Towards a World-Wide Civilization of Objects

Michael Condict†, Dejan Milojicic†,Franklin Reynolds‡, Don Bolinger†

†Open Software Foundation Research Institute
11 Cambridge Center
Cambridge, MA 02142
{condict,dejan,bolinger}@osf.org

‡Apple Computer, Inc.
1 Main Street
Cambridge, MA 02142
fdr@cambridge.apple.com

Abstract. The Internet today corresponds to a Feudal society, where fortress walls (firewalls) surround villages (LANs), little pockets of civilization connected by lawless highways (insecure networks) infested by bandits (hackers). The emergence of the World Wide Web and Java have shown the way towards a true civilization of electronic objects, although it does not yet exist. To assist in its evolution, we propose to extend the World Wide Web and Java with object-oriented, distributed OS services, implemented in Java. The goal is to make it commonplace for applications and objects to be distributed across unrelated sites and for mobile objects and agents to travel and trade freely around the world. We describe some problems such a system faces, such as in the areas of security and performance, and give a high-level design that addresses them.

1. Introduction

The current state of civilization on the Internet can be likened to feudalism, or perhaps to the tribalism that preceded it. The world consists of tribes or villages (LANs) protected by fortress walls (firewalls) from the barbarians running amok on the high roads (insecure networks). Only village residents are allowed to pass through the walls. Foreigners are allowed to stand outside and conduct trade or other business as best they can with the residents inside, but this isn't at all like travelling into the village square.

Outside of these villages, anarchy prevails. Bandits (hackers) travel at will ready to plunder and pillage anywhere they can breach a wall. There is no world-wide civilization in the conventional meaning of that term, i.e. an agreed upon set of laws and structures allowing societal interactions such as trade and travel to take place reliably and efficiently, with predictable consequences.

To establish a true global electronic civilization, we need to replace the fortress walls with structures that allow safe and efficient travel and trade between electronic entities around the world. The World Wide Web [Bern94] is the first such structure to be deployed across a significant percentage of the world's LANs, but it only allows a small subset of the desired functionality. All the same, it has subsumed (or soon will) most uses of the Internet. It is here to stay. The emergence of the Java portable programming environment [Sun94] has enhanced this functionality significantly. So the question is not what life after the Web will be like; it will evolve by incremental enhancements into something like the *cyberspace* portrayed in the novels of William Gibson [Gibs84]. The question is how to assist in that evolution.

To that end, as a direct extension of today's Web, we intend to build a portable, distributed, object-oriented environment, herein called the Trusted Internet Environment (TIE). It will be based on globally named distributed objects, optionally persistent, and will contain a hierarchical name service integrated with hierarchical object caching. One of its main innovations, however, will be the support of mobile objects and agents, together with the enhanced security they require. Finally, TIE will provide a high level of configurability.

TIE will be implemented initially as portable middleware in Java, so that it can run in any Java enabled computer. (The fully functional version may require enhancements to the Java runtime and to the language itself.) Technically, we could use any of several other portable/scripting languages, such as Tcl/Tk [Oust94], Oblique [Card95] and Python [Pyth95], but Java is rapidly becoming the *lingua franca* of the Web, just as English is for world commerce.

Much of the functionality in TIE is intended to allow a new set of policies by which organizations interact with the Internet. The intention is that it will become commonplace for most companies to allow mobile objects (customers) to travel into their public LAN (store) and invoke operations on the objects there. It will be equally commonplace for users to allow mobile objects (travelling sales-objects, entertainers or information providers) to enter their computer and interact with it, confident that their personal electronic possessions will not be stolen or destroyed. Finally, it will be the rule, rather than the exception that communications in this new environment are private, and that one knows who one is communicating with.

In Section 2 (*The Object Model*) we describe the object model of TIE and how it provides scalable object caching and naming services. We also show how the existing notion of a persistent Web object is extended in TIE to provide fully functional persistent objects, loaded on demand and transparently updated.

Section 3 (*Mobile Objects and Agents*) motivates the use of mobile agents on the Internet, and discusses problems of implementation and security associated with them.

