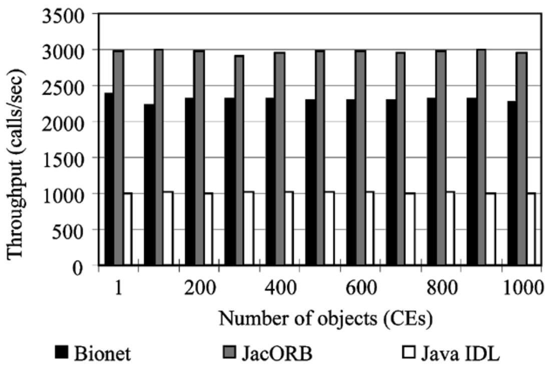

Fig. 5. Throughput.

the incoming message, and the bionet container dispatches the message to a receiver CE.

Fig. 4 shows the message transmission latency in the bionet platform.[16] It also shows the message transmission latency in well-known Java-based distributed object platforms (JacORB [17] and Java IDL [18]) for the purpose of comparison. Fig. 4 shows that the message transmission latency in the bionet platform is small and comparable with the other distributed object platforms. Fig. 4 also shows that the message transmission latency remains relatively constant as the number of receiver CEs increases, indicating that the bionet platform (i.e., the bionet message transport and bionet container) scales well. This is because the bionet message transport creates only one TCP connection between the sending and receiving platforms, and the sender CE transmits messages to multiple receiver CEs over the same TCP connection. The bionet message transport does not create a separate connection for each receiver CE.

Fig. 5 shows the throughput between two CEs on different platforms.[17] Similar to Fig. 4, Fig. 5 compares the throughput of JacORB and Java IDL. Fig. 5 shows that the throughput of the bionet platform (bionet message transport and bionet container) is comparable with existing distributed object platforms. This figure also shows that the throughput remains mostly constant as the number of CEs increases, indicating that the bionet container scales well. This is because the bionet container implements a hash-based table that contains references to the CEs on the same bionet platform. The overhead for the hash-based table to dispatch an incoming message to a target CE does not change even if the number of CEs increases.

[16]Note that the bionet message transport and bionet container of the bionet platform contribute to the message transmission latency.

[17]Note that the bionet message transport and bionet container of the bionet platform contribute to the throughput.

TABLE VII
OVERHEAD OF DISCOVERY USING SOCIAL NETWORKING SERVICE

Phases in Discovery	Overhead
Query Initialization	19.23 msec
Query Forwarding	29.33 msec
Query Matching	12.82 msec
Query Hit Backtracking	38.84 msec
Total	100.22 msec

4) Overhead of Discovery Using the Social Networking Service: In order to evaluate the overhead of the social networking service in discovery, GnutellaPlus, one of the three network applications described in Section IV-B, is used in this measurement. Two CEs (i.e., a discovery originator CE and a discovery target CE that matches the search criteria) are deployed on two different bionet platforms (on different PCs). The search target CE has the attributes described in Table I.

In discovery, the discovery originator CE issues a query that contains `serviceType` == ‘`HTTP/1.1`'`and serviceCost` < `150.0` as discovery criteria using the social networking service (query initialization phase). The discovery originator CE forward the query to a relationship partner CE using the social networking service (query forwarding phase). Upon the receipt of a query, the relationship partner CE examines whether the discovery criteria in the query matches its attributes using the social networking service (query matching phase), and if they match, returns a query hit to the discovery originator CE using the social networking service (query hit backtracking phase).

Table VII shows that the overhead of the social networking service in each phase of discovery is small. Note also that the time to perform query matching is very small (under 13 ms), and this is because the social networking service caches a received query and bypasses the overhead of parsing search criteria in subsequent queries. Although this measurement is for two CEs, given the small overhead shown in Table VII, the social networking service is expected to scale well for discovery in larger scale.

5) Overhead of Migration Using the Migration Service: In order to evaluate the time required for a CE to migrate from a bionet platform to another bionet platform using the migration service, two bionet platforms are deployed on different PCs, and CEs of varying sizes from 31 KB to 8 MB migrate from a bionet platform to the other using the migration service. The overhead of migration includes the time for the migration service on an origin bionet platform to serialize a CE into mobile code, the time for the bionet message transport to transmit the mobile code from the origin bionet platform to the destination bionet platform, and the time for the migration service on the destination platform to deserialize the incoming mobile code and instantiate a CE.

