Protected and Secure Mobile Object Computing in PLANET

Kazuhiko Kato, Kunihiko Toumura, Katsuya Matsubara, Susumu Aikawa, Jun Yoshida[1] *and Kenji Kono, Kenjiro Taura, Tatsurou Sekiguchi*[2]

1 {kato | ktoumura | matsu | aikawa | yoshida}@softlab.is.tsukuba.ac.jp, Institute of Information Sciences and Electronics, University of Tsukuba, Tsukuba, Japan

2 {kono | tau | cocoa}@is.s.u-tokyo.ac.jp, Deptartment of Information Sciences, Faculty of Science, University of Tokyo, Tokyo, Japan

Worldwide networks such as the Internet are becoming very popular, so distributed computing environments for such networks are in high demand. We think the design of such an environment should be based on a mobile-object computing model and are therefore designing a mobile-object system called Planet. One of the most significant issues in designing mobile object systems for world-wide networks is to provide the control needed to assure the protection and security of mobile objects and of computing resources. In this paper we describe our approach to this issue.

1 Introduction

Nowadays worldwide networks such as the Internet are becoming very popular in general society. Much academic and industrial research has been devoted to developing distributed programming and computing environments (see [6] for a survey), but in most cases this research has assumed explicitly or implicitly that the scale of the underlying networks is no greater than that of local area networks. The characteristics of worldwide networks are very different from those of local area networks. One difference is that communication latency is significant. Even if messages are transferred at the speed of light, round-trip communication between two computer sites located on opposite sides of the earth can be completed at most 7.5 times per second. In the actual environment, the communication latency tends to be several or several dozen of times larger because of protocol and packet-switching overheads. Another difference is that a very large number of users have to share a limited communication bandwidth. The number of computing sites is of course much larger in a worldwide network, and many of the sites are not known *a priori*. Lastly, the users and service-providers in the worldwide network might not be acquainted with each other. We are designing a distributed computing system for a worldwide network. The system is called Planet, and this paper describes the basic approaches and models of Planet. It also discusses the mechanisms providing protected and secure mobile-object computing in Planet.

2 Overviews

2.1 Basic Approaches

The basic approaches of Planet are summarized by the following aspects.

(a) Paradigm-shift from message passing to object passing.

Most practical distributed computing systems previously developed were based on the message passing paradigm (including RPC) or used the message passing mechanism as the key mechanism for implementation [6][45]. In worldwide networks, as long as we cling to the message passing paradigm, we cannot break the above-mentioned 7.5 (RPC/sec) limitation. To break through the limitation imposed by communication latency and to use the restricted communication bandwidth efficiently, Planet adopts the paradigm of *object passing*, namely *mobile objects*, in its basic system design. Since object-passing systems can be considered a progression from message-passing systems, this paradigm-shift is a natural evolution.

(b) Giving persistency as well as mobility to objects.

A worldwide distributed computing environments should give persistency as well as mobility to objects. This statement is justified by reconsidering currently useful applications of worldwide networks: Email, news, WWW, etc. In a worldwide network environment, it might be unreasonable to assume that the receiver process of the mobile object is awakened whenever a sender tries to send an object and that the sender can wait until the receiver becomes available. Mobile objects should therefore have their own lifetime, independent of the lifetimes of the computing processes. This characteristic of objects is referred to as *persistency.* Fortunately, adding the persistent nature to mobile objects is not a difficult task because one of the key functions of a mobile-object system is to separate an object from the address space on which the object exists and to reload it into another address space at a different computer site; this is quite similar to the implementation of persistent objects [61][43][50].

(c) Native mobile objects.

With respect to the program execution scheme, many currently proposed mobile object systems adopt a byte-code interpreter approach. There are several compelling reasons for adopting this approach. First, it is the most straightforward way to implement mobile objects. Secondly, the approach is undoubtedly suited to deal with hardware (and even operating system) heterogeneity. Lastly, the approach is suited to dealing with security issues. For several years our research group has worked on developing fundamental techniques that could be used to achieve mobile-object execution with *native* program code ([44][50][51]) that would be applicable even in heterogeneous-architecture environments. Planet takes the approach of creating a novel execution mechanism in which mobile objects are executed with native machine code, while security issues are still not sacrificed.

(d) Programming language neutrality.

Most currently proposed mobile-object systems have their own brand-new languages, and users cannot take advantage of mobile-object computing without using these new languages. We call this approach the *language-first* approach. To make our mobile-object system applicable to many programming languages, we have adopted a *language-neutral* approach, in which the protocols between language processors and the Planet mobile object system are clearly defined so that almost any programming language processors can be ported to the Planet system.