The most important enabling technology necessary to allow civilization of the Internet is security sufficient to allow business to be transacted with little risk of fraud and other malicious acts. Section 4 (*Security*) discusses the problems of local security (mostly, protecting the local node from foreign code) and network security (protecting the privacy and integrity of cached distributed objects, mobile agents and remote method integration) and how these problems are addressed in TIE.

Section 5 (*Related Work*) compares our project to similar efforts by others. While most of the functionality of TIE has been implemented before in isolation, it is the integration of such key features as security, a portable distributed object environment and mobile agents with the existing World Wide Web that is innovative, and may prove to have revolutionary effects.

2. The Object Model

TIE is an object-oriented operating system environment distributed across the Internet. It supports persistence (e.g. World-Wide-Web documents), and objects can migrate or can be distributed and cached across the network. Alternatively, stationary objects can be operated on by remote method invocation (RMI), the object-oriented equivalent of RPC. For each method implemented by an object, the programmer can specify whether the invocation causes the object to be cached at the location of the invoker and operated on locally, or an RMI message is sent to the object, which is then operated on remotely.

The implementation language of TIE is Java, and its objects are Java objects, extended to allow objects with distributed state , *active* objects (containing threads of control) and migrating objecs. These extensions mostly can be implemented in unenhanced Java, i.e. without modifying today's Java run-time environments.

The executable unit in TIE is the Java class file. TIE is a distributed, component-based execution environment. Java class files, which we sometimes call "modules", are both the factories and servers for its components. In order that the instantiated objects satisfy the usual desirable properties of distributed components, we require that there be no public data members in the public class implemented by a Java class file, and that the programmer identify (to the IDL compiler) both the set of interfaces implemented by the public class and the set of interfaces imported by it (i.e. the interfaces used by it, but not implemented internally).

In TIE, as in current Java usage, multiple modules (class files) may be loaded into a running Java-enabled program, wherein they can directly access each others components without using RMI. We call this Java runtime environment a *task*, or just an *address space*.

Rather than a pure Web-server/Java-client model as in current use of Java on the Web, TIE will allow arbitrary distribution of running modules across any number of address spaces and nodes, cooperating as peers in implementing an application. In order to be able to load modules transparently anywhere, there will be a TIE object server running on most nodes. It is a Java-enabled task whose function is to accept requests to load modules and objects, and then to accept remote method invocations on those objects. It will be accessible by URL, and could even use the http protocol (or ftp) to allow remote caches to obtain copies of distributed objects, and to receive back modified objects as they are flushed from remote caches.

2.1 Remote Method Invocation

To support RMI, there is an Interface Description Language (IDL) and an IDL compiler, which, given an abstract class declaration, generates a client-side and a server-side stub function (in Java) for each operation in that class. (With small extensions, Java interface declarations may be usable as the IDL.) In addition to transmitting remote method invocations across the Internet, the stubs also provide hooks allowing users to extend the semantics of RMI to include such functionality as security and replication.

There are several reasonable choices of network protocol that could be used to implement RMI, including OSF's DCE RPC [Rose92][Schi93], the OMG's CORBA IIOP protocol [Sole95], or the protocol used by JavaSoft to implement RMI [Woll96]. DCE RPC would provide security advantages, IIOP would allow invocations on CORBA objects independent of CORBA vendor, and the JavaSoft RMI protocol would preserve compatibility with what will probably become an important distributed Java environment.

2.2 Object Caching and Consistency

To support distributed shared objects, there is a hierarchical, persistent object cache. We believe that in the near future the shortage of available network bandwidth and the growing number of users for popular Internet servers will make it imperative to do aggressive caching of distributed objects to avoid server bottlenecks. For the same reason, it is undesirable for each local cache to directly communicate with the object's server. Thus, each address space on a node will cache (in memory) copies of the objects it is operating on locally, and each node will have a disk cache to supply objects efficiently to the per-address-space memory cache. Similarly, each LAN could have a very large disk cache on a server node, which is used to feed the per-node disk cache. The LAN cache in turn may be connected to a per-organization cache, etc. (Figure 1) Such a hierarchy may also exist on the server side, as in Harvest [Chan96], or may be provided between client and server organizations as a third-party service.