Fig. 6 shows that the overhead of the migration service is small and that the migration service allows CEs to efficiently migrate from a platform to a platform. Note also that a measurement study for Aglets [19], a well-known mobile agent platform, indicates that the migration service and Aglets' migration

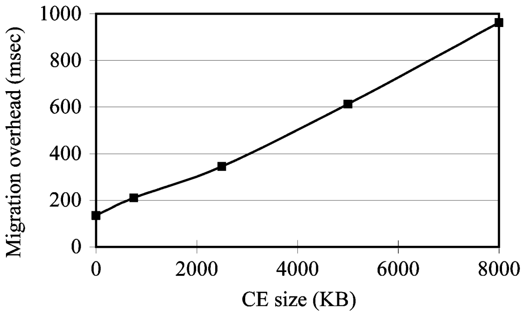

Fig. 6. Migration overhead.

TABLE VIII
OVERHEAD OF ENVIRONMENT SENSING WITH THE PHEROMONE SENSING SERVICE

Phases in Pheromone Sensing	Overhead
Access a pheromone list	17.28 msec
Contact a remote Platform Representative	69.41 msec
Contact a remote CE Sensing Service to locate a target CE	75.84 msec

service are comparable in their performance. Fig. 6 also shows that, as the size of CE increases, the overhead of the migration service increases linearly, indicating that the migration service scales.

6) Overhead of Environment Sensing Using the Pheromone Sensing Service: In order to evaluate the time required for pheromone sensing (i.e., environment sensing) using the pheromone sensing service, 16 bionet platforms are deployed on eight PCs (two platforms on each PC), and two CEs are deployed. One CE randomly migrates 15 times between these bionet platforms, and when it migrates, it leaves a pheromone behind on a bionet platform. The other CE senses pheromones emitted by the migrating CE to locate it. The overhead of pheromone sensing includes the time for the pheromone sensing service to find the migrating CE's pheromone by accessing a pheromone list maintained by the pheromone emission service, contact a representative of the bionet platform that the pheromone specifies (i.e., the platform that the CE migrated to), and locate the migrated CE on the remote bionet platform.

Table VIII and Fig. 7 show the overhead of pheromone sensing using the pheromone sensing service. Table VIII shows the overhead in each phase of pheromone sensing. Fig. 7 shows how the overhead changes when a CE senses pheromones emitted on remote bionet platforms that are multiple hops away. Table VIII illustrates that the overhead of the pheromone sensing service is small and that the pheromone sensing service efficiently performs pheromone sensing. Fig. 7 demonstrates that the overhead increases linearly, as the hop count to the remote bionet platform increases, indicating that the pheromone sensing service scales.

7) Latency in Relationship Examination and Energy Exchange Between CEs: As described in Sections III-A and III-C, a CE may receive a service from another CE that it has a relationship to in exchange for energy. Receiving a service involves using the relationship management service to examine if there is a valid relationship to the CE that provides a desired

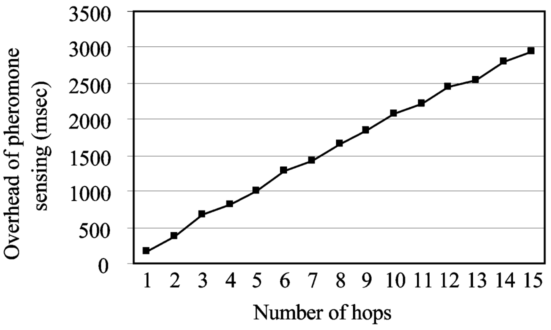

Fig. 7. Overhead of environment sensing (pheromone sensing) using pheromone emission services.

