Proceedings of the 15th IASTED International Conference
PARALLEL AND DISTRIBUTED COMPUTING AND SYSTEMS
November 3-5, 2003, Marina Del Rey, CA, USA
Selected as a best paper candidate (one of the 4 candidates out of accepted 150 papers)

Design and Implementation of a Scalable Infrastructure for Autonomous Adaptive Agents*

Junichi Suzuki and Tatsuya Suda
School of Information and Computer Science
University of California, Irvine
Irvine, CA 92697

Abstract

This paper describes our research effort to design, implement and deploy a scalable infrastructure for autonomous adaptive agents running on the Internet. We have designed a network application architecture, called the Bio-Networking Architecture, which models agents after several biological concepts and mechanisms, and implemented a platform software to host the architecture on the Internet. The platform aids developing and executing large-scale, highly distributed and dynamic network applications, each of which is composed of the biologically-inspired software agents, by abstracting low-level networking/operating details and providing a rich set of runtime services. We overview several key features of the agents in our architecture, and describe the design and implementation of the proposed platform, showing how the platform satisfies a set of functional requirements derived from the features of our agents. We also present some measurement results to examine scalability and efficiency of the platform.

Key Words

software agents, autonomous adaptive agents, agent platform, middleware, distributed computing software

1. Introduction

We believe the future network applications, which will be orders of magnitude more complex and larger than the current ones, should exhibit self-organization with inherent support for scalability, mobility and adaptability to dynamic changes in networks. In order to make this vision a reality, it is recognized as one of the promising strategies to use autonomous adaptive agents in network applications [1, 2]. Although a lot of existing research efforts have successfully clarified autonomous adaptive agents and showed they work well in some applications (e.g. [3, 4]), the number of large-scale agent systems are currently very limited [5]. Even in agent simulation systems, the scale of agents involved is often kept small, except several exceptions (e.g. [6]). The scale of agent systems running on actual networks is usually much smaller. For example, the claim that Auctionbot [7] is scalable is supported by an experiment with only 90 agents.

This paper describes our research effort to develop a scalable infrastructure that allows for deploying large-scale, highly distributed and dynamic network applications using autonomous adaptive agents. Our long-term research goal is to exploit autonomous adaptive agents, beyond simulations, for Internet-based distributed computing. In order to achieve this goal, we have designed a novel architecture, called the Bio-Networking Architecture, which models autonomous adaptive agents after several biological concepts and mechanisms [8, 9]. The architecture is motivated by the observation that the desirable properties in future network applications (such as scalability and adaptability) have already been realized in various biological systems.

This paper overviews several key features of the agents in our architecture, and identifies functional requirements to our agent infrastructure, called the Bio-Networking Platform (or bionet platform). The bionet platform is a middleware system that aids developing and deploying network applications (agents) by providing reusable software components. These components abstract low-level operating and networking details (e.g. I/O and concurrency), and provide agents a series of runtime services. We describe the design and implementation of the bionet platform, showing how the platform satisfies the identified requirements. In order to examine scalability and efficiency of the platform, we present some results of the initial empirical measurements.

This paper is organized as follows: Section 2 presents several key features of the agents in our architecture. Section 3 describes the design and implementation of the bionet platform, showing how the platform satisfies a set of functional requirements derived from the agent features. Measurement results are shown in Section 4. In Sections 5 and 6, we conclude with comparison with existing research work and future work.

2. Assumed Agent Features

In the Bio-Networking Architecture, each agent, called *cyber-entity*, consists of attributes, body and behaviors [8]. Attributes carry descriptive information regarding a

* Research supported by the NSF through grants ANI-0083074 and ANI-9903427, by DARPA through grant MDA972-99-1-0007, by AFOSR through grant MURI F49620-00-1-0330, and by grants from the California MICRO program, Hitachi, Hitachi America, Novell, NTT, NTT Docomo, Fujitsu, and NS Solutions Corporation.

cyber-entity (e.g. identifier). A body implements cyber-entity's functional service(s). Behaviors implement non-functional biological actions (e.g. reproduction and migration). Each cyber-entity lives on a specific bionet platform to execute its functional service implemented in its body. A bionet platform runs on each network node. Through a set of runtime services provided by a local bionet platform, each cyber-entity continuously sense the current surrounding conditions in network (e.g. network traffic) and performs its behavior [8, 9, 10, 11]. Cyber-entities maintain the following four key features.

(1) Decentralized. A network application is modeled as a decentralized collection of cyber-entities in the Bio-Networking Architecture. This is analogous to a bee colony (an application) consisting of multiple bees (cyber-entities). The ultimate advantage of decentralization is scalability and fault tolerance [12, 13]. Centralized systems can fail when central entities (e.g. directory server and resource manager) are overwhelmed or down, but decentralized systems can survive by spreading the load [14] or avoiding fatal errors even when some agents go down [15]. Decentralization is essential if a system grows beyond the management of a single administrative entity. Centralized entities also suffer from mobility of agents. They cannot eventually keep track of agents in highly dynamic environments where agents often join and leave the network [16]. Decentralized systems also have an organizational advantage. Users need no complicated setup work; they can simply develop and run their cyber-entities without knowing any central coordination. This lowers the barrier for developing and accessing autonomous adaptive agents.

(2) Autonomous. Autonomy is the ability of agents to act without any intervention from their users and other agents [17]. Autonomous agents are goal-oriented and control themselves in a proactive manner [18]. Cyber-entities are autonomous in the sense that each of them has its own goal (e.g. staying close to users and living long), senses surrounding network conditions, and performs its behaviors, according to the sensed environmental conditions, which will support future goal achievement [9]. Our previous simulation work has confirmed that the desirable system characteristics (e.g. adaptability and survivability) emerge as collective results of cyber-entities' autonomous behavior invocation and decentralized interactions among them [9].

