An Approach to Mobile Software Robots for the WWW

Kazuhiko Kato Yuuichi Someya Katsuya Matsubara
Kunihiko Toumura Hirotake Abe

Technical Report ISE-TR-98-154
Institute of Information Sciences and Electronics
University of Tsukuba

June 4, 1998

An Approach to Mobile Software Robots for the WWW

Kazuhiko Kato[†‡] Yuuichi Someya[†] Katsuya Matsubara[†]
Kunihiko Toumura[†] Hirotake Abe[†]

† Institute of Information Sciences and Electronics
University of Tsukuba
Tenoudai 1-1-1, Tsukuba, Ibaraki 305-8573, Japan
‡ Japan Science and Technology Corporation Email: kato@is.tsukuba.ac.jp
URL http://www.softlab.is.tsukuba.ac.jp/~kato/

Abstract

The paper describes a framework to develop mobile software robots in the Web environment through using the PLANET mobile object system we developed. Among the many recent proposals of mobile object systems, the system is characterized by language-neutral layered architecture, the native code execution of mobile objects, and asynchronous object passing. We propose an approach to implement mobile Web robots by fully utilizing the characteristics. We verify and discuss its effectiveness based on experiments conducted in the Internet environment.

Keywords: WWW, Web, mobile object, mobile agent, search robot, Internet, intranet.

1 Introduction

The recent widespread use of the Internet and the Web has had a significant effect on the whole of human culture. People who like to provide information can easily publish their information by just putting it on the Web server, and the users can easily access the information by requesting it to the servers through the use of comfortable Web browsers. The procedures to provide and access information on the Web are extremely easy, but, from the viewpoint of network resource management, the Web is not an efficient system since it is an *uncontrolled* system[1], although the freedom has contributed to the evolution of the Web as well as other services on the Internet.

If all the users of the Internet and WWW were human, their uncontrollability might not be so serious with the exception of ethical issues. In reality, however, the users are not limited to human beings. Typical instances of non-human users are WWW search robots. A Web robot is a computer program that traverses the Web's hypertext structure, recurrently retrieving all documents that are referenced [17]. Web robots can request data access to servers without the constraints of biological

[1]The Web is an uncontrolled system in the sense that nobody can control the Web.

properties, so Web robots have the potential to exhaust the network bandwidth of the Internet and the computing power of Web servers.

Software robots (sometimes called "agents") play a very important role in the Web environment. One of the most important tasks of these robots is *resource discovery.* A lot of useful data is stored on Web servers throughout the world, but to access it the user has to specify the URL (universal resource locator) of the information. To obtain these URL's for information, indexing functionality are necessary. As a method of automatically building the indices, software robot technology is used. Example systems are the WWW Worm [21], WebCrawler [23], Lycos [20], Harvest [3], AltaVista [24] and WISE [27]. Furthermore, software robot technology is used on the Web to manage server and client machines, to analyze statistics, to maintain the consistency among hypertext documents, to mirror data, etc.[17, 6].

Software robots in the Web are sometimes called *crawlers*, *spiders* or *wanderers.* As Koster discussed [17], these names, while perhaps more appealing, may be misleading, as the term "spider" and "wanderer" give the false impression that the robot itself moves, and the term "worm" might imply that the robot multiplies itself, like the infamous Internet worm [17]. In the engineering field a robot is assumed to be a mobile machine which is programmed to automatically perform a number of tasks instead of humans, but so far in the Web, "robots" are automatic but *not* mobile.

(* The next column *)

We claim that mobile software robot technology is a key technology to make the Web environment controllable without losing the virtue of freedom. This claim can be illustrated for a case of resource discovery on the Web as follows. Conventionally almost all Web indexing robots are nonmobile and they connect with target Web sites directly from robot base sites through using the application layer protocol, HTTP. A robot connects with a Web server, requests an HTML document, receives it, analyzes it, makes indices from it, extracts URLs that it should visit, and visits Web sites according to the extracted URLs. If we could use a *mobile* Web robot, the robot would move to a target Web site first. At the site, the robot would obtain and analyze HTML files, prepare indices, and extract URLs. The process would be repeated until the extracted URLs references files at the site were exhausted. Then the mobile robot would move to another Web site to obtain more Web data. The robot would continue to move until some specified predicate was satisfied. In the middle of moving, the obtained data would be sent back to the robot base at appropriate times.

The mobile robot is advantageous at least in the following respects. First, the source HTML files are not necessarily transferred via the Internet since in some cases it is sufficient to transfer only portions of HTML files or the results of some computing. Second, the HTTP server can schedule the time to process the robot's request by scheduling the execution of the visiting mobile robot.

This paper describes a framework to develop mobile software robots in the Web environment by using the PLANET mobile object system we developed. The most notable feature of PLANET is that it was designed as middleware; that is, PLANET is in the middle layer between operating systems and programming language systems and is independent of both, though some porting efforts are required to run on a typical operating and programming system [14, 12, 19]. Another notable feature of PLANET is that it implements mobile objects by the direct execution of native codes, not by interpreting bytecodes. These two features distinguish PLANET from other mobile object systems, since most other mobile object systems such as Emerald [10], Telescript [26], Aglets [18], Voyager [8], Obliq [4] and Messengers [2] take a language-centric and bytecode interpretation approach.

The rest of the paper is organized as follows. Section 2 describes the logical and Section 3 describes the physical structure of the PLANET mobile object system. Section 4 explains how mobile Web robots can be implemented with PLANET and discusses emerging issues. Section 5 discusses the results of doing experiments on mobile Web robots. Section 6 is the conclusion.

2 Logical Structure of the PLANET Mobile Object System

This section describes the background and basic concepts of the PLANET mobile object system.

2.1 Basic Approaches

Currently much research has been expended on creating Web robots [6] and creating mobile objects [25] and the advantages expected by combining the two seems to be obvious as discussed in Section 1. But why has not the combination become popular so far? The reasons are as follows.

- *Administration issue.* The Web is composed of a large collection of independent and autonomous computer sites and it constitutes an *open* distributed system, but most mobile object systems previously designed here assumed that objects are passed within a *closed* environment. By "a closed environment" we mean that the whole system is managed by an administrating person or group (here, simply called an administrator) and is operated under a single administration policy, and that all the services, service providers, and service clients (end-users) are well known to the administrator. Assuming a closed environment simplifies many technical issues. Also, the system and the users can assume that all the users and mobile objects are malice-free, so the mechanisms and policies of protection do not need to be so sophisticated.

- *Security issue.* Needless to mention the infamous Internet worms, it is very dangerous to permit software robots to operate in a uncontrolled and open environment such as the Internet. Thus, the mobile object system used in the Internet environment must have sophisticated mechanisms and policies for protection and security.

- *Performance issue.* The performance of execution is very important in practical Web robots since it must access as many Web servers as possible within a limited time. Most other mobile object systems are implemented based on the interpreter-based language processing technique, since it is well suited to mobilizing objects, to controlling protection and security, and to dealing with hardware and platform heterogeneity. To offset these advantages, the interpreter-based systems are inferior in execution performance. If a mobile robot runs slower on the Web server, the advantage of mobility may be canceled out.

The PLANET mobile object system has been designed to resolve these issues. The basic approaches of the PLANET can be summarized as follows.

- *Layered architecture.* PLANET has been designed to be middleware located between the operating system layer and the programming language layer. This contrasts with the language-centric approach that has been taken by most other mobile object systems. In these systems, the supported language, which is often brand-new, is fixed and the user cannot choose his favorite. PLANET is designed to be language-neutral and virtually any programming language can be selected so that PLANET can be used as a runtime system.
- *Native mobile objects.* PLANET premises that the execution of the mobile code is performed in the form of native codes. PLANET protects the system resources by using a virtual memory mechanism that is available for almost all modern operating systems such as Solaris, FreeBSD, Linux, and WindowsNT. PLANET regards a virtual address space as a *protection domain.* Any number of mobile objects can be loaded in one protection domain, and the objects loaded there can directly access each other since they are in the same domain. All attempts to operate computer resources located outside a virtual address space must be done by issuing system calls from inside the virtual address space. PLANET captures the issues related to system calls with the help of the functionality of operating systems, it validates these with respect to the policy settled beforehand, and it only permits the validated issues to continue.
- *Asynchronous object passing.* If a synchronous object passing style has to be used, the sender side must first probe the current state of the destination side whenever the sender wants to pass an object to the destination. This makes programming complex and inefficient. This can be understood by considering a situation where an Email can only be sent when the destination side is ready to receive it. In a practical Email system, we can send Email anytime, and the mail spooling system essentially implements asynchronous mail (object) passing. PLANET has adopted asynchronous object passing as the basic object passing scheme.

2.2 The System Model

The system model for PLANET can be described using five basic abstractions: *(mobile) object*, *place*, *protection domain*, *distributed shared repository*, and *object port* (Fig. 1).