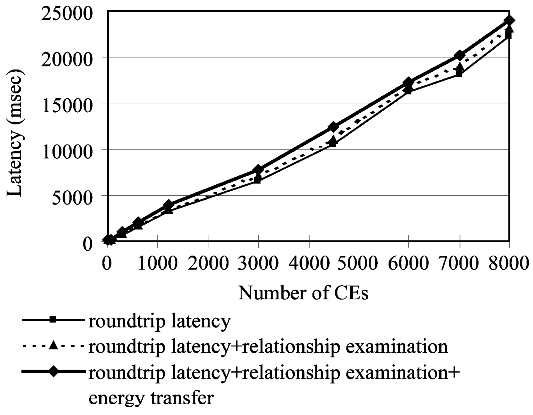

Fig. 8. Latencies in round-trip latency, relationship examination, and energy transfer between CEs.

service and the energy management service to pay energy units for the service received. In order to evaluate the time for the relationship management service to examine relationships and the time for the energy management service to exchange energy, 16 bionet platforms are deployed on eight PCs (two platforms on each PC), and the varying number of CEs are randomly deployed on the bionet platforms. The number of CEs varies from 1 to 500 on each platform (i.e., 16–8000 CEs in total). Each CE randomly chooses a CE and establishes a relationship with it. Each CE continuously requests a service to its relationship partner CE by sending an empty message (considered as a service request). It examines if its relationship is valid[18] before sending a message using the relationship management service. Upon receiving a service request message, a relationship partner CE immediately sends back a reply message (considered as a service) to the CE that sent the service request message. When receiving a reply service (a service), a CE transfers 100 energy units to its relationship partner CE using the energy management service.

Fig. 8 illustrates the overhead of examining relationships using relationship management service (denoted as "relationship examination" in Fig. 8) and the overhead of exchanging energy using the energy management service (denoted as "energy transfer" in Fig. 8). For the purpose of comparison, Fig. 8 also shows the delay to exchange a service request message and a service message between two CEs (denoted as "round-trip

[18]In this measurement, when a CE examines its relationship, the relationship is always valid because its relationship partner CE does not migrate nor die.

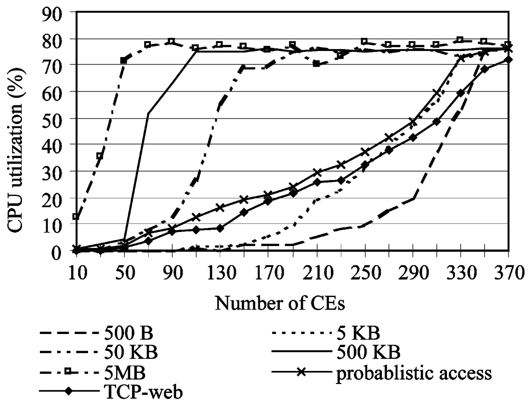

Fig. 9. CPU utilization of the bionet platform and CEs.

latency" in Fig. 8). Fig. 8 demonstrates that the overhead of examining relationships and that of exchanging energy are small, compared with the round-trip latency of exchanging messages between two CEs. For instance, for 8000 CEs, the overhead of examining relationships and the overhead of exchanging energy are 4.7% and 7.1%, respectively, of the round-trip latency. This demonstrates that the relationship management service and the energy management service are efficient. Fig. 8 also illustrates that the overhead increases linearly, as the number of platforms and CEs increases, indicating that relationship management service and energy management services scale.

8) CPU Utilization of the Bionet Platform and CEs: In order to evaluate how much CPU power the bionet platform and CE's consume, CEs that implement the Web service described in Section IV-B are deployed on a bionet platform. In this measurement, each Web service CE contains in its body five files whose sizes are 500 B, 5 KB, 50 KB, 500 KB, and 5 MB. These file sizes are taken from the specification of Webstone [20], a performance profiling tool for Web servers. Note that each CE contains an identical set of files, and there is no sharing of files among CEs. In addition, a user is implemented as a CE, and a user CE is deployed on the same bionet platform as Web service CEs. A user CE sends HTTP request messages to a randomly selected Web service CE, and the request rate of the user CE is set at ten requests per second.

Fig. 9 shows the CPU utilization by the bionet platform and Web service CEs. Note that as the underlying operating system consumes approximately 25% of the CPU power, the total CPU utilization of the PC used for this measurement reaches 100% when the CPU utilization by the bionet platform and Web service CEs reach approximately 75%. In Fig. 9, the CPU utilization for a specific file size is obtained when a user CE always requests the given file size (among the five files that Web CEs house). Fig. 9 also shows the CPU utilization when a user CE probabilistically requests files of different sizes. Probabilities that a user CE follows to request files are taken from WebStone [20] and are shown in Table IX. The CPU utilization of the probabilistic access case on Fig. 9 shows that approximately 330 CEs can simultaneously run on a platform under 75% CPU utilization. Fig. 9 also shows that the CPU utilization increases almost linearly as the number of CEs increases up to 290. The bionet platform scales well to the number of CEs.