(e) Orthogonal protection domain.

The paradigm shift from message-passing to object-passing enables objects to move all over the place, but we must be able to protect both the computing resources and the objects themselves from illegitimate accesses by mobile objects, whether intentional or accidental. In our terminology, method execution of an object is performed in a *protection domain*, which is a unit to give controlled capabilities to access the computing resources. Direct inter-protection-domain accesses are inhibited by the system. The protection domain concept should be *orthogonal* to the encapsulation concept in the object-oriented programming; that is, the same program code can be used whether or not the communicating object is in the same protection domain. This contribute not only to software productivity but also to efficient computing: when objects in the same protection domain interact with each other, the system can omit protection control and permit direct object accesses among them.

2.2 The Planet Mobile Object Model

The system model for Planet (Fig. 1) is described using five basic abstractions: *(mobile) object*, *activity*, *place*, *protection domain*, and *distributed shared repository.*

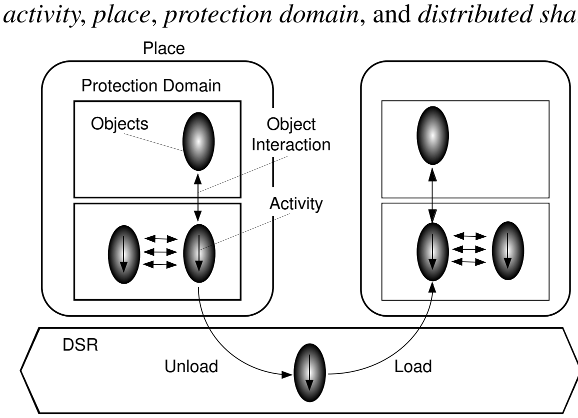

Fig. 1 Overview of Planet.

An *object* is an informational entity that encapsulates both data and programs to manipulate the data. We refer to the programs encapsulated in an object as *methods*. An *activity* is a computational entity that encapsulates the current state of computation. Activity is similar to the conventional “thread of control” concept except that activity can move across various boundaries, such as address-space boundaries and machine-architecture boundaries, and can be made persistent. In the Planet system, objects and activities both are mobile. When an activity is moved, the associated objects, which are referenced from the current state of computation, are also moved[1]. Thus, whenever an object is computed, an activity must be associated with it so the two can be considered to constitute an integrated (active) object. In the following, we use the term “objects” as a generic term to refer both to objects alone and to objects associated to an activity.

Place is an abstraction for computational resources, with which activities can perform their computation on the related objects[2]. A very simplified example of a place would be a computer host with one or more CPUs and memories. Another simplified example of a place is a LAN that has several computer hosts. A place, however, does not necessarily have persistent storage, such as magnetic disc devices, to store objects, since persistency is provided by the distributed shared repository.

The *distributed shared repository* (DSR for short) is an abstraction for the world-wide persistent object store, and it plays the central role in our object-passing mechanism. The DSR is a virtual medium spread over the world. Objects put on the DSR become accessible from places all over the world and are guaranteed to persist until they are explicitly removed. In Planet, all the inter-place object passing is performed through the DSR. When we want to send an object on a place to another place, we give a logical (location-independent) name to the object and *unload* the object from the place to the DSR. We refer to the name as the *object name*, and an object name uniquely identifies an object in the DSR. The DSR is implemented by multiple servers cooperating, and the primary copy of an object resides in one of those DSR servers. The DSR provides the name-resolution functionality to find the server that stores the object for a given object name [44]. The name space of the DSR can be seen from any place, and objects in the DSR can be *loaded* into any places provided that the places have capabilities to load the objects.

The object passing via DSR differs much from the object passing of other mobile-object systems like Telescript [91] and Obliq [13]. In these systems, object-passing is synchronous; both sender and receivers have to be awakened. In Planet, on the other hand, object-passing is asynchronous; the sender and the receiver can communicate

1. In general, which objects are associated with an activity, therefore should be moved together must be defined by each programming language. For example, associating every object in a process with a single activity effectively implements process migration, while associating a single object to an activity effectively implements so-called “active agents.”
2. This terminology of “place” is slightly different from that of Telescript, where a “place” is itself an object in which mobile objects “meet” with each other.

objects independently of the other's lifetime. This style of object passing is similar to that in Secure Object Spaces [87].