Objects. An *object* is an entity that may encapsulate data, the program, and computational state (so-called thread). As illustrated in Fig. 2, objects are classified into four types according to the data segments included in the objects. A basic type segment includes data for basic data types such as integers, floating point numbers, strings, bitmap image data, etc. A structured data segment includes references (pointers) to data within the same segment. A program code segment includes codes that manipulate data in the objects. A computational state segment includes all the

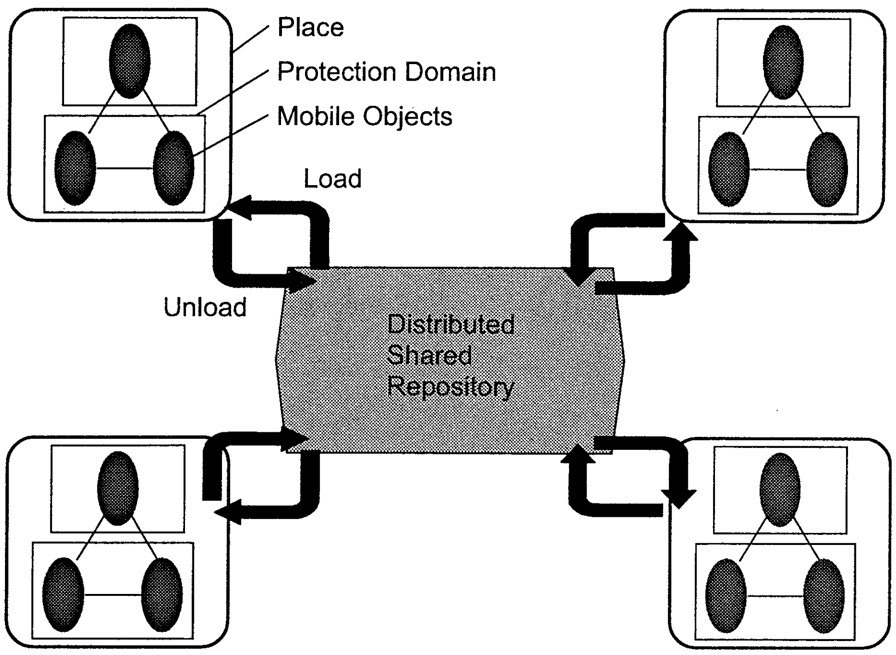


Figure 1: Logical overview of PLANET.

information required to maintain the state of computation, i.e., a program counter, CPU registers and other CPU-state materials, and a stack.

Places. *Place* is an abstraction for computational resources, through which objects do their computation. 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 as is described in the following.

Protection Domains. Since a place may be visited by a number of inherently enigmatic or malicious objects, it must have a protection mechanism. The protection mechanism should have 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 access by each other. A *protection domain* is an abstraction for 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 it. The system grants access rights on a protection-domain basis, and the access rights granted to a protection domain are shared by all the objects within it. The level of protection is uniform within a protection domain, so object interaction within a single protection domain does not require domain-switching and thus it can be efficiently executed. Sometimes we are required to give different access rights to certain objects in a protection domain.

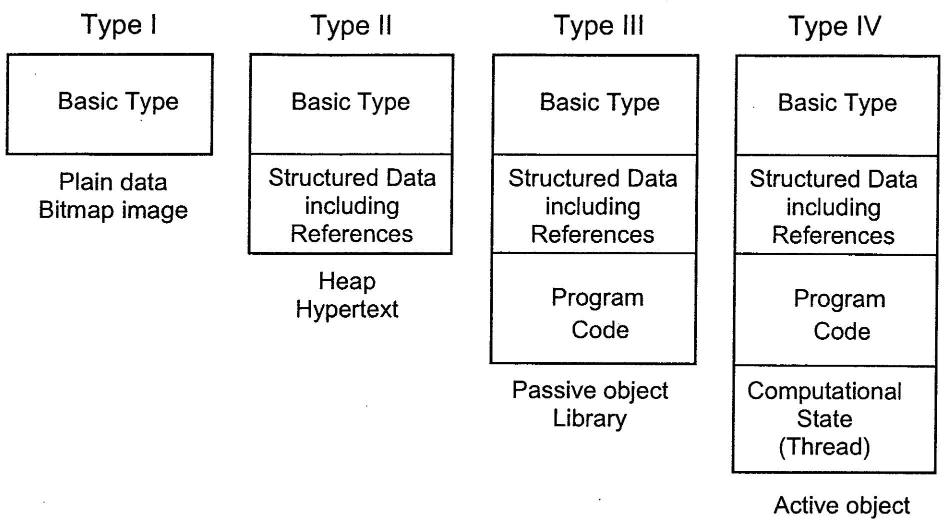


Figure 2: Object types.

For this purpose, PLANET has a functionality that allows an object to be attached to several protection domains simultaneously, maintaining the coherency of the attached object with simple one-copy semantics.

Distributed Shared Repository. The *distributed shared repository* (DSR for short) [13] is an abstraction for the worldwide persistent object store, and it plays the central role in our object-passing mechanism. The DSR can be considered as a virtual medium spread over the world. Objects put on the DSR become accessible from places all over the world and they are guaranteed to persist until they are explicitly removed.

Object passing via DSR differs greatly from the object passing of other mobile-object systems such as Emerald [10], Telescript [26] and Obliq [4]. In these systems, object-passing is synchronous in the sense that both sender and receivers have to be aware that object passing is taking place, thus the success of object sending depends on the state of the receiver. In PLANET, on the other hand, object-passing is asynchronous; the sender and the receiver can send and receive objects independently of the other's state or lifetime.

Object Ports. An *object port* is a named and access-controlled queue for objects on DSR (Fig. 3). When a sender (object) unloads an object to DSR, the sender must specify the object port into which the unloaded object is to be placed and have a write-access right for the object port. Similarly, when a receiver (object) loads an object from DSR, the receiver must specify the object port from which the loaded object is to be obtained and have a read-access right for the object port. The name given to an object port is location-independent and it must be unique. It is the PLANET system's responsibility to find the physical location of an object port for the specified name.

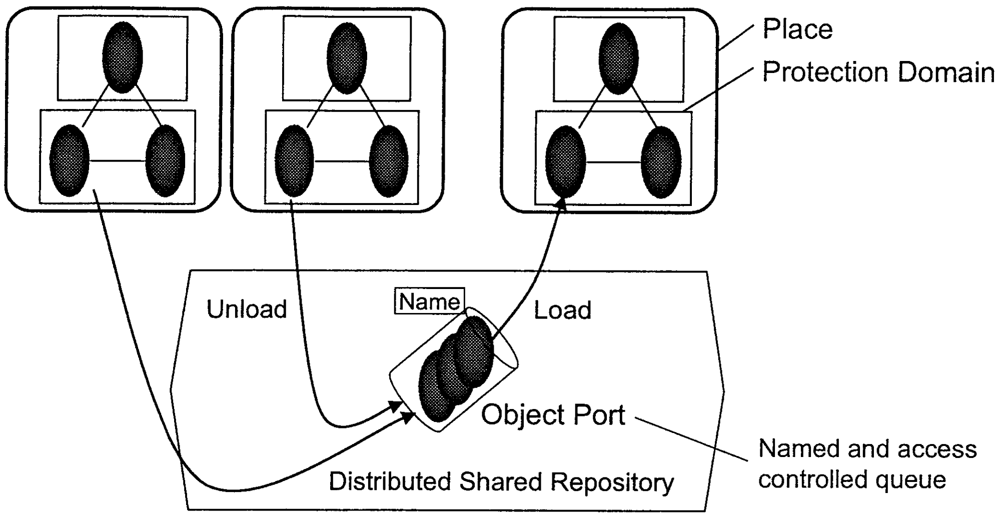

Place
Protection Domain
Unload
Name
Load
Object Port
Named and access controlled queue
Distributed Shared Repository

Figure 3: Object passing through an (persistent) object port in DSR.

3 Physical Structure of the PLANET Mobile Object System

The logical structure of PLANET is fairly simple, but the physical structure of our current implementation is rather sophisticated. The current implementation runs on SPARC workstations and the Solaris 2.5.x operating system. It is written in 40,000 lines of C programs and two hundreds lines of the SPARC assembly programs. The rest of the section briefly explains the essence of the implementation, which is necessary in order to understand the sections which follow.

3.1 Remote Memory-mapping Architecture

As described in Section 2, all the object passing is done through DSR, thus it is centric to the implementation of PLANET and its efficient implementation affects the overall performance of the system. After careful consideration, we reached the conclusion that the adoption of a *remote memory-mapping* mechanism would be the best solution to implementing DSR and the object passing mechanism done via it. The reasons for this are as follows. First, remote memory-mapping is a technique to implement persistent object in a network environment, so it is well suited to the concept of DSR. Second, with the remote memory-mapping mechanism, the system can only send the pages referenced by the clients, and only the modified pages can be returned to the DSR server. Thus the amount of transferred data can be minimized. Third, the remote memory-mapping mechanism enables implicit data transfer; that is, data transfer can be done in the middle of page fault handling, thus the programmer does not have to describe explicit data transfer. Empirically this dramatically reduces the number of program codes.