(3) Adaptive. Adaptability is the ability of agents to increase their fitness to environment. Cyber-entities adapt themselves to environmental changes in short-term and long-term fashions. The short-term adaptation is achieved by performing behaviors according to the current surrounding network conditions [9, 11]. For example, a cyber-entity may migrate to a neighboring platform when traffic volume grows or resource availability becomes scarce. The long-term adaptation is achieved by applying biological evolutionary concepts. Cyber-entities evolve by generating behavioral diversity and executing natural selection [10]. Behavioral diversity means that it is likely different cyber-entities implement different policies on their behaviors. It is generated through mutation and crossover, which dynamically modify behavior policies during replication and reproduction. Natural selection is executed based on the concept of *energy*. Each cyber-entity stores and expends energy for living, as biological entities naturally strive to gain energy by seeking and consuming food. Cyber-entities gain energy in exchange for performing their functional services, and expend energy to consume resources such as CPU cycles and memory space. The abundance and scarcity of stored energy affects various behaviors and contributes to the natural selection process. For example, abundance of stored energy is an indication of higher demand for the cyber-entity; thus the cyber-entity may be designed to favor reproduction in response to higher level of stored energy. Scarcity of stored energy (an indication of lack of demand or ineffective behaviors) may eventually cause the cyber-entity's death. Our previous simulation work has shown that our evolutionary process allows cyber-entities to adapt to dynamic environmental changes (e.g. changes in workload, user's location and resource availability) [10].

(4) Self-describing. In order to make agents autonomous and decentralized, they need to be loosely coupled with each other. As a result, the agents that an agent interacts with may not exist at the point of time it is developed, and they may not always be available in the future, for example, due to relocation and shutdown/upgrade by their administrator. Therefore, autonomous decentralized agents should be able to dynamically discover and interact with other agents without recompiling or changing any lines of code. In the Bio-Networking Architecture, each cyber-entity keeps its own descriptive information as attributes, and makes it available to other cyber-entities. It also maintains a set of *relationships* with other cyber-entities. A relationship is established between two cyber-entities, and it contains descriptive information about a peer cyber-entity. With the relationships and descriptive information, cyber-entities dynamically discover others and interact with each other [19].

3. The Bio-Networking Platform

Given an initial set of successful simulation results [9, 10, 11, 19], we built the bionet platform in order to implement and evaluate the features of cyber-entities on real networks. This section describes the design and implementation of the bionet platform.

3.1 Software Architecture

We implemented the bionet platform in Java, and each platform runs on a Java virtual machine (JVM) atop a network node. It is an object-oriented configurable and

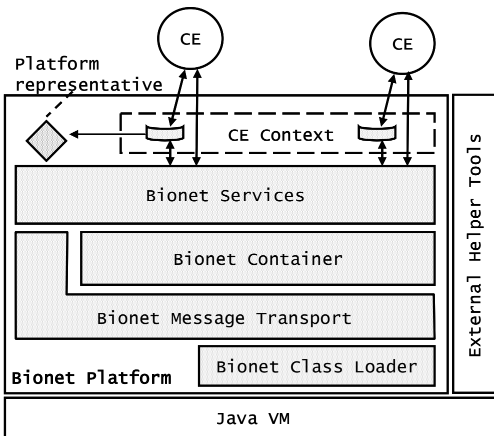

Figure 1. Architecture of the bionet platform

extensible framework on which various network applications can be developed. The bionet platform consists of six architectural components (see Figure 1).

A *platform representative* is an object that represents a bionet platform and runs on per-platform basis. It keeps a table listing all the bionet services and bionet container (see below) on a local platform with their names and references. It is initialized when a bionet platform boots.

A *CE context* is an entry point for a cyber-entity to access underlying platform components (e.g. bionet services). It examines if a bionet service requested by a cyber-entity is available, and if it is, the CE context returns a reference to the service. Each CE context performs this lookup for bionet services through the local platform representative. Each cyber-entity has its own CE context. A CE context is created and associated with a cyber-entity by the lifecycle service (one of the bionet services, see Section 3.3), when the cyber-entity is created, replicated, reproduced or completes a migration.

The *bionet services* provide a set of runtime services that cyber-entities use for performing their behaviors. Each bionet service implements one or more behaviors of cyber-entities. The behaviors the current bionet services support are *energy exchange and storage*, *migration*, *replication and reproduction*, *relationship maintenance*, *discovery of cyber-entities*, and *resource sensing*.

The *bionet message transport* abstracts low-level networking and operating details such as network I/O, concurrency, messaging and network connection management. The current bionet platform uses the CORBA IIOP ver. 1.1 [20] to transmit messages on TCP.

The *bionet container* maintains a table that contains references to the cyber-entities running on a local platform, and dispatches incoming messages to them. It follows the interfaces of the CORBA Portable Object Adaptor (POA) [20] for the table maintenance and message dispatching. It also keeps track of the network traffic load by counting the size of received IIOP packets and the number of method dispatches.

The *bionet class loader* is a custom class loader that extends JVM's system (default) class loader. It is used to

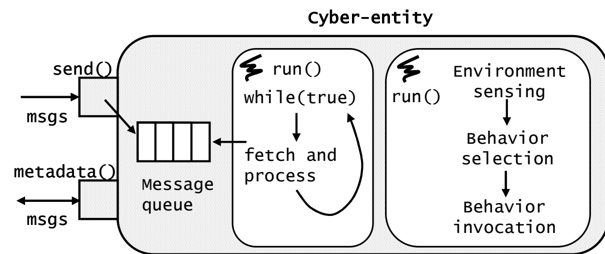

Figure 2. Internal design of a cyber-entity

dynamically load a cyber-entity's class definition into a JVM when it is newly created or completes a migration.

External helper tools are the software intended to improve the productivity of developers and administrators. They include GUI tools to visualize cyber-entities' attributes, relationship structures and performance measurement results. Appendix shows an example tool, which monitors response time of a cyber-entity and graphically displays that on a window.

The current code base of the bionet platform contains approximately 29,700 semicolons, and is the work of one full-time research staff and six part-time undergraduate students. It has been open for public use at UC Irvine since 2002 [21], and will be released soon for researchers who explore the design space of autonomous adaptive agents and investigate them on the Internet.