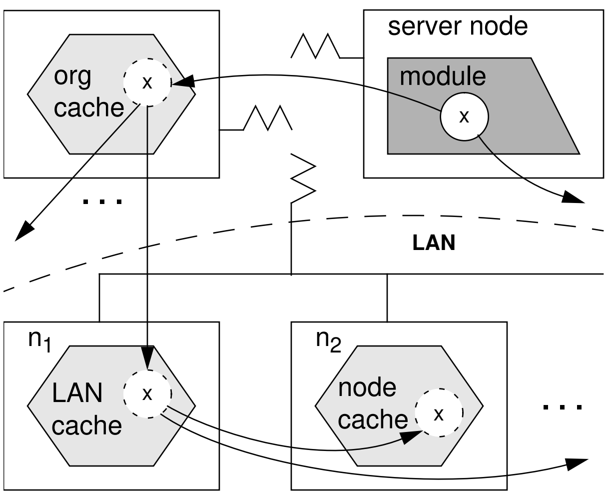

Figure 1. Hierarchical Object Caching

For each object that can be distributed, the IDL compiler generates two stub functions, one for getting a copy of the object into the cache, and one for sending a modified object back to the node from which it was obtained (which may be another cache). The caching service uses these functions to implement cache coherency. Programmers can replace these default functions with more efficient object-specific cache update functions that use application-specific network protocols to improve performance.

The safe use of distributed shared objects requires operations that ensure mutually exclusive access. TIE provides locking functions integrated with the object cache. The particular cache coherency and locking policies used by TIE can be replaced by the user. By default, the system uses entry consistency [Bers93] for cache coherency. The default locking policy, which is token-based, has the server keep track of which client, if any, has the unique write token. But, to scale successfully, it requires readers to contact the server before each use (after some user-definable grace period) to find out whether they still hold a valid copy.

2.3 Name Service

To allow users to locate and invoke remote objects using human-readable identifiers, TIE incorporates a world-wide hierarchical name service. Instead of attempting to define a new syntax for names, we note that the Web already has a workable convention, the Universal Resource Locator (URL), which we will continue to use as the default naming syntax (until it is superceded by the use of the more location-transparent URN).

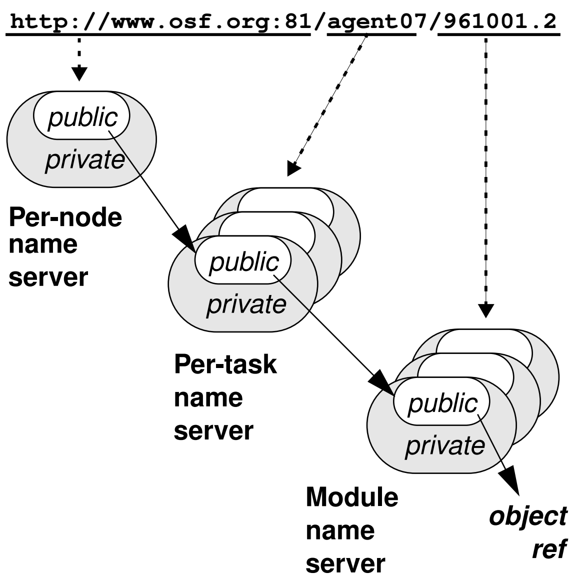

Figure 2. Hierarchical Name Service

The job of the name service in TIE is to provide an object reference, given the name of an object, as shown in Figure 2. An object reference is an efficient handle for an object, which unlike the name, is location-specific. The user invokes a bind operation on a name server to convert a name to a reference. The reference can then be used to invoke operations on the named object. Another job of the name service is to provide location transparency. By interacting with the hierarchical object cache, it transparently provides a reference to the nearest, or otherwise best, copy of an object that may be cached on several nodes. (When the object migrates or is flushed from the cache, this reference will need to be updated, using forwarding with notification to the sender, for example.)