Figure 4 shows a typical structure for PLANET. A place consists of one or more client sites and a client site has protection domains, an object cache, and a name cache. A protection domain is implemented by a virtual address space. The object cache is used by the remote memory-mapping mechanism as will be described later. The name cache is used by the system to efficiently map the

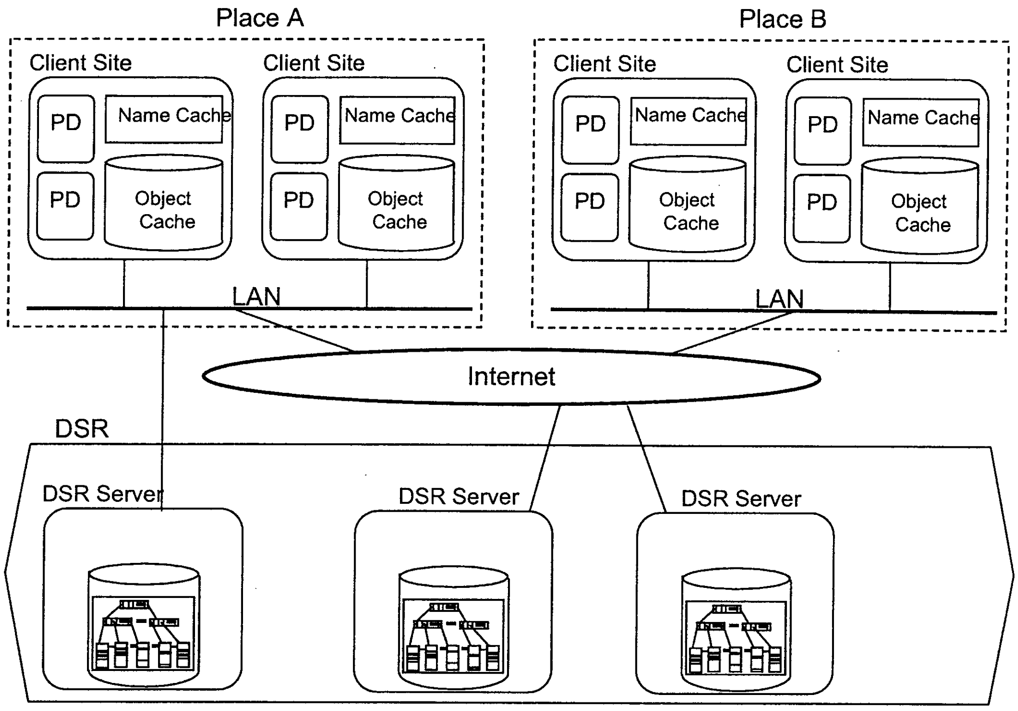


Figure 4: Physical system structure.

logical names of object ports to the corresponding physical names.

DSR is logically one space, but it is implemented by multiple servers. In Fig. 4, one DSR server is connected to the LAN of Place A and the other two DSR servers are connected to the Internet. Since the name space for DSR is unique and location-independent, the connection topology of the client sites and DSR servers do not affect the name specifications.

In PLANET, objects are loaded to protection domains and unloaded to object ports in DSR through the use of a *remote memory-mapping mechanism.* The mechanism has several advantages in the implementation of mobile objects. First, communication between a place client and a DSR server is implicit; once an object stored on a DSR server site has been specified to be loaded to a virtual address space that implements a protection domain at a place client site, each portion of the object is transferred when the portion is accessed by the CPU on the virtual address space. The transfer is done automatically by PLANET's runtime system, and it is not necessary for programmers to control the transfer. Second, network communication is minimized, since the portions transferred to a client site are locally cached, and subsequent access to the portions does not require data communication. Furthermore, only the modified portion is written back to the DSR server. Third, the same representation of objects can be used both when an object is loaded to a protection domain and unloaded to DSR. Finally, the double buffering problem between a user address space and the kernel space is avoided as a result of the local memory-mapping mechanism.

Figure 5 shows our implementation of the remote memory-mapping mechanism. The numbers in the figure indicate the sequence for the process to resolve the first access to a portion of a loaded object. At the place client site, a first reference (1) to a page that has not been locally cached

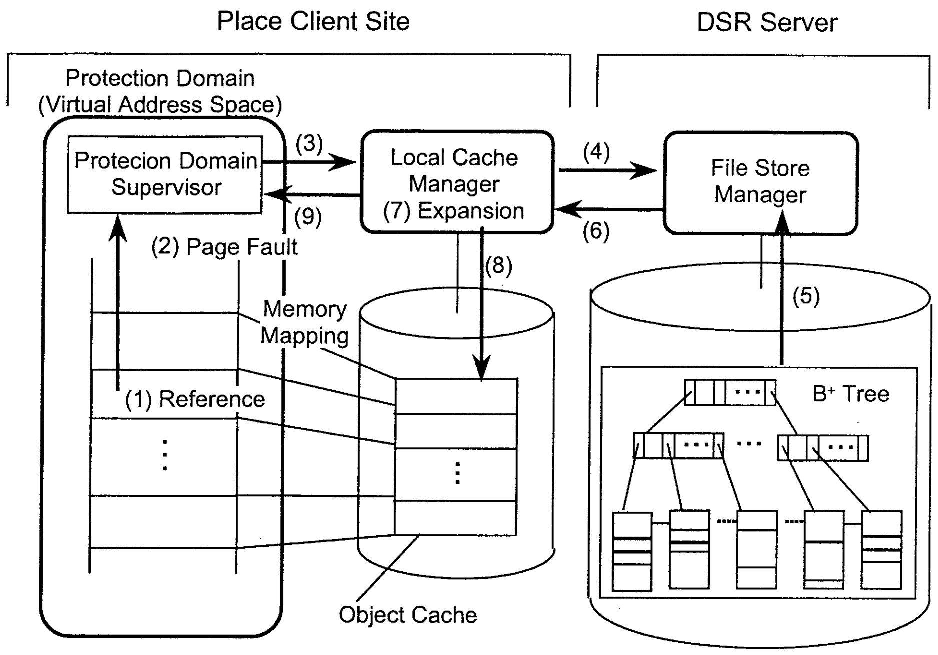


Figure 5: Implementation of remote memory-mapping mechanism.

causes a page fault (2). The page fault is detected by the MMU hardware and the event is caught by the protection domain supervisor. The protection domain supervisor sends a page request to the local cache manager of the client site along arrow (3). The local cache manager at the client site manages sharing and local caches of objects. When the local cache manager does not have the most recent copy of the requested page, it forwards the page request to the file store manager in the DSR server that manages the object port containing the processed object by using the communication line (4). The file store manager reads the requested page from persistent storage which is managed by a B^+-tree structure. After the completion of read operation (5), the file store manager sends back the page to the local cache manager in the client site by using the communication line (6). Step (7) will be explained in the next paragraph. The local cache manager stores the received page in the local object cache (8), then it acknowledges the completion of the page transfer along the arrow (9) and the protection domain supervisor resumes the execution of the object code.

To improve the speed of communication, we incorporated a data compression technique into the remote memory-mapping mechanism. An object is divided into several fragments (a fragment consists of several virtual pages), and each fragment is compressed just before being stored in the B^+-tree structure. When a portion of an object is first required by a client, the compressed fragment containing the required portion is transferred from the DSR server to the client, and it is expanded just before the fragment is stored in the local object cache at the client site. This expansion timing is designated by (7) in Fig. 5.

4 Implementing Mobile Web Robots with PLANET

We will now describe how the mobile Web robots are implemented within the PLANET mobile object system. Section 4.1 describes PLANET/C++, a tool to describe mobile robots executed in native codes, and Section 4.2 concentrates on mobile Web robot issues.

4.1 PLANET/C++: A Tool to Describe Mobile Robots

PLANET/C++ is the language mapping of C++ to the PLANET mobile object system. Please note that C++ language was originally not suited for mobile object computing in an open environment such as the Internet for several reasons. First, C++ has pointer arithmetic and any data in a virtual address space can be potentially accessed, thus the object encapsulation concept cannot be guaranteed at runtime. Furthermore, C++ ordinarily does not have dynamic loading and linking mechanisms. These unsuitable characteristics were a crucial part of the motivation to design a novel object-oriented programming language, e.g., Java [9]. PLANET gives any programming language the capabilities of performing mobile object computing in an open network environment. The above-mentioned unsuitability of C++ can be overcome by porting a C++ language processor to PLANET, and our PLANET/C++ is such an implementation. PLANET/C++ adds the following functionalities to the C++ language.

(a) Dynamic loading and unloading of objects via DSR

The programmer can declare that a class is used to instantiate mobile objects by adding `mobile` qualifier before the `class` declaration. Figure 6 shows a class definition example that is actually used in the experiment in Section 5. In the example, objects instantiated from the `WebRobot` class become mobile. A mobile object can be loaded from DSR to a protection domain and unloaded from the protection domain to DSR. As described in Section 3, the loading/unloading mechanism is implemented using the remote memory-mapping technique, thus only the accessed portion of a mobile C++ object is physically loaded and only the modified portion is physically unloaded.

A mobile object may contain address references (so-called pointers). In the above example, member variables `server`, `location`, `unsolved`, `point` and `next` contain address references. The address references must refer to either addresses within the object itself or addresses of the mobile heap area associated with each object. The runtime memory management module of PLANET/C++ allocates and associates an individual heap area for each mobile object. When a mobile object is moved, the associated heap area is moved with it. The runtime system manages the location of address references in mobile objects and it automatically adjusts the references according to the address into which the object is loaded.

(b) Orthogonal inter-protection-domain communication

PLANET/C++ supports a notable functionality that many other mobile object languages do not

```
mobile class WebRobot { // ‘‘mobile’’ is a syntax extension to C++.
public:
  int url1(char *);
  int url2(char *, char *, char *);
  int web_robot_main(dsrobject_t );
  int solve(char *);
  char *hashchecker(char *);
  void hashin(char * , char *);
  int hashfunc(char * );
  void resultprint();
private:
  char server[URL_BUFFSIZE];
  char location[URL_BUFFSIZE];
  LIST *unsolved;
  int position;
  static struct _hash {
    char *point;
    struct _hash *next;
  } HASH[HASH_ENTRYSIZE];
};
```

Figure 6: Class definition example in PLANET/C++.

support but which is very useful in practical mobile object computing. There are two possible cases when two mobile objects communicate with each other in a place: the first is where two mobile objects are loaded into the same protection domain, and the second is where they are loaded to distinct protection domains. If the programmer is required to write different codes for the two cases, this decreases the reusability of mobile objects. To solve the problem, PLANET/C++ supports *orthogonal inter-protection-domain communication* functionality. When programming, the PLANET/C++ programmer only notices the object encapsulation barrier of object-oriented programming, and he/she does not have to be concerned about the protection domain barrier provided by the PLANET runtime system. Thus the slogan is: write once, compile once, and run in both cases.