We implemented the bionet platform in Java for several reasons. The most important was speed of development. Unlike C and C++, Java supports strong typing and automated runtime garbage collection. These two features greatly reduce debugging time, especially in a large-scale project with a rapid development pace. The second reason was portability. Multithreaded code in Java is much easier to port than the one in C or C++. In fact, our code base, which was implemented and tested solely on Window 2000/XP PCs, was ported onto Solaris in under a week of part-time work.

We implemented the bionet message transport and bionet container based on the CORBA IIOP and POA specifications, respectively. A reason of this choice was language neutrality. The cyber-entities written in Java can interoperate with the programs in C++, Lisp and even script languages such as Python. Another reason was portability. The programs compliant with the CORBA interfaces are easy to port from a CORBA implementation to another. Our code base, which contains our own CORBA implementation, was ported onto JacORB [1], which is another Java-based CORBA implementation, within two days of part-time work.

3.2 Design of Cyber-entity

Since the bionet platform uses Java as an implementation language and CORBA IIOP as a message transport protocol, a cyber-entity is designed as a Java object

[1] www.jacorb.org

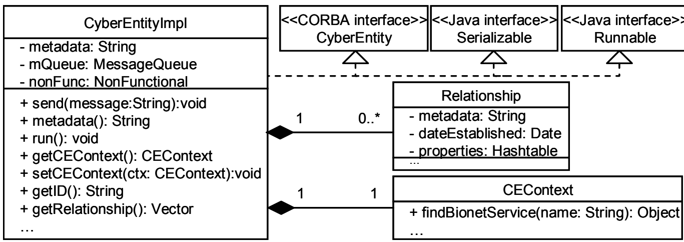

Figure 3. Class diagram around `CyberEntityImpl`

implementing a CORBA interface. Every cyber-entity implements the following CORBA interface.

```
interface CyberEntity {
  oneway send(in string message);
  string metadata();};
```

Cyber-entities use `send()` to communicate with each other in an asynchronous manner. The operation accepts a message from another cyber-entity as its parameter. We use a subset of the FIPA agent communication language[2] for the message format. Due to space limitation, please see [19] for more details about the message format. The reason we chose an asynchronous message-based communication scheme, instead of a synchronous request-reply scheme, is that the scheme can provide better scalability in terms of response time and throughput [34]. It also contributes to the loose-coupling among cyber-entities, described in Section 2. The `send()` operation inserts a received message in cyber-entity's message queue (Figure2). The cyber-entity fetches the message to process it on an individual thread. When no message is available, the thread `waits` for a new message on the queue. When a cyber-entity migrates to another platform, all the unprocessed (queued) messages are transmitted and processed at a destination platform.

Each cyber-entity maintains another thread to perform its non-functional logic including environment sensing, behavior selection and behavior invocation (Figure 2). It is implemented as a subclass of `java.util.TimerTask`, and executed at certain intervals. We assigned different threads to functional and non-functional aspects, because it is different how often these aspects need to be executed; the functional aspect should be executed immediately when a message is queued and the non-functional aspect can be executed on the order of seconds, minutes or maybe even hours, depending on application requirements. Please note that it is beyond of the scope of this paper to describe the behavior selection scheme (i.e. which behavior to be selected in given network conditions). Please see [9, 10, 11] for details about this issue.

The `metadata()` operation of `CyberEntity` is used to obtain a cyber-entity's attributes. The mandatory attributes that every cyber-entity must maintain are (1) the cyber-entity's GUID (globally unique ID), (2) the cyber-entity's reference, (3) the type of service the cyber-entity provides, and (4) the energy units that the cyber-entity

[2] www.fipa.org

Name	Functionality
Relationship management	allows cyber-entities to establish, examine, update and eliminate their relationships.
Social networking	allows cyber-entities to locate other cyber-entities through their relationships with their search criteria.
CE sensing	allows cyber-entities to locate the cyber-entities running on the local platform.
Migration	allows cyber-entities to move to another platform.
Pheromone emission	allows cyber-entities to emit their pheromones and sense pheromones emitted by other cyber-entities.
Lifecycle service	provides cyber-entities lifecycle operations.
Resource sensing	allows cyber-entities to sense the type, amount and unit cost of available resources.
Energy management	keeps track of energy level of the cyber-entities running on the local platform.

Table 1. A list of the bionet services

requires to provide its service. A GUID is a 32-digits string data made from hexadecimal representations of IP address, JVM identity hash code[3] of the singleton GUID generator, the current time in milliseconds and a random number[4]. A cyber-entity's reference is formatted as a stringfied CORBA object reference [20]. Cyber-entities can specify any other information as their optional attributes. Attributes are represented as name-value pairs based on the OMG constraint language [22]. A sample of mandatory attributes is described as follows:

```
GUID='sti3sdr98rd56fn...'
ref='IOR:daforimklcmd...'
serviceType='HTTP/1.1'
serviceCost=100.0
```

Figure 3 shows the design of the base class for cyber-entities, `CyberEntityImpl`. This class defines a set of variables and methods that are common among all the cyber-entities. Developers define their own cyber-entities by extending this class.

3.3 Bionet Services

The bionet platform currently provides eight bionet services that cyber-entities use for performing their behaviors (Table 1). We implemented the bionet services based on five functional requirements derived from the features and behaviors of cyber-entities. We describe the design of bionet services along with the requirements.

Each bionet service runs on per-platform basis. Since decentralization is a key design principle in our mind (see Section 2), we implemented all the bionet services in a decentralized manner; no centralized entities exist to control cyber-entities.

(1) Relationship management. As described earlier, cyber-entities use their relationships to represent their acquaintances, discover other cyber-entities and interact with them. Therefore, the bionet platform provides the relationship management service, which allows cyber-

[3] obtained by calling `System.identityHashCode()`

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

entities to establish, examine, update and eliminate their relationships (Table 1). As shown in Figure 3, each cyber-entity has a list of `Relationship` objects, each of which represents a relationship with another cyber-entity. A `Relationship` object contains the metadata (attributes) of a partner cyber-entity and the date when the relationship is established. Cyber-entities can put any additional information (e.g. keywords describing their partner cyber-entities) in the `properties` variable.