The basic object of the TIE naming service is a directory object, which is simply a set of identifier/object-reference pairs defining a human-readable name for the referenced object. Since the object reference can denote another directory object (not necessarily within the same name server module), this allows the construction of a name space that is an arbitrary directed graph, with identifiers labelling the arcs. The identifiers in the “pathname” portion of a URL specify a path through this directory graph, starting at the per-node name server identified by the hostname (and port number) portion of the URL. Processing such a pathname may involve nested RMI to any number of different name servers, possibly on different nodes.

By convention, the namespace graph will usually be a tree, and mostly hierarchical in the follow sense: the per-node name server will define a flat name space into which any running module can insert an entry that refers to a per-task name space. Similarly, each module will have its own name server, and may insert a reference to a portion of its name space into the per-task name space.

The TIE name service is *federated*, meaning that the interfaces to it are specified, and it is expected that multiple implementations of these interfaces (one implementation is provided with TIE) can be linked together into a worldwide directory graph The configurability of TIE will also allow its federated naming service to be integrated with other existing federated naming systems, e.g. CORBA [Sole95] and X/Open Federated Naming [Xope95]).

2.4 Object Mobility

Objects in TIE can migrate from one node to another. In implementation, this migration is similar to caching an object on multiple nodes, but mobile objects may be employed for other purposes. Objects can also be active, i.e. can contain autonomous threads of control that do more than simply respond to external operation invocations. An active object is said to be an agent if it carries out some useful background task on behalf of a user. The threads of an active object migrate with it. Section 3 (*Mobile Objects and Agents*) describes this in more detail.

2.5 Persistent Objects

Some objects are persistent, meaning the object can be referred to by the same name indefinitely, even across reboots of any nodes that have a copy of the object, and that its state is preserved as well. An example of a persistent, named object is a Web document named by a URL.

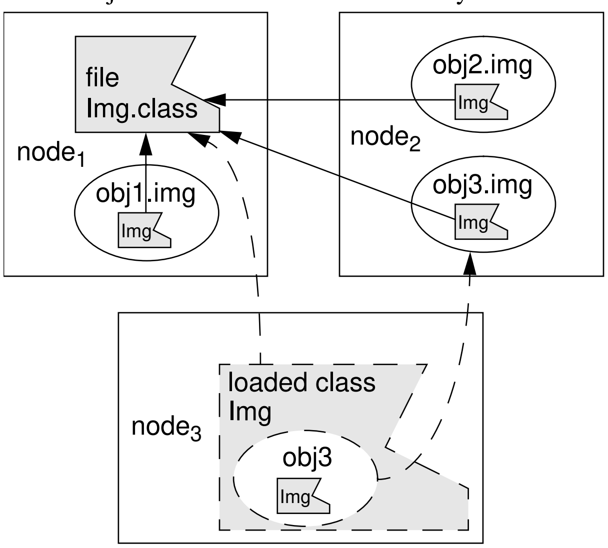

Figure 3. Persistent Objects and Classes

In order to integrate persistent Java objects smoothly with Web objects, a persistent TIE object is stored as a file that contains header information referring (by URL) to the Java class of which it is an instance. This allows TIE to transparently load the module (class file) that knows how to manage the object, if not already loaded, at the time the persistent object is loaded. In addition, there is class header information that describes where to load the module. It can be specified that it should be loaded into the address space of the module that called for the loading of the persistent object; or it could be loaded into the TIE object server on the node where the module is stored, or on any third node.

An example of this is shown in Figure 3. There is a class “Img” that has three persistent objects, stored as files “obj*.img” on two different Internet nodes. A third node ($node_3$) requests the loading of obj3. The class header for the Img class specifies that the Img module should be loaded into the address space of the caller, so the Img class is loaded onto $node_3$ from the file Img.class on $node_1$. As it is loaded, a parameter consisting of the full URL for obj3.img is passed to its startup code, causing it to transparently instantiate a new object in memory and load the contents of the file into it (using the cache “get” function generated by the IDL compiler), thus activating obj3.