Support of the orthogonal protection domain is expected to contribute to both the software productivity and execution efficiency of a mobile object system. The same program code can be used regardless of the protection domain setting at runtime. When two mobile objects communicate within the same protection domain, they can interact with each other in a lightweight way through the omission of safe execution checking and the elimination of context switching overheads. When two mobile objects in distinct protection domains are required to communicate with each other, execution speed decreases slightly but object interactions are rigorously checked by the policy modules associated with each protection domain.

(c) Implementation of orthogonal inter-protection-domain communication

Figure 7 illustrates our scheme to implement orthogonal inter-protection-domain communication. The lower part of Fig. 7 is a case where two interactive objects are loaded into the same protection domain. Dynamic linking is required to establish the "relevance" of the interacting objects in a protection domain. The dynamic linking mechanism resolves the pair exporting and importing of symbols declared as external entries in each object [13].

For interacting objects loaded into different protection domains, we provided a mechanism whereby the objects interact as if the protection domain barrier did not exist, and we also provided a policy module which can validate each inter-protection-domain interaction operation. To achieve this, PLANET takes advantage of its remote memory-mapping architecture, the object passing mechanism, and an advanced RPC technique. The upper part of Fig. 7 shows a typical case where two interacting objects are loaded into different protection domains. The two objects cannot communicate with one another directly since the protection domain has been implemented by using a virtual address space. In such cases, PLANET loads the sender and receiver proxy objects as Fig. 7 shows. The proxy encapsulates the interprocess communication primitives and marshalling/unmarshalling (sometimes called argument serialization) routines. All the side effects on the outside of the protection domain must be done through system calls in PLANET, so for each protection domain a *protection domain verifier* including a user- or administrator-supplied policy module monitors the issuing of system calls, and it interrupts the execution of the protection domain being monitored when some protection policy violations are detected.

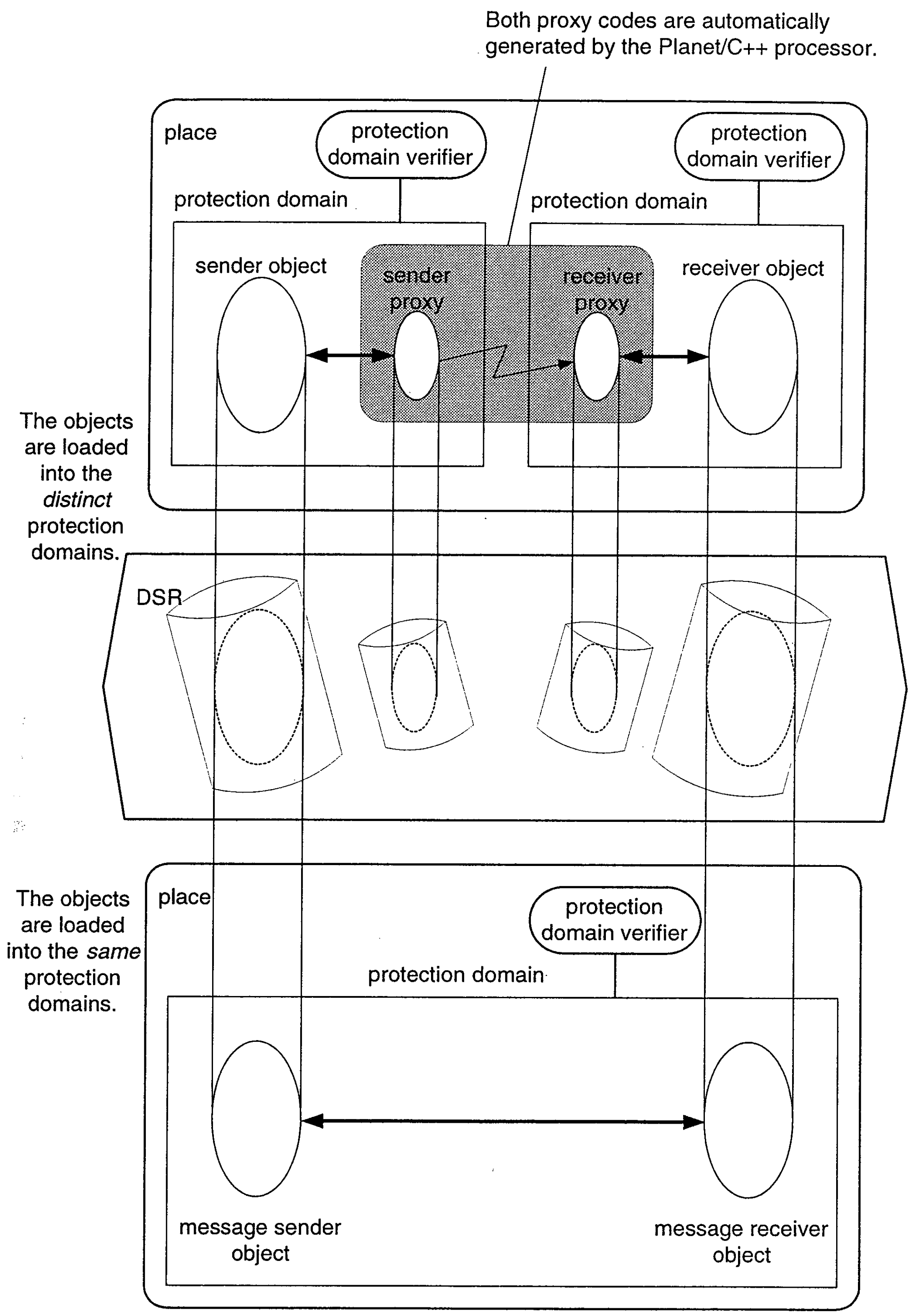

Both proxy codes are automatically generated by the Planet/C++ processor.
place
protection domain verifier
protection domain verifier
protection domain
protection domain
sender object
sender proxy
receiver proxy
receiver object
The objects are loaded into the *distinct* protection domains.
DSR
The objects are loaded into the *same* protection domains.
place
protection domain verifier
protection domain
message sender object
message receiver object

Figure 7: Orthogonal inter-protection-domain communication.

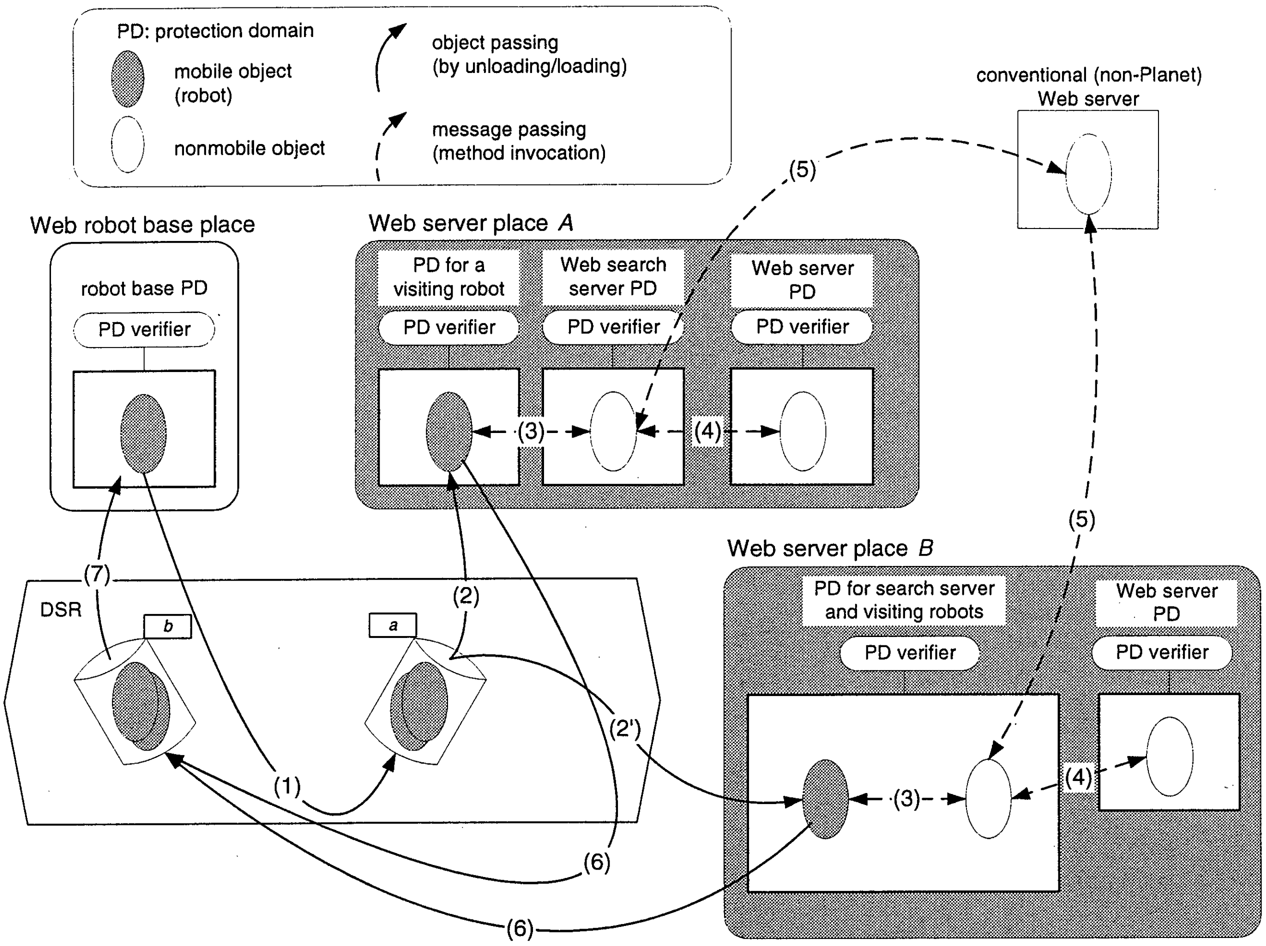


Figure 8: Typical configuration of a Web search robot system with PLANET.