When a cyber-entity establishes a relationship with another one, it calls the `establishRelationship()` operation that accepts the relationship partner's GUID and/or reference as its parameters. The operation checks if the partner exists, and if it does, obtains the partner's attributes, and instantiates a `Relationship` object.

In order to establish an initial set of relationships, a cyber-entity typically searches for other cyber-entities running on the same platform by using the CE sensing service (Table 1). It may also ask its partners to introduce their partners in order to establish more relationships.

(2) Dynamic discovery. The autonomy and decentralization features of cyber-entities produce the need for a method to dynamically locate cyber-entities. Therefore, the bionet platform provides the social networking service, which allows cyber-entities to discover others with various search criteria in a decentralized manner (Table 1). The design approach of the social networking service is similar to that of peer-to-peer networking systems [23, 24, 25, 26]. Cyber-entities construct an overlay network with their relationships for routing discovery queries. A discovery process involves in four phases: *query initialization*, *query matching*, *query forwarding*, and *query hit backtracking*.

In *query initialization*, a discovery originator (i.e. a cyber-entity) begins a discovery process by generating a query through an operation of the social networking service. Each query contains its GUID to distinguish it from other queries, hops-to-live count to determine discovery termination, and search criteria that specify which cyber-entities are being searched for. Search criteria are described based on the OMG constraint language [22]. Examples of search criteria are as follows:

```
GUID=='sti3sdr98rd56fn...'
serviceType=='HTTP/1.1' and serviceCost<150.0
```

The *query matching* phase is performed after a query is initialized or a cyber-entity receives a query from another cyber-entity. The social networking service provides an evaluator object used to examine if the received query (i.e. the query's search criteria) matches a given cyber-entity. If the query matches, a query hit is returned to a discovery originator. Otherwise, the query is forwarded to other cyber-entities.

In the *query forwarding* phase, queries are moved from cyber-entity to cyber-entity through their relationships, seeking the cyber-entities that satisfy search criteria. In order to forward a query, a cyber-entity uses the `forwardQuery()` operation of the social networking service (Figure 4). This operation decrements the hops-to-live value in a received query, and if the value becomes zero, the query is discarded. It also examines whether the query has already been forwarded, using a discovery message table (Figure 4), and if it already has, the query is discarded to avoid a forwarding loop. Otherwise, the query is forwarded to the relationship partners of the cyber-entity that invoked `forwardQuery()`. When a query is forwarded, the social networking service inserts an entry regarding the query in its discovery message table (Figure 4). The entry keeps a record of the query's GUID, the cyber-entity from which the query is received, and the cyber-entity to which the query is forwarded.

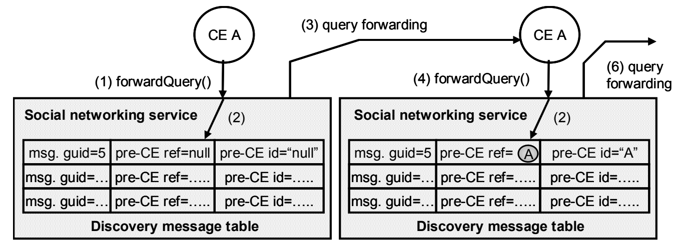

Figure 4. Query forwarding process in the social networking service

The *query hit backtracking* phase is performed when a query matches a cyber-entity. A query hit is generated and returned back to a discovery originator, following the reverse route of the forwarding path that led to the cyber-entity being returning the query hit. The back propagation path information can be obtained from a discovery message table in each social networking service (Figure 4).

In addition to the social networking service, the bionet platform provides another service, called the CE sensing service, to locate cyber-entities (Table 1). This service keeps track of the cyber-entities that exist on a local platform. This service is typically used for cyber-entities to establish their initial relationships.

(3) Migration. Since cyber-entities move around the network, the bionet platform provides the migration service, which allows them to migrate from a platform to another platform without losing the information they keep. The design approach of this service aligns to so-called *weak migration* [27], in which data state associated with an object is transferred between different network nodes.

The migration service is responsible for sending out a cyber-entity and receiving a migrating cyber-entity. It transfers a cyber-entity's class name, class definition and runtime data state to a migration service running on a destination platform. The class definition and data state are serialized at an origin platform and de-serialized on a destination by using Java serialization mechanism. The transferred class definition is loaded into a JVM on a destination platform using *bionet class loader* (see Section 3.1 and Figure 1). After the class definition is

loaded and data state of a cyber-entity is de-serialized, a destination-side migration service instantiates the cyber-entity. Then, the instantiated cyber-entity is passed to the lifecycle service (Table 1) to start executing its `run()` method (see also Figure 2).

Since cyber-entities are autonomous, they move around the network without any intervention from others. As a result, after a cyber-entity moves, the relationships (particularly, references contained in the relationships) associated with the cyber-entity become invalid. In this case, by using the social networking service, cyber-entities may try to locate the missing cyber-entity or other cyber-entities that implement the service the missing one provides.

The bionet platform provides another service for cyber-entities to locate missing cyber-entities through the pheromone emission service (Table 1). This service allows a cyber-entity to leave its pheromone (or trace) on a local platform when it migrates to another platform, so that other cyber-entities will be able to find it on a destination platform. The service keeps a record of the emitted pheromones in a certain time period. Each pheromone contains a cyber-entity's GUID and the reference pointing to a destination platform's representative. When a cyber-entity tries to locate a missing one, it accesses a pheromone emission service running on the platform where the missing one used to exist and asks the service for the pheromone of the missing one with its GUID. Then, it contacts a destination platform's representative, which is contained in the obtained pheromone, to find the current reference of the missing cyber-entity through the CE sensing service running on the destination platform.

(4) Lifecycle management. As cyber-entities are dynamically initialized, replicated or reproduced, the bionet platform provides the lifecycle service, which provides a series of lifecycle operations to them (Table 1).