Since a place might be visited by a number of inherently enigmatic activities, a place must have a protection mechanism. The protection mechanism should provide at least two functionalities: the computational resources of a place must not be affected by accidental or intentional violations of access rights, and the objects and activities in a place must be protected against illegal accesses from each other. A *protection domain* is an abstraction for the controlled object accesses. When an object is loaded into a place, at least one protection domain has to be specified; then if no access right violation is determined by the system, the object is *attached* to the protection domain. Any number of objects can be attached to a protection domain provided that enough computational resources are available for that protection domain. The system grants access rights on the protection-domain basis, and the access rights granted to a protection domain are shared by all the objects within it. The protection level is uniform within a protection domain, so object interaction within a single protection domain does not require domain-switching and thus can be executed efficiently. Sometimes we are required to give different access rights to certain objects in a protection domain. For this purpose, Planet provides a functionality to attach an object to several protection domains simultaneously, maintaining the coherency of the attached object with the simple one-copy semantics.

3 Mechanisms for Protected and Secure Mobile Object Computing

Mobile-object systems permit objects (including program codes) to be communicated between sites, so they require more sophisticated protection and security mechanisms than message-passing-based distributed systems require. Here, by "protection" we mean to control accesses to objects and computing resources within a computer system; by "security" we mean to control accesses not only within the computer system but also within the environment in which the system operates. In this section we describe the protection and security mechanisms of the Planet mobile-object system.

3.1 Protection Mechanism

Previously proposed protection mechanisms for mobile objects have used two approaches: the *interpreter* approach and the *code translation* approach. The *interpreter* approach is probably the one mostly widely used, and Java [33], Telescript [91], and Obliq [14] are typical examples. In the *code translation approach*, proposed by Wahbe et al [89], mobile objects can be represented in native executable code. When a host machine receives code, the code is modified so as not to violate the protection barrier and is then executed directly by hardware.

Planet adopts an approach quite different from these. Two key techniques are used in its protection domain mechanism. Firstly, to implement mobile object computation efficiently under protection-domain control, we use standard virtual-memory management unit (MMU) hardware that is already available in virtually all modern computers, including personal computers. Owing to the MMU hardware, each virtual address space is completely separated, and the hardware can strictly inhibit object accesses beyond address-space barriers. Secondly, each protection domain has a distinct virtual address space, yet any number of objects (including activities) are attached and detached dynamically and iteratively by using the technique described in Reference [44]. Furthermore, pointers (i.e., virtual addresses) can be passed between protection domains in the same place by using the technique described in Reference [50] and [51], as if virtual address space barriers did not exist. Interestingly, the technique does not assume that virtual address spaces are shared nor that hardware architectures are homogeneous.

As described in Section 2, Planet incorporates the protection domain into the object concept orthogonally. To achieve the orthogonality we require the following three aspects to be unchanged for programmers regardless of protection domain setting at run time: *object identifiers*, *instance variable accesses*, and *method invocations*.

To keep object identifiers unchanged, we permit the programmers to use an address reference (a pointer) to identify an object within a place. When the pointer is passed to other protection domains, the system automatically detects the passing and applies the *pointer swizzling* technique integrated with virtual memory management; the system properly transforms pointer representations between universal and address-space-specific object-identifier formats.

To keep instance variable accesses unchanged, the system automatically caches the remote virtual address pages in the local address space. Every first access to the pages is detected by the memory-protection function of the MMU hardware, then the validity of the access is tested. If the access is valid, the page is cached in the local address space and successive accesses to the page are performed the same way—and with the same operation and access costs—that local page accesses are performed. If the access is invalid, the system raises an exception of protection domain violation. Since the mechanism is a kind of caching, we require a proper cache coherency protocol as described in Reference [50] and [51].

To keep method invocations unchanged, we use stub-based RPC techniques. If the target object for method invocation resides in the same address space, the stub simply makes a procedure call to the method in the target local object. If the target object resides in a different address space, the stub performs an RPC. The systematic stub-generation method presented in Reference [70] makes it possible to pass the arguments of *any* statically typed data in the RPC.

3.2 Security Mechanism

By combining the protection mechanism described in Section 3.1 and the well-known cryptographic techniques, we designed a secure mobile-object computing mechanism based on the *wrap-unwrap* model, which in turn is based on an analogy borrowed from the parcel post system. Interestingly, the security system can be implemented on top of the basic object-mobility functionality of Planet, so it might be considered the first "application" of Planet. The *wrap* operation encrypts an object in a place by using some encryption algorithm, attaches a unique object name to the object, then unloads the encrypted object from the protection domain to the DSR. The *unwrap* operation performs the inverse sequence of steps: that is, it loads the encrypted object from the DSR into the protection domain, and decrypts the object. The wrap and unwrap operations and the object names are encapsulated in objects called *wrappers* and *unwrappers*. A wrapper contains the methods implementing the wrap operation, an encryption key, and an object name. An unwrapper contains the methods implementing the unwrap operation, decryption key, and an object name. Since wrappers and unwrappers are objects, both can themselves be wrapped and unwrapped. Wrappers and unwrappers play the following three roles at least.