4.2 Describing and Executing Web Robots

By using the PLANET runtime system presented in Sections 2 and 3 and the programming tool described in Section 4.1, we can implement a mobile Web-searching (or -indexing) robot system on both the Internet and an intranet environment as shown in Fig. 8 shows. We call a site that ships a mobile Web search robot *the Web robot base*. We call a site that receives the visits of a mobile Web-searching robot *the Web server*. We assume that the Web server and the robot base have object-receiving access rights of object ports *a* and *b*, respectively.

In the following description, the numbers correspond to the arrows in Fig. 8.

(1) The Web robot base obtains an object-sending right to its corresponding object port *a* of the target Web server. (*a* could be an open name declared by each Web server place.) Then the robot base send a Web-searching robot to the object port *a*.

(2, 2') When the Web server wants to provide a Web search service, it receives a Web search robot from the object port of name *a*. There are two possible schemes to organize a protection domain to receive a mobile Web robot: a Web robot and Web search object are either separated by protection domains (Arrow 2 in Web server place A in Fig. 8) or are not (Arrow 2' in Web server place B). The former scheme is adopted when the administrator of the Web server cannot trust the visiting Web robots, and the latter is when he can. The latter case is common when both the Web robot base and the Web server place are on an intranet. The former is more secure since each visiting Web robot is executed under the protection-domain control of its own policy. The latter is more efficient since the robot and the Web search server can directly communicate without protection-domain switching.

(3, 3') The Web robot communicates with the Web search server object to ask it to provide useful information in a place. With Web search place A (Arrow 3) the communication is indirect, and with Web search place B (Arrow 3') it is direct, yet the robot program remains completely the same due to the orthogonal inter-protection-domain communication functionality described in Section 4.1.

(4) The Web search server can be considered to be an abstraction of the Web search services on the Web server. Some intelligent Web search servers might cache the searching information to accelerate searching, or they might know of a clever way to efficiently obtain the required information.

(5) The Web search server may communicate with a conventional (non-PLANET) Web server. Thus the Web search server plays the role of "gateway" between the PLANET system and the ordinary Web system.

(6) After the Web robot collects sufficient information (or some precondition is satisfied), it is unloaded and written to object port b (the robot knows the name).

(7) The robot base receives the robot with the search information from object port b when it is ready to.

The mobile Web search robot scheme with PLANET has several advantages as follows.

Efficiency. With the scheme, the network between a Web robot base place and a Web server place is only used when the robot is moved (unloaded to and loaded from DSR). The situation is analogous to long distance calls in business; a client phones a service provider only for a request, he/she switches offline until the requested job is finished by the provider, then he/she accepts call back from the provider, to save telephone charges.

Asynchronism. The asynchronous style of sending and receiving robots is invaluable, since if such a functionality was not supported by the system, the Web robot base would first have to check whether the target Web servers were in a state that could accept the mobile search robot and it would have to support functionality to manage a robot queue and to probe the state of target Web servers. PLANET provide functionalities in a systematic way.

Protection and productivity. In an open network environment such as the Internet, a Web robot cannot be trusted in general and each Web server place has to protect itself individually. In an intranet environment, a visiting robot might be very trustworthy. The situation differs case by case, and the administrator of the Web server place should have the capability of setting a proper protection policy. PLANET has this capability. Also, without knowing the policies, the programmer can write the robot's program.

5 Experiments

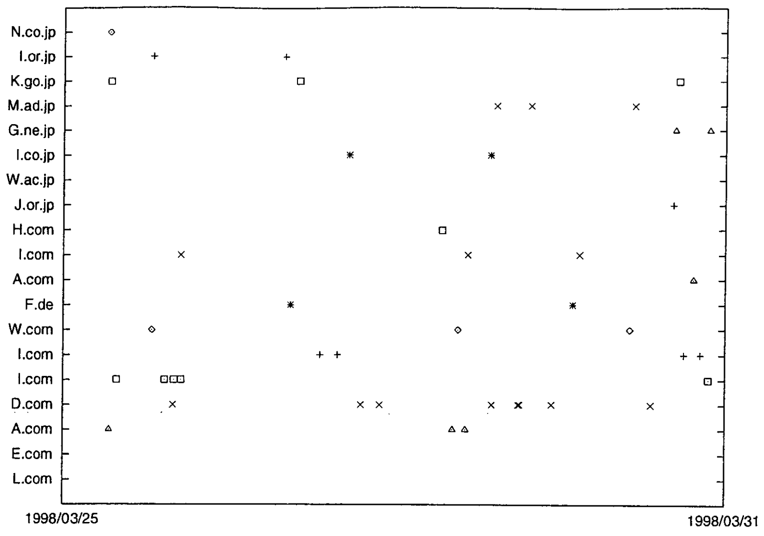
We will begin this section by showing the actual activity of conventional nonmobile Web robots in Section 5.1. In Section 5.2 we will show the experimental results for mobile Web robots implemented with PLANET. Finally, in Section 5.3 we compare mobile Web robots implemented with a native-code execution approach (PLANET) and with a bytecode interpreter approach (Java).

5.1 How much is a Web site accessed by conventional nonmobile Web robots?

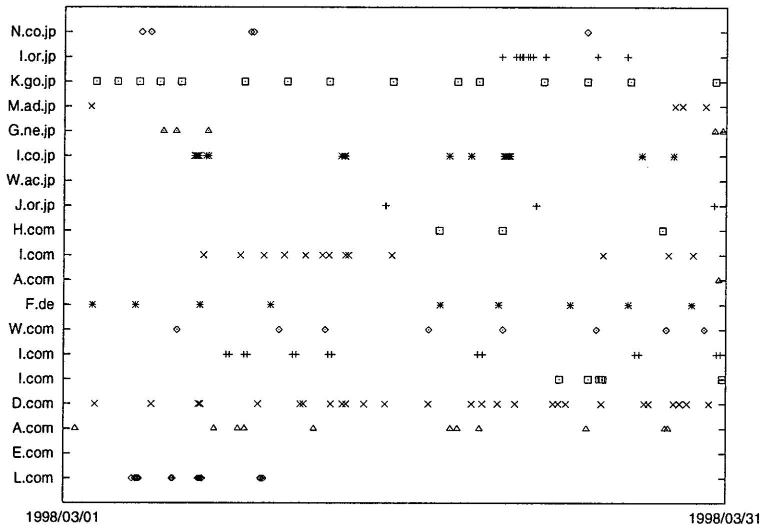
In order to observe the actual behavior of conventional Web searching robots, we analyzed the log file of our Web site. We directed our attention to accesses to the "`robots.txt`" file by the Web robots. In the Web world, a Web robot should first read "`robots.txt`" when it accesses a Web site [16, 15, 6]. Webmasters can advise robots which pages may be accessed by specifying these in the file. Figures 9(a), (b) and (c) were created by plotting the access to the "`robots.txt`" file of our non-famous Web server (`http://www.softlab.is.tsukuba.ac.jp`) for one week (1998/3/25–1998/3/31), one month (1998/3/1–1998/3/31) and one year (1997/4/1–1998/3/31), respectively. Thus one plot in the figures designates the beginning of a series of accesses by a Web searching robot. The horizontal axis represents the time periods. The vertical axis represents the site names at which nonmobile Web robots reside (the first part of each site name is substituted by its initial).

Since not all Web robots are following the "`robots.txt`" rule, the figures show a minimum access pattern for Web robots accessing our server. We can see some interesting aspects of conventional Web robots from the figures. First, a few robots accessed the Web site repeatedly for a whole year. Second, about half the robots came from domestic (Japanese) sites, and the other half came from U.S. sites. In the near future, the more Internet technology spreads throughout the world, the greater frequency of access will be. Third, on average, active robots accessed our server about 2.7 times per week. We think that this number is a practical tradeoff between the indexing performance of Web robots, the load each Web server can endure, and convenience of information providers and consumers.