The lifecycle service is used to initialize a cyber-entity when it is newly created or when it completes a migration. The service accepts a cyber-entity's instance, and creates a CE context to associate it with the cyber-entity, assigns a GUID to the cyber-entity, registers the cyber-entity to the bionet container, and starts running its `run()` method (see Figure 2).

The lifecycle service is also used to replicate a cyber-entity or reproduce a child cyber-entity from two parent cyber-entities. The service makes a deep copy of a parent cyber-entity using Java serialization mechanism. Mutation may happen on a child cyber-entity during replication and reproduction. For example, an inherited set of relationships and other properties (e.g. behavior policies) may be randomly modified. Crossover happens during reproduction to inherit relationships and other properties from two parents. The evolutionary aspect of cyber-entities is beyond the scope of this paper. Please see [9, 10] for more details about this issue.

(5) Environment sensing. Since cyber-entities need to sense their surrounding network conditions to perform their biological behaviors, the bionet platform provides a series of services for environment sensing. They allow for each cyber-entity to sense (1) its current energy level, (2) resource availability on a local platform, (3) the current traffic load on a local platform, and (4) the number of cyber-entities running on a local platform.

The current energy level of a cyber-entity is available through the energy management service (Table 1). This service keeps track of the energy level of every cyber-entity running on a local platform, and any cyber-entity can ask the service for its current energy level. The service maintains the energy table that contains pairs of cyber-entity's GUID and current energy level. A table entry associated with a cyber-entity is created by a lifecycle service when the cyber-entity is initialized.

The resource sensing service allows cyber-entities to monitor the type, amount and unit cost of resources (CPU cycles and memory space) available on a local platform. The service calculates the CPU availability by measuring the current CPU utilization. Since any Java program cannot inspect CPU utilization through the standard APIs, we built an external library implemented in C with JNI (Java Native Interface). The library determines the JVM's process ID and obtains CPU time spent executing kernel and user code on behalf of the process[5]. The resource sensing service calls this library to take CPU usage snapshots at regular intervals and obtains the current CPU utilization on percentage. Memory utilization is obtained by executing garbage collections[6] until the amounts of free memory in JVM[7] become same before and after a garbage collection. The resource sensing service can be invoked remotely as well, so that cyber-entities can sense the resource availability on a remote platform.

Cyber-entities can also sense the current traffic load and the number of cyber-entities on a local platform. As described earlier, the traffic load is available from the bionet container, and the number of local cyber-entities is available through the CE sensing service (Table 1).

4. Initial Measurement Results

This section describes some of the initial measurement results to examine the footprint, efficiency and scalability of the bionet platform.

4.1 Measurement Configuration

The measurements were conducted with two bionet platforms running on different Windows 2000 PCs (Service Pack 4), each of which hosts Java 2 SDK (version 1.4.2_01 from Sun Microsystems) with an Intel Pentium 4 processor (1.8 GHz) and 512 MB RAM. The PCs are connected through a 100Mbps Ethernet switch.

---

[5] obtained through the `getProcessTimes()` system call on Windows.
[6] through the `Runtime.gc()` method.
[7] measured by calling `Runtime.freeMemory()`.

In order to measure time duration in our experiments, we used our own timer written in C with JNI. We did not use the `currentTimeMillis()` [8] method of Java's `System` class, because its resolution is coarse. On Windows 2000, Sun SDK 1.4.2 provides 10ms resolution. The method is suitable for profiling relatively long-lasting (e.g. 100 ms and longer) operations, but it does not work well in our measurements. Our timer uses Win32 native functions; `QueryPerformanceFrequency()` and `QueryPerformanceCounter()`. The first function returns the frequency of the timing counter in cycles per second. The second function returns the current counter value (i.e. the number of CPU clock cycles) since PC's powerup. Through these native functions, our timer provides a resolution of 0.001ms.

In every experiment to measure time duration, we *warmed*[9] the JVM(s) before the experiment by executing measurement code for enough time. Since Java code is generally optimized at runtime, the first several executions of a line of code are slow as the JVM is still optimizing it. Our measurement code was optimized through JVM warming before each measurement.

CPU and memory utilizations were measured in the way described in Section 3.3 (the subsection about environment sensing).

4.1 Measurement Results

Table 2 shows the bootstrap overhead and memory footprint of each platform component. The bootstrap overhead measures the time for the bionet platform to initialize each platform component, and the bootstrap memory footprint measures the amount of memory space each platform component consumes when it is initialized.

platform component	overhead	footprint
Bionet message transport	22.98 msec	6.65 KB
Bionet container	127.06 msec	8.88 KB
Bionet class loader	9.11 msec	3.97 KB
Platform representative	82.31 msec	5.23 KB
Relationship mgt service	23.17 msec	4.48 KB
Social networking service	69.85 msec	12.03 KB
CE sensing service	56.43 msec	7.82 KB
Migration service	33.13 msec	4.88 KB
Pheromone emission service	37.79 msec	7.39 KB
Lifecycle service	91.92 msec	44.07 KB
Resource sensing service	64.36 msec	42.12 KB
Energy management service	59.02 msec	8.12 KB
Total	677.13 msec	154.64 KB

Table 2. Bootstrap overhead and memory footprint of each platform component

The measurement results show that the bootstrap overhead and memory footprint of each platform component are fairly small. The footprint of the lifecycle service is relatively large because the service creates a thread pool that contains five idle threads when it is initialized. Also, an external JNI-based library used to measure CPU utilization is dominant (approx. 37 KB) in the footprint size of the resource sensing service. The service tests loading the external library when initialized.