TABLE IX
PROBABILITY OF FILE REQUESTS

File Sizes	Probability
500 B	35%
5 KB	50%
50 KB	14%
500 KB	0.9%
5 M	0.1%

For the purpose of comparison, Fig. 9 shows the CPU utilization (denoted as "TCP-Web" in Fig. 9) of a Web server implemented in a conventional manner. As with the Web service CEs, each conventional Web server houses five files of different sizes, and the number of Web servers is varied in Fig. 9. Further, a user randomly selects a conventional and probabilistically accesses one of the files on the Web server. Unlike Web service CEs, which use the bionet platform (the bionet message transport and bionet container) to exchange messages, the conventional Web servers use the TCP interface of the underlyng operating system to exchange messages. Fig. 9 shows that the difference in the CPU utilization between the Web service CEs (denoted as "probabilistic access") and the conventional Web servers (denoted as "TCP-Web") is small, indicating that the bionet platform does not impose significant performance overhead on network applications.

V. Concluding Remarks

This paper describes and empirically evaluates the middleware platform for a new network architecture, called the Bio-Networking Architecture. With biologically inspired principles and mechanisms, network applications created based on the Bio-Networking Architecture satisfy the key requirements of future network applications such as autonomy, scalability, adaptability, and simplicity. The empirical evaluation shows that the platform is efficient, scalable, reusable, and significantly simplifies development of network applications.

An extended set of empirical measurements are being planned to provide additional performance implications of the bionet platform. Further deployment of the bionet platform and CEs on more realistic environments (e.g., PlanetLab [21]) would identify the impact of the network size and realistic constraints on the bionet platform performance.

Acknowledgment

The authors would like to thank three anonymous reviewers, as well as M. Moore, Y. Pan, J. Lu, T. Nakano, and K. Fujii for their valuable comments and contributions that improved the quality of the paper.

References

[1] 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 *The Internet as a Large-Scale Complex System*, K. Park, Ed. Princeton, NJ: Princeton Univ. Press, Feb. 2005.

[2] M. Wang and T. Suda, "The bio-networking architecture: A biologically inspired approach to the design of scalable, adaptive, and survivable/available network applications," in *Proc. 1st IEEE SAINT*, 2001, pp. 43–56.

[3] T. Itao, S. Tanaka, T. Suda, and T. Aoyama, "A framework for adaptive UbiComp applications based on the jack-in-the-net architecture," *Kluwer/ACM Wireless Network J.*, vol. 10, no. 3, pp. 287–299, 2004.

[4] J. Suzuki and T. Suda, "Design and implementation of a scalable infrastructure for autonomous adaptive agents," in *Proc. 15th IASTED Int. Conf. Parallel Distrib. Comput. Syst.*, Nov. 2003, pp. 594–603.

[5] T. Nakano and T. Suda, "Adaptive and evolvable network services," in *Proc. GECCO*, Jun. 2004, pp. 151–162.

[6] S. Sameshima, J. Suzuki, S. Steglich, and T. Suda. (2004, Apr.) Platform independent model (PIM) and platform specific model (PSM) for super distributed objects. OMG final recommended spec.. [Online]. Available: http://www.omg.org/cgi-bin/doc?sdo/03-09-01

[7] R. Albert, H. Jeong, and A. Barabasi, "Error and attack tolerance of complex networks," *Nature*, vol. 406, pp. 378–382, Jul. 2000.

[8] N. Minar, K. H. Kramer, and P. Maes, "Cooperating mobile agents for dynamic network routing," in *Software Agents for Future Communications Systems*, A. Hayzelden, Ed. New York: Springer-Verlag, 1999, ch. 12.

[9] Object Management Group, The CORBA Specification, Version 3.0, 2002.

[10] Object Management Group, "The Trading Object Service,", 2000.

[11] D. Lange and M. Oshima, *Programming and Deploying Java Mobile Agents with Aglets*. Reading, MA: Addison-Wesley, 1998.

[12] N. J. E. Wijngaards, B. J. Overeinder, M. van Steen, and F. M. T. Brazier, "Supporting Internet-scale multi-agent systems," *Data Knowl. Eng.*, vol. 41, no. 2–3, pp. 229–245, 2002.