(a) Capabilities.

Wrappers and unwrappers can be used as capabilities for mobile objects. When an object O_1 in a place must first interact with an object O_2 in the DSR, O_1 must interact with the unwrapper U_{O2} that can unwrap O_2. For this, the protection domain on which O_1 is attached must have acquired the access right to interact with U_{O2}. Delegation of access capabilities can be simply represented by passing wrappers (to send) and unwrappers (to receive).

(b) References to persistent mobile objects.

In Planet, the system-support reference to an object is assigned to the object when the object is loaded into a place, and the reference is effective only in the place and only until the object is unloaded. Since an unwrapper identifies a particular object in the DSR, the programmer can use an unwrapper as a substitute for persistent reference to an object.

(c) Encryption and decryption modules.

Details of encrypting and decrypting functionalities are encapsulated in wrappers and unwrappers. Thus we might have a different encryption algorithm for each object. In other situations, such as limited internal uses, we might omit the encryption/decryption operations. Or we might create a special unwrapper that decrypts an object partially to permit restricted access to the object in a secure way.

Fig. 2 illustrates how a client place can send an object to a server place in a secure way. In the following, the step numbers are consistent with those in Fig. 2. (In the example, the object communications shown by dotted arrows in Steps (1)–(4) require object passing via the DSR, but to simplify the explanation we omit the description.) (1) The Server Object S requests the Authenticator Object A to register its service in the service directory in A. (2) A registers the service and generates a pair consisting of

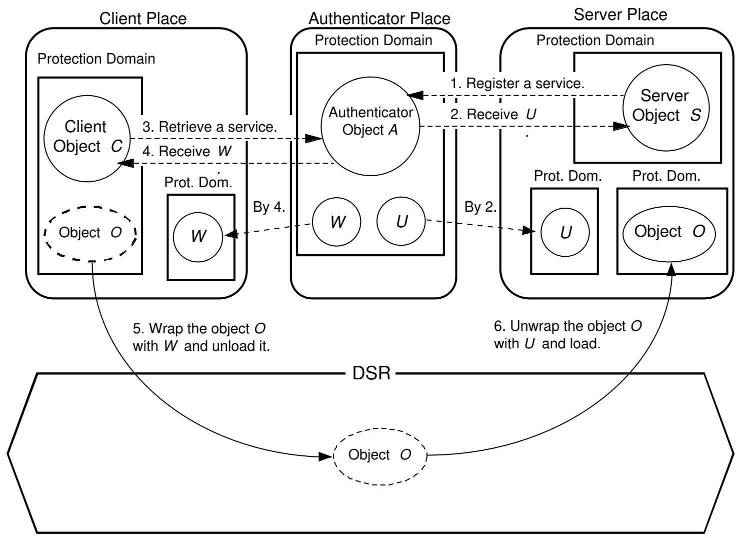

Fig. 2 Wrap-unwrap model for secure mobile object computing.

a wrapper *W* and an unwrapper *U*. *A* passes *U* to *S*, and *S* waits until an object *O* that can be unwrapped with *U* appears in the DSR. (This synchronization operation could be encapsulated in *U*.) (3) The Client Object *C* makes a request for authentication of itself and for retrieving the desired service from the *A*'s service directory. (4) *A* passes *W* to *C* as the reply to Step 3. (5) *C* puts necessary information into an object *O*, then uses *W* to wrap *O*. (6) *S* unwraps *O* with *U*, and receives from *U* the object reference of *O* valid in the Server Place.

4 Conclusion

This paper describes the secure and protected computing mechanism of the Planet mobile-object system for worldwide networks. The system gives persistency as well as mobility to objects, permits native-code execution of mobile objects without sacrificing security, provides programming language neutrality, and supports orthogonal protection domains. A prototype system of Planet with limited functionalities is already workable on a LAN environment, and we are working on moving the system to the Internet environment.

Acknowledgments

The work is partially supported by the Parallel and Distributed Consortium (PDC) in Japan and Electricity and Communication Frontier Research Funds from the Ministry of Posts and Telecommunications in Japan. We would like to thank Akinori Yonezawa and all the members of the Parallel and Distributed Language Environment Category of PDC for their discussions and encouragement on this work.