2.6 Extensibility and Performance

TIE is highly configurable. Most parts of the framework are objects which can be replaced by the user. An example of this is in the invocation of objects. As in Spring [Hami93], we allow the user to specify the methods to be invoked to marshal and unmarshal arguments in an RMI and/or the method to send the RMI message itself. This allows such functionality as security (via encryption), replication and real-time to be provided as plug-ins without affecting the performance of the system for users who don’t need them.

An important performance feature of TIE is that when an object and its invoker are in the same address space, the overhead of RMI stubs is completely avoided. If an object is local, the user will be given a direct reference to the object. For a remote object, the reference will be to a proxy object whose methods are IDL-generated stubs that send RMI’s over the network. This is implemented using a per-task table that maps TIE object references to direct references to local Java objects. Whenever an external reference arrives in the task as a parameter (or result) of an RMI, the table is consulted to see if the reference is to a local object and can hence be converted into a direct local reference.

3. Mobile Objects and Agents

Mobility has been a focus of research in industry and academia ever since distributed computing was introduced, motivated by the need to improve load distribution, system administration, fault resilience, etc. Despite the overwhelming interest, which continues up to the present [Stee95, Smit96, Rous96], object mobility has never really became widespread because of concerns about complexity, cost or limitations on transparency or homogeneity. However, technical solutions have now been found for each of these problems. Migration has been supported as a simple, user level mechanism [Milo93]; performance as good as 14ms for full process migration has been achieved [Rous96] and techniques have been introduced for different kinds of state transfer optimization [Thei85]; transparency has been achieved at different levels, from limited to complete; and recent work demonstrates improvements in heterogeneity [Stee95, Smit96].

Instead, we believe that the true impediment has been the lack of infrastructure, and sociological factors, such as the mindset where each desktop "belongs" to a user. The Web has changed this profoundly. In contrast to any distributed operating system, the Web has become ubiquitous, motivating would-be participants to adapt to it. Instead of a local workstation model, we have a Web environment with computers connected as interfaces to a "network-as-computer" model. Objects are relieved of the burden of precise UNIX semantics, with limited requirements for transparency; performance is dominated by network latency so migration latency is not as visible as it is on a local area network; heterogeneity is enabled via portable languages such as Java [Gosl95] and Tcl/Tk [Oust94].

We plan to extend the notion of distributed objects to encompass mobile objects and agents. Mobility will be supported as Telescript does with its "go" utility, except that in contrast to Telescript's closed environment, our mobile agents will be integrated with the Web. Key issues in object mobility range from the high-level—choosing a mobile object model, and conventions for naming, auditing and controlling mobile objects—to the low-level—such as designing a checkpointing mechanism or how to handle communication channels.

The migration mechanism addresses the problem of how to move an active object from node A to node B while preserving its control state. In current Java-enabled browsers it is only possible to move data between the browser and the server from which the executing code originated. In theory this would suffice for a two step algorithm (migrating from one node to another through the server), but this is not acceptable for performance and accessibility reasons: an object may be an agent representing a user no longer connected to the network [Rigg96].

An open problem is how to checkpoint and maintain the intermediate state accumulated on previous nodes. The most limited transparency would allow the agent to start movement only after its execution is completed, restarting it from scratch on the new node. Greater transparency would allow an agent to migrate itself by invoking a method on its own behalf. (This is the approach we shall take.) Full transparency would allow anyone to migrate any agent at any time. Although useful for some purposes, this approach introduces complexity we believe is unwarranted.