Table 3 shows the overhead of typical activities to install and start running a cyber-entity on a bionet platform. We used an empty subclass of `CyberEntityImpl` for this measurement. The total overhead to instantiate the cyber-entity class through the `new` operator and perform a series of activities was 2,194.59 msec. The total overhead to replicate the cyber-entity class through the lifecycle service was 2,295.44 msec. In both cases, an initialized cyber-entity contacts the local CE sensing service to locate 100 other cyber-entities running on the same platform. It also establishes relationships with the located cyber-entities. We believe these two overhead results are small and acceptable for the programming and deployment work by developers.

activity		overhead
Class loading		11.21 msec
Instantiation	created by a developer	3.73 msec
	Replicated by a parent cyber-entity	104.58 msec
Initialized through the lifecycle service		198.24 msec
Discovers 100 cyber-entities running on the same platform using the CE sensing service		723.59 msec
Establishes (initial) relationships with the discovered 100 cyber-entities using the relationship management service		1,257.82 msec
Total	created by a developer	2,194.59 msec
	through replication	2,295.44 msec

Table 3. Overhead to install and initialize a cyber-entity

Figure 5 shows the messaging roundtrip time between two cyber-entities that run on different bionet platforms. In this measurement, we deployed a single cyber-entity (sender cyber-entity) on a platform and a range of cyber-entities (from 1 to 1000 receiver cyber-entities) on the other platform. The sender randomly chose one of the remote receivers and sent an empty message to the chosen receiver. Then, the receiver sends back an empty message to the sender. Figure 5 depicts that the roundtrip time is comparable with well-known Java-based distributed object platforms (JacORB and Java IDL[10]), indicating that the bionet message transport and bionet container are implemented efficient.

Figure 5 also shows that the roundtrip time remains relatively constant as the number of receiver cyber-entities grows up to 1,000, indicating that the bionet message transport and bionet container scales well. This result owes the connection management design in the bionet message transport. It does not create a new socket for each receiver cyber-entity. Instead, a sender transmits messages to multiple receivers running on a remote platform over the same TCP connection (a single TCP connection is shared between two different platforms).

[8] returns the difference, measured in milliseconds, between the current time and midnight, January 1, 1970.

[9] It is called JVM warming to perform several passes through a line of code to allow JVM to optimize the execution of the code.

[10] java.sun.com/products/jdk/idl/

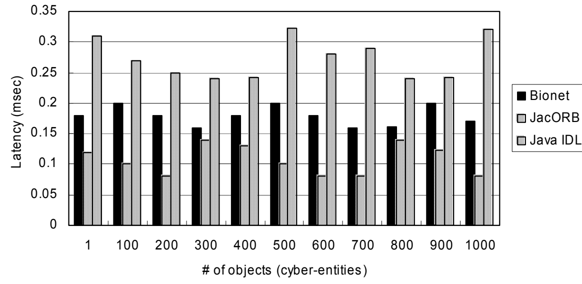

Figure 5. Messaging roundtrip latency

Figure 6 shows the throughput of the bionet platform per cyber-entity (i.e. how many interactions two cyber-entities can perform per second). The measurement configuration is the same as the previous one. As Figure 6 shows, two cyber-entities running on different platforms can send approximately 2,200 messages per second with each other. This throughput result is competitive with existing distributed object platforms, and we believe the bionet message transport and bionet container are efficient enough. Figure 6 also shows that the throughput remains mostly constant as the number of cyber-entities grows up to 1,000, indicating that the bionet platform scales well.

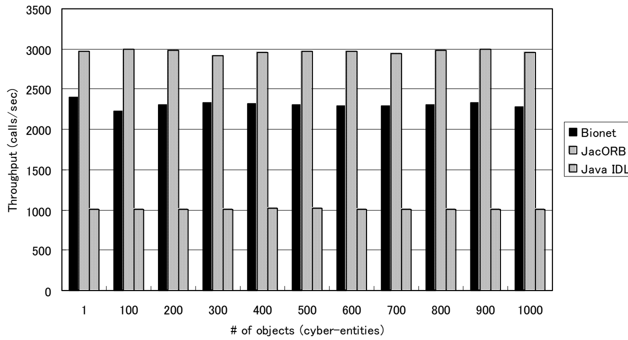

Figure 6. Throughput of message exchanges

Table 4 shows the overhead of each phase in a discovery process using the social networking service. In this measurement, two cyber-entities were deployed on different platforms, and a cyber-entity (discovery originator) established a relationship with the other one (discovery responder). The discovery responder maintained the mandatory attributes that are shown as examples of attributes in Section 3.2.

Phase in a discovery process		overhead
Relationship establishment between 2 cyber-entities		2.48 msec
Query initialization		7.23 msec
Query forwarding		29.33 msec
Query matching (on a discovery responder)	GUID matching	6.56 msec
	Complex matching	12.82 msec
Query hit backtracking		24.84 msec

Table 4. Discovery overhead

We tried two different search criteria: the first one was `GUID=='sti3sdr98rd56fn...'` as a GUID matching, and the second one was `serviceType=='HTTP/1.1' and serviceCost<150.0` as a complex matching (see Table 4). The overhead results of query initialization, query forwarding and query hit backtracking were same in different measurements using different search criteria. As shown in Table 4, the social networking service efficiently performs distributed discovery process.

Table 5 shows overhead results of the activities using the energy management service. We deployed two bionet platforms, and 100 cyber-entities on each platform. In the first activity, a cyber-entity looks up its own entry from 100 entries in the energy table of the local energy management service, in order to see its current energy level. In the second activity, upon a request from a local cyber-entity running on the same platform, an energy management service notifies a remote energy management service to increase a (remote) cyber-entity's current energy level by 100.0. Then, the local energy management service decreases the local cyber-entity's energy level by 100.0. We believe that the energy transaction cost is acceptable and it does not have any harmful effects on other platform components or cyber-entities.

Activities	Overhead
A cyber-entity asks the local energy management service for its current energy level.	18.67 msec
A cyber-entity asks the local energy mgt. service to pay 100.0 energy units to another cyber- entity running on a different platform.	46.51 msec

Table 5. Overhead for energy transactions

Figure 7 shows the overhead for a cyber-entity to migrate from a platform to another using the migration service. The migration overhead includes the transmission time over the network and the processing time at both origin and destination platforms. As the size of mobile code grows, the overhead increases linearly, instead of exponentially, indicating the migration service scales. The dominant factor in migration overhead is the cost to serialize and de-serialize a cyber-entity's data state (83.6% with a mobile code of 31KB, and 92.8% with a mobile code of 8MB).