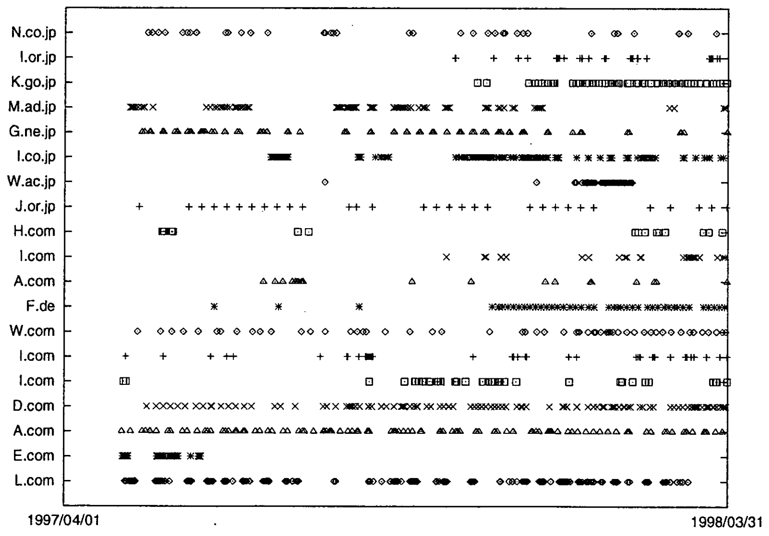
These observations support the motivation for our work; Web robots constantly consume the network resources, and improving the efficiency of Web robots is important to the Internet infrastructure.



(a) One week.



(b) One month.



(c) One year.

Figure 9: Access patterns by conventional Web robots.

Table 1: Devices used in the experiments

	Web robot base place	Web server place
CPU	UltraSPARC (167 MHz)	UltraSPARC-II (296 MHz)
Main Memory	64 MB	128 MB
OS	Solaris 2.5.1	Solaris 2.5.1

5.2 Mobile vs. Nonmobile Web Robots

We implemented a mobile Web robot system with PLANET according to the scheme described in Section 4.2. Using the implementation, we did experiments to verify the effectiveness of the proposed scheme. The devices used in the experiments are in Table 1. We assumed that a DSR server was located at the same site as the DSR client site that implemented the Web robot base. Note that this is not a limitation of our implementation; both the Web robot base place and the Web server place may have their own DSR servers. As described in Sections 2 and 3, the logical name space for DSR is singular, whatever number of DSR servers are running.

Table 2: Nonmobile vs. mobile Web robots with PLANET (in seconds).

Trial #	Scheme 1	Scheme 2	Scheme 3
1	1,117.83	705.12	643.55
2	1,141.59	851.26	783.66
3	1,132.67	865.74	819.42
4	1,135.64	868.50	816.09
5	1,139.88	866.45	818.17
6	1,132.17	866.65	814.04
7	1,139.15	892.86	834.52
8	1,188.18	875.25	843.35
9	1,136.92	886.03	820.03
10	1,157.45	878.32	827.53
Average	(i) 1,142.15	(ii) 855.62	(iii) 802.04
Advantage	–	25.1 (%)†	29.8 (%)‡

† Relative advantage to the nonmobile case on average and calculated by ((i) – (ii))/(i).
‡ The same as the above and calculated by ((i) – (iii))/(i).

We examined the following three schemes.

- **Scheme 1**: This is the same as conventional nonmobile Web robots. A robot at the Web robot base place (University of Tokyo) accesses the Web server at the University of Tsukuba by using the HTTP protocol.
- **Scheme 2**: This is essentially the scheme presented in Section 4.2 (see Fig. 10). The visiting robot is in the protection domain for visiting robots separated from the Web search server protection domain. The protection domain verifier associated with a visiting Web robot checks all the system calls issued by the visiting Web robot so that the robot only communicates

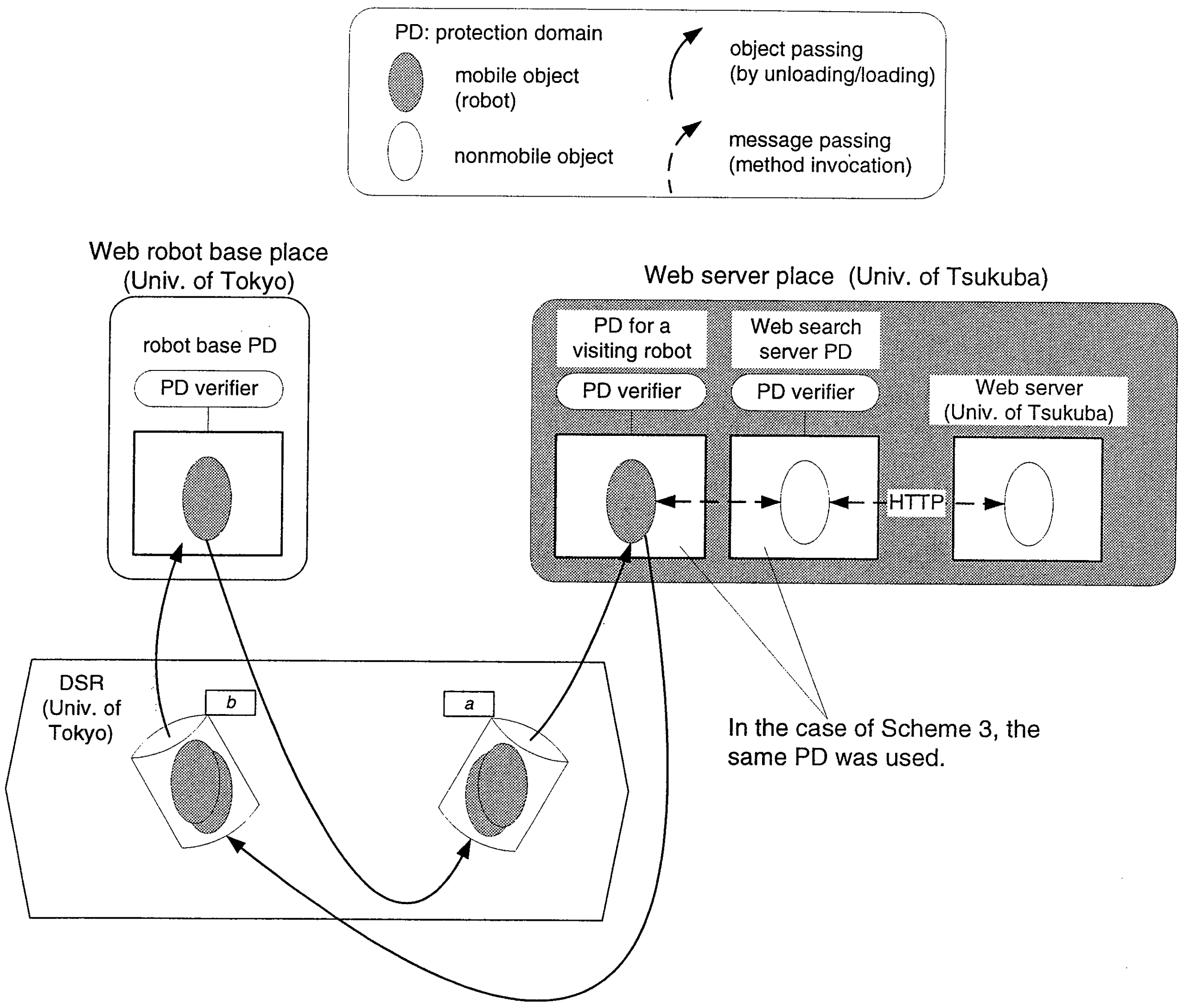

PD: protection domain
mobile object (robot)
nonmobile object
object passing (by unloading/loading)
message passing (method invocation)
Web robot base place (Univ. of Tokyo)
robot base PD
PD verifier
Web server place (Univ. of Tsukuba)
PD for a visiting robot
PD verifier
Web search server PD
PD verifier
Web server (Univ. of Tsukuba)
HTTP
DSR (Univ. of Tokyo)
b
a
In the case of Scheme 3, the same PD was used.

Figure 10: Environment used in Schemes 2 and 3 (implemented with PLANET).

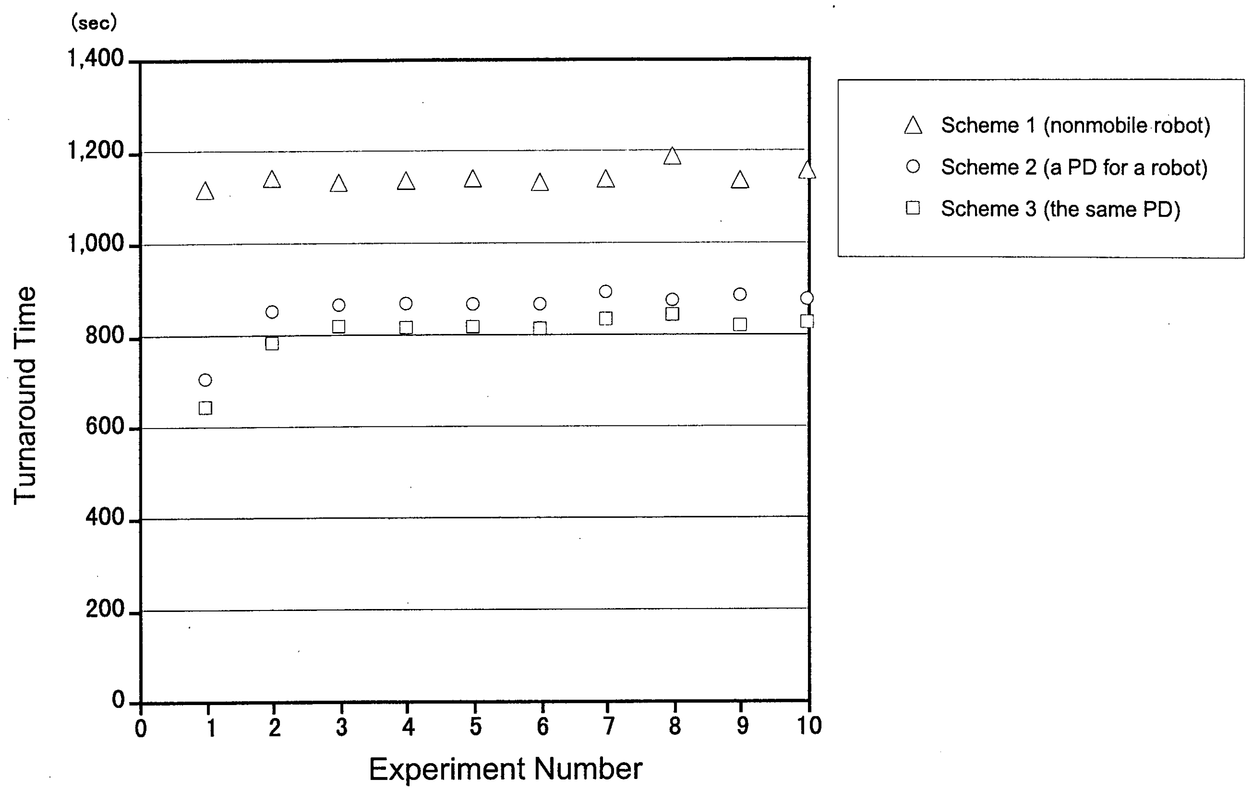


Figure 11: Nonmobile vs. mobile Web robots with PLANET.

with the object in the Web search server protection domain and that all other system call issues are inhibited.