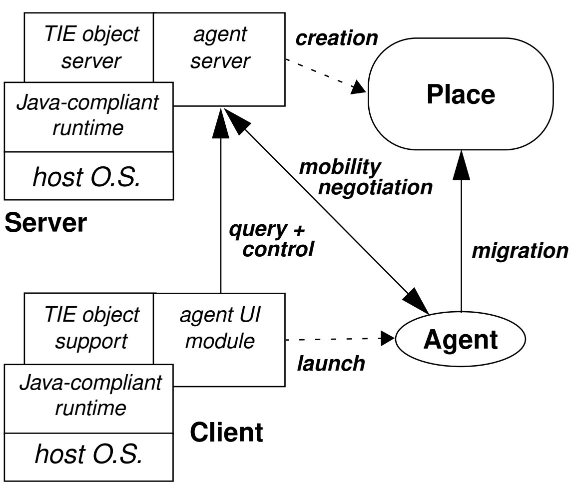

Figure 4. Initial Agent Migration

Object references need to be more dynamic than is possible using existing Web mechanisms, such as URLs. Even URNs are probably inadequate because mobility requires relatively frequent updates to location. Therefore, in TIE the URL which names a mobile object will not change when it migrates. This is also necessary to enable users to find and access mobile objects using a predictable name. The low-level reference used by the TIE runtime to invoke operations on an object will, however, need to be updateable. One way to do this is to leave behind a proxy object when an object migrates. The proxy is responsible both for forwarding any incoming method invocation to the new object location, and for notifying the invoker to update its reference to the new location.

At a lower level, communication channels that an object has opened on one node need to be reopened after it is moved to another node. Different approaches provide varying degrees of transparency. The first possibility is for an object to close all connections before migration, re-opening them afterward. To prevent the loss of messages during migration, a forwarding proxy object could be left on the original node as described above for updating object references. A less transparent (but simpler) approach refuses messages during migration and requires senders to retransmit after some period of time (this is our choice).

The mobility model that we propose supports negotiation between agents and the sites they visit using primitives such as *move*, *meet*, *fork*, *stop*, and *resume*. At least initially, both client and server nodes will need to have explicit support for MOA installed. In Figure 4, we depict the creation of an agent on a client node, and its initial migration to a server. Solid arrows indicate cross-node interactions, while dotted arrows show local operations. The environment in which an agent runs is called a "place." Agents may agree to meet in a shared place, or may request one of their own. Placement is just one of many aspects of the agent environment negotiated between the agent and its desired new host. Later agent migrations look just like the initial one, and involve new negotiations between agent and agent server.

Agent auditing will allow logging of agent activities, such as migrations and communication. It will also be applicable to servers (log visiting agents, communication, etc.) Agent control will be achieved through user accessible utilities that permit users to manipulate agents with commands such as *stop*, *kill*, and *suspend*. The TIE object browser will allow viewing agents, listing agents by node, and so on.

4. Security

The security problems in the world-wide civilization of objects fall primarily into two categories: network security and single-node security.

The network security problems concern the need for privacy, the need to authenticate the identity of clients doing RMI and the identity of objects migrating between nodes, and the need for capable but flexible distributed authorization. TIE will address authentication and confidentiality problems with standard methods such as public key cryptography and digital signatures [Rive78, Sper96]. This will allow, for example, a mobile object to pass directly from one user to another user that doesn't trust the first, without the overhead of contacting the server for the object.

In authorization, we believe that practical secure use of the Web requires expressing limited, varying and specific degrees of trust between users and providers of resources. We will leverage the Adage work in trust models at the OSF RI [Zurk96] to provide notions of amount and type of mutual trust around specific resources, and to present these notions to users in an intuitive fashion. The current server-only authorization model will be expanded to encompass all relevant client-side entities (both "subjects" like objects, applications, hosts or users; and "targets" like system resources). The expanded model will be used to make security policy decisions on which TIE objects can access which resources (and to what degree), based on who vouches for the object, what resources the object needs, who is running the object (either host or user) and some aspects of its past migration. Issues of ancestry and transitivity will have to be considered. For instance, if site S1 normally trusts an object when received directly from user U at site S2, does S1 also trust that object if received via site S3?