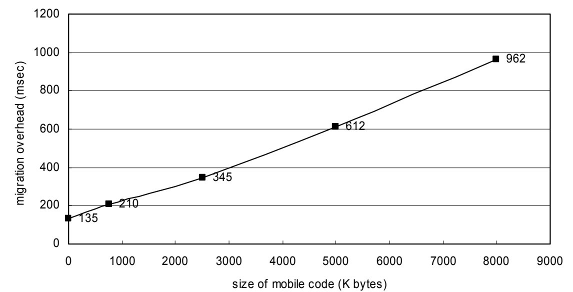

Figure 7. Migration overhead

In the next measurement, we deployed a bionet platform on a PC and multiple cyber-entities on the platform. Each cyber-entity implements a web server

function that processes the GET request message defined in the HTTP specification version 1.0. A simulated user was deployed on the same PC, and it pushed HTTP GET messages to the message queues of the cyber-entities. When receiving a message, a cyber-entity locates, reads and returns a requested file. Each cyber-entity keeps five different files whose sizes are 500B, 5KB, 50KB, 500KB and 5MB. These five types of file request are representative in Webstone [35], a well-known (de-fact standard) performance profiling tool for web servers. The request rate was 10 requests per second.

Figure 8 shows the CPU utilization of the web server cyber-entities and bionet platform. When the CPU utilization goes around 75%, the total utilization on the testbed PC reaches 100%; the other 25% is consumed by the operating system. In the case of 500B file requests, 350 cyber-entities can be executed under 75% CPU utilization. In 5M file requests, 50 cyber-entities can be executed.

A heavy line in Figure 8 shows the CPU utilization in the case that a user requests, in a single measurement run, different-sized files based on the probability shown in Table 6. This probability is defined by WebStone. In this configuration, 320 cyber-entities can work simultaneously before the CPU utilization reaches 75%. 290 cyber-entities can run under 50% CPU utilization. Also, the CPU utilization increases almost linearly as the number of cyber-entities grows. Given these results, we confirmed the bionet platform is scalable enough in terms of the number of cyber-entities.

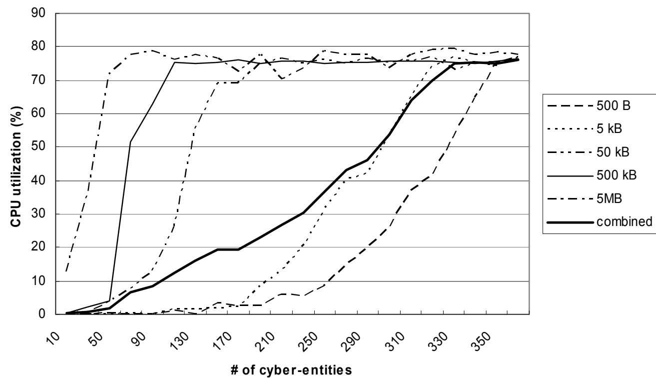

Figure 8. CPU utilization of the cyber-entities that implement web server functions

File size (bytes)	Probability (%)
500	35
5 K	50
50 K	14
500 K	0.9
5 M	0.1

Table 6. Probability to request different sized files

Please note that each cyber-entity had its own set of files in this measurement; different cyber-entities did not access a shared set of files. Also, each cyber-entity's message queue was configured to have infinite length so that it prevented message overflow.

5. Related Work

The bionet platform is similar to existing mobile agent platforms, such as Aglets[11], Mole [28], AgentSpace [5] and SOMA [29], in the sense that it implements a weak migration mechanism for agents. However, unlike them, the bionet platform emphasizes on decentralized organization of agents. Almost all the existing agent platforms assume the existence of centralized entities. Hive addresses decentralization of agents [30], but its implementation currently depends on a centralized directory (Java RMI registry). In contrast, the bionet platform allows agents (i.e. cyber-entities) to form an overlay (virtual) network among agents using their relationships and perform distributed discoveries through the relationships with the social networking service and pheromone emission service.

Pole is similar to our social networking service in the sense that it implements a decentralized agent discovery mechanism [31]. Its discovery process is performed on a *structured* peer-to-peer overlay network[12] with a distributed hash function. In the discovery mechanisms based on distributed hash functions (including several peer-to-peer systems such as Chord [25] and OceanStore [26]), it is expensive and hard to maintain their overlay routing structures in dynamic environment where peers (or agents) often join and leave the network [32]. Also, they do not allow each peer to specify multiple search criteria for each query. Unlike them, instead of relying on any distributed hash function, our social networking service is designed on a *loosely-structured* overlay network[12] among cyber-entities in order to assume dynamic network environments. It also provides a flexible discovery scheme that allows cyber-entities to specify multiple search criteria (as name-value pairs) for each query.

6. Concluding Remarks

This paper described our research effort to develop a scalable and efficient infrastructure for autonomous adaptive agents running on the Internet. We presented the design of our platform services that contribute to increase the scalability, autonomy, decentralization and flexibility of agents, and also showed that those services can be implemented scalable, efficient and lightweight through measurement results.

As future work, we plan an extended set of measurements. We evaluated scalability and efficiency of our platform services in terms of the number of cyber-entities running on platforms, but the network size is still small. We will deploy the bionet platforms and cyber-entities on larger-scale networks (e.g. PlanetLab[13]) to identify the impacts of network size on the platform

[11] http://sourceforge.net/projects/aglets/

[12] See [32] for the difference between structured and loosely-structured peer-to-peer overlay networks.

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

performance by comparing the measurement results shown in this paper with new measurement results.

Although the bionet platform provides a rich set of services, every user and application does not always require all of them. Also, every network node may not be able to deploy all the platform components due to resource constraints. Therefore, we started decomposing the bionet platform into finer-grained components and implementing a reconfiguration framework so that the platform can statically/dynamically load, unload, replace the fine-grained platform components [10]. Our goal is to customize and deploy different configurations of the bionet platform on different kind of network nodes (e.g. from cell phones to regular workstations) without breaking the platform architectural design.