- **Scheme 3**: The scheme is the same as Scheme 2 except that the visiting robot is loaded into the same protection domain of the Web search server (see Fig. 10).

A Web server at the University of Tsukuba, `www.softlab.is.tsukuba.ac.jp`, was used in the experiments. At the time of the experiment, the Web server stored about 4,500 files accessible from the Internet via the HTTP protocol. The size of the files was about 19 MBytes. In the experiments, the "transitive closure" of HTML pages from the top page of the Web server was computed in a breadth-first manner.

Table 2 shows the results of the experiments. Figure 10 was created by plotting the data in Table 2. The advantage of mobilizing Web robots is obvious; on average, the mobile Web robots have a 25.1% (Scheme 2) or 29.8% (Scheme 3) improvement compared to the nonmobile one (Scheme 1). The distance between the University of Tokyo and University of Tsukuba is about 60 kilometers and the the widearea network between the two universities is relatively fast. If the network was slower, there would be greater advantages.

The difference between Schemes 2 and 3 is about 5% on average. The overhead is caused by interprocess communication, process switching, and verification processing in the protection domain

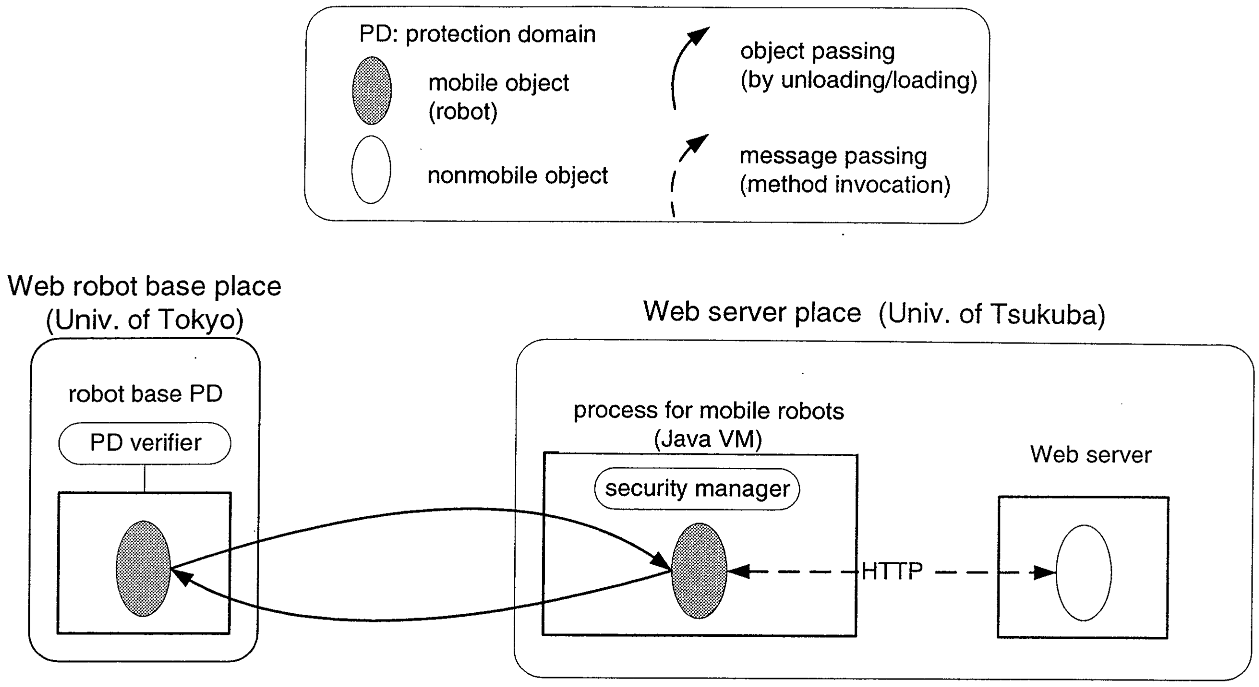

PD: protection domain
mobile object (robot)
nonmobile object
object passing (by unloading/loading)
message passing (method invocation)
Web robot base place (Univ. of Tokyo)
robot base PD
PD verifier
Web server place (Univ. of Tsukuba)
process for mobile robots (Java VM)
security manager
Web server
HTTP

Figure 12: Environment used in Schemes 5 and 6 (implemented with Java and Voyager).

verifiers. From the experiments, we think that the overhead is reasonable. Hence we recommend that the Scheme 2 setting should be used in an open environment such as the Internet and Scheme 3 can be used in closed environment or so-called intranets.

5.3 Native Code Execution Approach vs. Bytecode Interpretation Approach

To compare the native-code-based object mobility adopted in PLANET and the interpreter-based approach adopted in most other mobile object systems, we implemented a mobile Web robot with Java. Java by itself can only provide mobility of classes (or codes) and it cannot provide object mobility. For object mobility functionality, we used one of the most sophisticated and easy-to-use Java-based mobile object systems, Voyager, developed by ObjectSpace, Inc. [22, 8] and implemented the following three schemes:

- **Scheme 4**: This is the as same as conventional nonmobile Web robots. A robot at the Web robot base place of University of Tokyo accesses the Web server at the University of Tsukuba by using the HTTP protocol.
- **Scheme 5**: A mobile robot is moved between the Web robot base place at the University of Tokyo and the Web server place at the University of Tsukuba (see Fig. 12). In the scheme, the visiting robot is executed under security control by the Java *security manager* [1, 5], which hooks each issue of the system resource accessing library and checks the issue with respect to the policies specified by the administrator of the Web server place. The security manager checks that the mobile robot only communicates with the Web server.

Table 3: Nonmobile vs. mobile Web robots with Java and Voyager (in seconds).

Trial #	Scheme 4	Scheme 5	Scheme 6
1	1,539.08	1031.06	908.46
2	1,545.04	1038.95	884.85
3	1,556.13	1012.68	858.53
4	1,587.28	1031.07	881.38
5	1,664.99	1068.37	865.79
6	1,617.30	1110.64	886.47
7	1,635.15	1047.53	865.51
8	1,847.00	1046.91	872.50
9	1,689.34	1057.99	968.19
10	1,663.66	1047.56	966.51
Average	(i) 1,634.50	(ii) 1049.28	(iii) 895.82
Advantage	–	35.80 (%)†	45.19 (%)‡

† Relative advantage to the nonmobile case on average and calculated by ((i) – (ii))/(i).
‡ The same as the above and calculated by ((i) – (iii))/(i).

- **Scheme 6**: This is the same as Scheme 5 except that security manager functionality is not used.

The execution environment in the experiment was the same as that described in Section 5.2, but Schemes 4–6 were used instead of Schemes 1–3. Note that asynchronous object passing functionality was not available in Schemes 4–6, The Java bytecode compiler and virtual machine (VM) we used were those included in JDK 1.1.5 (the virtual machine does not have a just-in-time compiler for the SPARC). Table 3 shows the experimental results. Figure 13 was created by plotting the data in both Tables 2 and 3.

By comparing Schemes 1 and 4, the average bytecode interpreter execution is about 40 % slower than the native code execution with PLANET. Thus for a nonmobile Web search system, the use of Java is not attractive.

For a closed network (so-called "intranet") environment, Scheme 3 or Scheme 6 can be used. Comparing the averages of the schemes, Scheme 3 is more efficient by about 12%. In an open network environment such as the Internet, mobile Web robots must be implemented with Schemes 2 or 5. Comparing the two schemes, Scheme 2 is about 23% more efficient than Scheme 5. If these differences are negligible, using the Java system is an attractive choice since it has a high degree of multiplatform-interoperability. Currently, PLANET is running on Solaris 2.5.x for the SPARC platform, but the Java system can run on many more platforms. However, if we can use a platform on which PLANET is available, we can give several reasons why the PLANET approach should be selected. First, the performance of execution is significant in the Web search system. The better the execution performance is, the more the search frequency can be raised. Second, asynchronous object passing functionality, which is available in PLANET, is not available in the Java system; Java programmers have to describe the program code for probing the server state or provide a clever scheduling for object passing.