[13] A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding code mobility," *IEEE Trans. Softw. Eng.*, vol. 24, no. 5, pp. 342–361, 1998.

[14] D. Roberts and R. Johnson *et al.*, "Evolving frameworks: A pattern language for developing object-oriented frameworks," in *Pattern Languages of Program Design*, R. Martin *et al.*, Eds. Reading, MA: Addison-Wesley, 1997, vol. 3, ch. 26.

[15] [Online]. Available: http://gnutella.wego.com

[16] P. Wurman, M. Wellman, and W. Walsh, "The michigan Internet auctionbot: A configurable auction server for human and software agents," in *Proc. Agents'98*, 1998, pp. 301–308.

[17] G. Brose, "JacORB: Implementation and design of a Java ORB," in *Proc. IFIP DAIS'97*, Sep. 1997, pp. 143–154.

[18] G. Lewis, S. Barber, and E. Siegel, *Programming with Java IDL*. New York: Wiley, 1997.

[19] D. Hagimont and L. Ismail, "A performance evaluation of the mobile agent paradigm," in *ACM SIGNAL Notices*, vol. 34, 1999, pp. 306–313.

[20] G. Trent and M. Sake, *WebStone: The First Generation in HTTP Server Benchmarking*. Los Gatos, CA: Mindcraft, 1995.

[21] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Spalink, T. Roscoe, and M. Wawrzoniak, "Operating system support for planetary-scale services," in *Proc. 1st Symp. Network Syst. Design Implementation*, Mar. 2004, pp. 253–266.

Junichi Suzuki (M'99) received the Ph.D. degree in computer science from Keio University, Keio, Japan, in 2001.

He joined the Department of Computer Science, University of Massachusetts, Boston, in September 2004, where he is currently an Assistant Professor. From 2001 to 2004, he was with the School of Information and Computer Science, University of California, Irvine (UCI), as a Postdoctoral Research Scholar. Before joining UCI, he was with Object Management Group Japan, Inc., as a Technical Director. His research interests include autonomous adaptive distributed systems, biologically inspired software adaptation, self-organizing overlay networks, and model-driven software development.

Dr. Suzuki is a member of the Association for Computing Machinery (ACM). He is an active participant and contributor of the International Standard Organization SC7/WG19 and the Object Management Group, Super Distributed Objects SIG.

Tatsuya Suda (S'80–M'82–SM'97–F'01) received the B.E., M.E., and Dr.E. degrees in applied mathematics and physics from Kyoto University, Kyoto, Japan, in 1977, 1979, and 1982, respectively.

From 1982 to 1984, he was with the Department of Computer Science, Columbia University, New York, as a Postdoctoral Research Associate. Since 1984, he has been with the Department of Information and Computer Science, University of California, Irvine, where he is currently a Professor. He has also served as a Program Director of the Networking Research Program, National Science Foundation from 1996 to 1999. He was a Visiting Associate Professor at the University of California, San Diego, a Hitachi Professor at the Osaka University, and currently is a NTT Research Professor. He is an Area Editor of the *International Journal of Computer and Software Engineering*. He has been engaged in research in the fields of computer communications and networks, high-speed networks, multimedia systems, ubiquitous networks, distributed systems, object oriented communication systems, network applications, performance modeling and evaluation, and application of biological concepts to networks and network applications.

Dr. Suda is a member of the Association for Computing Machinery (ACM). He received an IBM Postdoctoral Fellowship in 1983. He was the Conference Coordinator from 1989 to 1991, the Secretary and Treasurer from 1991 to 1993, the Vice Chairman from 1993 to 1995, and the Chairman from 1995 to 1997 of the IEEE Technical Committee on Computer Communications. He was also the Director of the U.S. Society Relations of the IEEE Communications Society from 1997 to 1999. He is an Editor of the IEEE/ACM TRANSACTION ON NETWORKING, a Senior Technical Consultant to the IEEE TRANSACTIONS ON COMMUNICATIONS, and a former Editor of the IEEE TRANSACTION ON COMMUNICATIONS. He is a member of the Editorial Board of the Encyclopedia of Electrical and Electronics Engineering, Wiley. He was the Chair of the 8th IEEE Workshop on Computer Communications and the TPC Co-Chair of the IEEE INFOCOM 1997.