Finally, we proposed several key designs in the bionet platform to the Object Management Group (OMG) as a standard reference architecture for the Super Distributed Objects specification. The specification was adopted by OMG in April, 2003 [33]. It will be finalized and officially published within 2003.

References

[1] P. Maes, "Modeling Autonomous Adaptive Agents," *Artificial Life, I (1&2)9, 1994.*

[2] S. Franklin and A. Graesser, "Is it an agent or just a program?: A Taxonomy for Autonomous Agents," *Proc. of ATAL'96*, 1996.

[3] Y. Fan and S. Gauch, "Autonomous Adaptive Agents for Information Gathering," *Proc. of AAAI '96*, 1996.

[4] B. J. Rhodes, N. Minar and J. Weaver, "Wearable Computing Meets Ubiquitous Computing: Reaping the best of both worlds," *Proc. of ISWC '99*, 1999.

[5] N.J.E. Wijngaards, B.J. Overeinder, M. van Steen, and F.M.T. Brazier, "Supporting Internet-Scale Multi-Agent Systems," *Data Knowledge Engineering (4)2-3,* 2002.

[6] S. Moss, "Critical Incident Management: An Empirically Derived Computational Model," *Artificial Societies and Social Simulation, 1*(4), 1998.

[7] P. Wurman, M. Wellman and W. Walsh, "The Michigan Internet AuctionBot: A Configurable Auction Server for Human and Software Agents," *Proc. of Agents' 98*, 1998.

[8] 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 K. Park (ed.) *The Internet as a Large-Scale Complex System*, Princeton University Press, 2002

[9] M. Wang and T. Suda, "The Bio-Networking Architecture: A Biologically Inspired Approach to the Design of Scalable, Adaptive, and Survivable/Available Network Applications," *Proc. of the 1st IEEE SAINT conference*, 2001.

[10] J. Suzuki, T. Nakano, K. Fujii, N. Ikeda and T. Suda, "Dynamic Reconfiguration of Network Applications and Middleware Systems in the Bio-Networking Architecture," *Proc. of IEEE LARTES*, 2002.

[11] J. Suzuki and T. Suda, "Adaptive Behavior Selection of Autonomous Objects in the Bio-Networking Architecture," *Proc. of AINS*, 2002.

[12] K. Kelly, *Out of Control,* Brokman Inc., 1994.

[13] T. Hong, "Performance," *Peer-to-Peer*, A. Oram (ed.), Chapter 14, Wiley, 2001.

[14] N. Minar, K. H. Kramer and P. Maes, "Cooperating Mobile Agents for Dynamic Network Routing," *Software Agents for Future Communications Systems*, 1999.

[15] R. Albert, H. Jeong and A. Barabasi, "Error and Attack Tolerance of Complex Networks," *Nature* 406.

[16] G Cabri, L. Leonardi and F Zambonelli, "Mobile-Agent Coordination Models for Internet Applications," Computer 33(2):82-89, February 2000.

[17] C. Castelfranchi, "Guarantees for Autonomy in Cognitive Agent Architecture," *Proc. of ECAI-94 Workshop on Agents Theories, Architectures, and Languages*, Springer, 1995.

[18] M. Luck and M. P. D'Inverno, "A Formal Framework for Agency and Autonomy," *Proc. of MAS"95*, 1995.

[19] T. Itao, T. Nakamura, M. Matsuo, T. Suda and T. Aoyama, "The Model and Design of Cooperative Interaction for Service Composition," *Proc. of the DICOMO*, 2001.

[20] OMG, *The CORBA Specification, version 3.0*, 2002.

[21] http://netresearch.ics.uci.edu/bionet/resources/platform/

[22] OMG, *The Trading Object Service*, 2000.

[23] M. Ripeanu, I. Foster and A. Iamnitchi, "Mapping Gnutella Network: Properties of large-scale peer-to-peer systems and implications for system design," *IEEE Internet Computing Journal, 6*(1), 2002.

[24] I. Clarke et al., O. Sandberg, B. Wiley, and T.W. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System in Designing Privacy Enhancing Technologies," *Proc. International Workshop on Design Issues in Anonymity and Unobservability*, LNCS 2009, Springer, 2001.

[25] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," *Proc. of ACM SIGCOMM 2001*, 2001.

[26] J. Kubiatowicz, et al., "OceanStore: An Architecture for Global-Scale Persistent Storage," *Proc. of the Ninth ASPLOS 2000,* 2000.

[27] A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding Code Mobility,". *IEEE Trans. on Software Engineering, 24*(5), May 1998.

[28] J. Baumann, F. Hohl, K. Rothermel and M. Straßer, "Mole: ncepts of a Mobile Agent System," *World Wide Web 1*(3), 1998.

[29] P. Bellavista, A. Corradi, C. Stefanelli, "Mobile Agent Middleware to Support Mobile Computing," *IEEE Computer*, *34*(3), pages 73-81, March 2001.

[30] N. Minar, M. Gray, O. Roup, R. Krikorian and P. Maes, "Hive: Distributed Agents for Networking Things," *Proc. of ASA99*, 1999.

[31] B. Overeinder, E. Osthumus and F. Brazier, "Integrating Peer-to-Peer Networking and Computing in AgentSpace Framework," Proc. of the 2nd IEEE International Conference on Peer-to-Peer Computing, 2002.

[32] S. Androutsellis-Theotokis, "A Survey of Peer-to-Peer File Sharing Technologies," white paper, Athens University of Economics and Business, 2002.

[33] S. Sameshima, S. Steglich, J. Suzuki and T. Suda, "Platform Independent Model (PIM) and Platform Specific Model (PSM) for Super Distributed Objects," Object Management Group, draft adopted specification, OMG document number: dtc/03-04-02, April 2003.

[34] I. Singh, B. Stearns and M. Johnson, *Designing Enterprise Applications with the J2EE Platform*, second edition, Addison-Wesley, 2002.

[35] G. Trent and M Sake, "WebStone: The First Generation in HTTP Server Benchmarking," Mindcraft, Inc., 1995.

Appendix