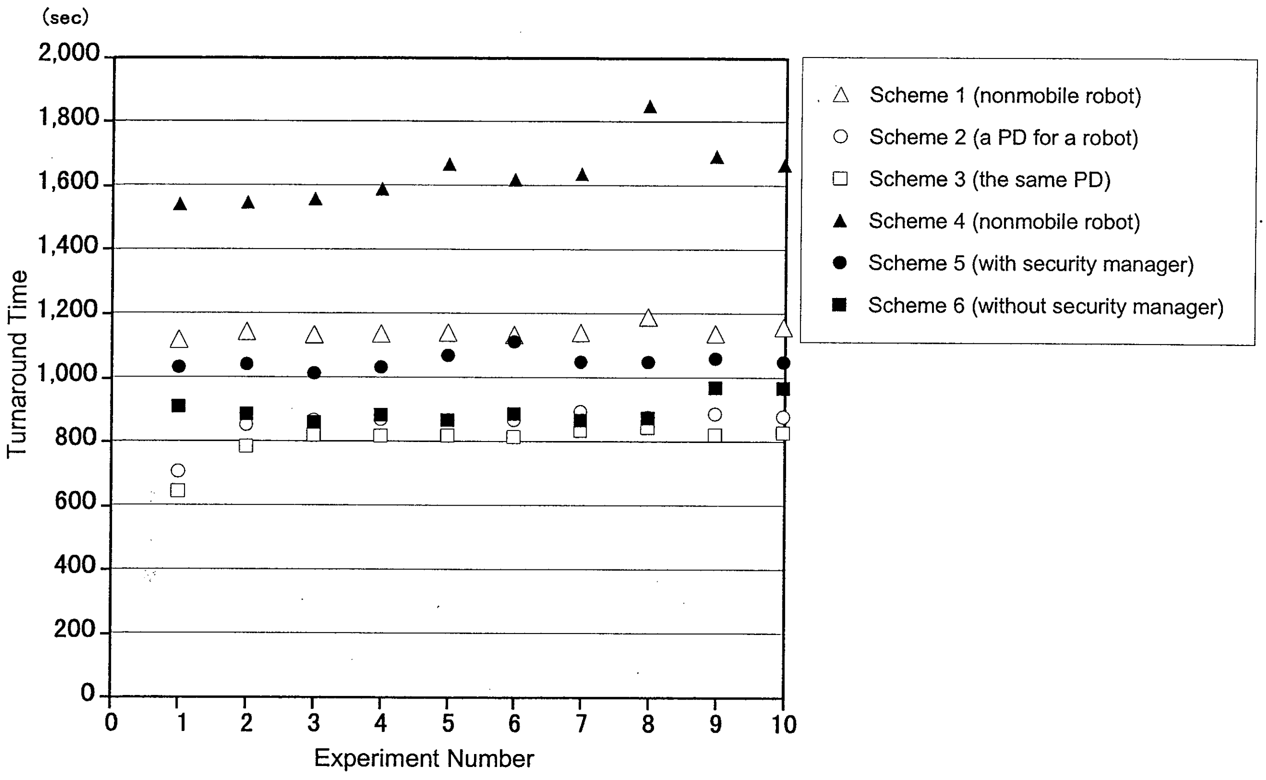


Figure 13: Comparison.

6 Conclusion

We presented the logical and physical structure for the PLANET mobile object system. Among the many recent proposals for mobile object systems, the system is characterized by language-neutral layered architecture, the native code execution of mobile objects, and asynchronous object passing via DSR. We proposed an approach to implement mobile Web robots by fully utilizing these characteristics, We verified and discussed its effectiveness based on experiments conducted in the Internet environment.

We believe that mobile object computing technology will play important roles in worldwide networks. The WWW is undoubtly the most important area of application in current worldwide networks, and we showed how our technology can contribute to the Web.

The design of a mobile Web robot system raises many research problems that need to be solved. First, a robot base may ship a few robots simultaneously. We should research how to control the parallelism of Web robots. Second, a Web server may be visited by a few robots simultaneously. Therefore, we should do research on the effective scheduling of robots on a Web server. Third, in the experiments described in Sections 5.2 and 5.3, the Web search server objects were not intelligent; they could be made more intelligent and efficient by caching some information [11] on the Web server or by providing higher-level abstractions. Fourth, Web robots could interchange their collected knowledge. DSR could become an effective tool in achieving such functionality.

Acknowledgement

We would like to thank Akinori Yonezawa, Kenjiro Taura and various members of their laboratory at the University of Tokyo for cooperating with us in conducting the experiments. We would also like to thank Shigeru Chiba of the University of Tsukuba who developed Open C++ [7], a useful extensible C++ language processor. We used Open C++ to implement PLANET/C++ and it contributed to shortening the development time. We wish to thank ObjectSpace, Inc. who developed Voyager [22], an efficient and easy-to-use mobile object system in Java.

The work was partly supported by the Parallel and Distributed Consortium.

References

[1] K. Arnold and J. Gosling. *The Java programming language.* Addison-Wesley, second edition, 1997.

[2] L. F. Bic, M. Fukuda, and M. B. Dillencourt. Distributed computing using autonomous objects. *IEEE Computer*, Aug. 1996.

[3] C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz. The Harvest information discovery and access system. In *Proc. of the 2nd Int. World Wide Web Conf.*, pages 763–771, Oct. 1994.

[4] L. Cardelli. A language with distributed scope. *Computing Systems*, 8(1):27–59, Jan. 1995.

[5] P. Chan and R. Lee. *The Java Class Libraries — An Annotated Reference.* Addison-Wesley, 1996.

[6] F. C. Cheong. *Internet Agent.* New Riders Publishing, 1996.

[7] S. Chiba. A metaobject protocol for C++. In *Proc. of OOPSLA'95*, pages 285–299. ACM, 1995.

[8] G. Glass. ObjectSpace Voyager: the agent ORB for Java. In *In Proc. of 2nd Int. Conf. on Worldwide Computing and Its Applications*, volume LNCS-. Springer-Verlag, 1998.

[9] J. Gosling and H. McGilton. The Java language environment: A white paper. Technical report, Sun Microsystems, 1995.

[10] E. Jul, H. Levy, N. Hutchinson, and A. Black. Fine-grained mobility in the Emerald system. *ACM Trans. Computer Systems*, 6(1):109–133, Feb. 1988.

[11] K. Kato and T. Masuda. Persistent caching: An implementation technique for complex objects with object identity. *IEEE Transactions on Software Engineering*, 18(7):631–645, Jul. 1992.

[12] K. Kato, K. Matsubara, K. Toumura, S. Aikawa, and Y. Someya. Object passing and interaction mechanism of the PLANET mobile object system. In *France-Japan Workshop on Object-Based Parallel and Distributed Computation*, pages 153–162, 1997.

[13] K. Kato, A. Narita, S. Inohara, and T. Masuda. Distributed shared repository: A unified approach to distribution and persistency. In *Proc. 13th IEEE Int. Conf. on Distributed Computing Systems*, pages 20–29, May 1993.

[14] K. Kato, K. Toumura, K. Matsubara, S. Aikawa, J. Yoshida, K. Kono, K. Taura, and T. Sekiguchi. Protected and secure mobile object computing in PLANET. In *Special Issues in Object-Oriented Programming—Workshop Reader of the 10th European Conference on Object-Oriented Programming*, pages 319–326. dpunkt.verlag, 1997.

[15] M. Koster. Guide for robot writers. http://ww.nexor.co.uk/mak/doc/robots/guidelines.html, 1994.

[16] M. Koster. A standard for robot exclusion. http://ww.nexor.co.uk/mak/doc/robots/norobots.html, 1994.

[17] M. Koster. Robots in the Web: threat or treat? *ConneXions*, 9(4), Apr. 1995. Available in http://info.webcrawler.com/mak/projects/robots/guidelines.html.

[18] D. B. Lange and M. Oshima. Aglets: Programming mobile agents in java. In *Proc. of Worldwide Computing and Its Applications*, volume LNCS-1274, pages 253–266. Springer-Verlag, Mar. 1997.

[19] K. Matsubara, T. Maekawa, and K. Kato. Worldwide component scripting with the PLANET mobile object system. In *Proc. of 2nd Int. Conf. on Worldwide Computing and Its Applications*, LNCS-1368, pages 56–71. Springer-Verlag, 1998.

[20] M. L. Mauldin. Measuring the Web with Lycos. In *Proc. of the 3rd Int. World Wide Web Conf.*, Apr. 1995.

[21] O. McBryan. GENVL and WWWW: Tools for taming the Web. In R. Cailliau, O. Nierstrasz, and M. Ruggier, editors, *Proc. of the 1st Int. World Wide Web Conference*, 1994.

[22] Inc. ObjectSpace. http://www.objectspace.com/.

[23] B. Pinkerton. Finding what people want: Experiences with the WebCrawler. In *Proc. of the 2nd Int. World Wide Web Conf.*, 1994. http://webcrawler.com/WebCrawler/WWW94.html.

[24] R. Seltzer, E. J. Ray, and D. S. Ray. *The AltaVista Search Revolution.* McGraw-Hill, 1997.

[25] J. Vitek and C. Tschudin, editors. *Mobile Object Systems.* LNCS-1222. Springer-Verlag, 1997.

[26] J. E. White. Mobile agents. In J. Bradshaw, editor, *Software Agents.* MIT Press, 1996.

[27] B. Yuwono and D. L. Lee. WISE: a world wide web resource database system. *IEEE Trans. Knowledge and Data Engineering*, 8(4), Aug. 1996.