Fundamentally, we do not plan to build presuppositions about trust into our environment, as was done in Telescript. For example, the trust model will provide opportunities for secure mobility that encompass different amounts and types of trust. Mobile objects will have the option of attempting to authenticate the places they move to, and determining from the information returned whether the place is sufficiently trustworthy for them to move there. We believe the confidentiality and integrity of remote computations are as important as those of the remote environment. A user must be able to trust that his data were not corrupted or improperly disclosed while residing on a remote site.

In addition to work on the trust model, we will consider the implications of a mobile object's security policy moving with that object. This will involve encapsulating the policy in a form suitable for movement even outside its home domain. The policy will need to be extended to consider which hosts the object may move to. Some objects may move only to well-known places (e.g., intranet objects). Others may move to any site, even unknown ones. In that case, the object's own authorization information can act locally only as a hint; it cannot be enforced.

On individual nodes, key security problems consist of authorizing access to local resources (given that we can be sure about the source of RMI's and mobile objects), and preventing untrusted code from damaging the system or acting as a Trojan horse. Currently, the Java environment attacks problems of this sort by providing a safe, object-oriented environment for programs, and using run-time checking to determine that compiled programs continue to meet safety constraints. There is a simple trust model for code; locally generated code is trusted, remote code is untrusted. Untrusted code faces severe restrictions on its actions. This trust model is both unnecessarily restrictive and insufficient to prevent harm [Dean96]. Applets can mount a variety of denial-of-service and GUI-based spoofing attacks on system resources. There is no support for a "trusted path" between user and code, no comprehensive mediation of access to different name spaces, and no protection from overconsumption of resources.

The dilemma is how to provide distributed applications with sufficient access to resources like memory, CPU cycles, or I/O bandwidth while not compromising the integrity of other applications, the browser, or the host system. We will limit the risk posed by untrusted code by executing it within a Secure Application Execution Framework (SAEF), which presents a limited view of each visible name space (similar in spirit to Unix chroot) and enforces restrictions (similar to quotas) on usage of resources.

The framework also denies untrusted objects access to various system services and information such as user identity. For users of a local computing environment, the SAEF will provide the equivalent of a trusted-path windowing environment [Epst92] so that the user knows which output is coming from a trusted object and which is coming from untrusted objects. This will prevent malicious applications from mounting "snooping," "spoofing," or GUI denial of service attacks.

The SAEF will be more systematic than the current Java design but more lightweight than a pure virtual machine monitor approach [Karg90]. For instance, as provided in our own implementation of a Java runtime environment (currently under development at the OSF RI [Loep96]), namespace mediation by the SAEF will occur purely at download or compile time, using the notion of Domain Type Enforcement [Boeb85] integrated with the runtime security manager and with language class naming. Thus mediation will incur no runtime overhead.

5. Related Work

TIE has goals similar to the recent work of Tanenbaum's group on wide-area distributed systems [Stee95]. Some differences are that we have chosen to use Java and the Web as a starting point, to produce an easily deployed portable system, and have attempted to address security concerns in more detail.

The management of distributed shared objects was inspired by the work on DISOM [Gued93], an application-level distributed shared object environment based on flexible, integrated locking and caching schemes.

Recent programming languages, such as Java [Sun94], Tcl/Tk [Oust94] and Telescript [Gene95], have enabled this work by expanding the possible domains of distributed computing to include heterogeneous wide-area environments, with a safe and easy software deployment path.

Mobile objects have been investigated for many years, yet their potential has only become realizable with the widespread use of the Web and portable/scripting languages. A group at IBM addresses security, interfaces between agents and servers and languages for building agents [Ches95]. The Tacoma project, jointly conducted at Tromso University in Norway and Cornell University, USA, addresses mobile agents based on Tcl/Tk, on top of Unix and Horus system [Joha95]. The most appealing system is a commercial product, named Telescript, from General Magic, offering technology for electronic commerce [Gene95]. We will enhance Java to support mobile objects in a manner similar to Telescript [Whit94]. Unlike Telescript, however, we will not build assumptions of trust into our environment.

Historically high-trust systems have employed virtual machine architectures to make strong guarantees about the partitioning of untrusted applications. The Java approach is more lightweight and *ad hoc*. Our strategy of providing a Secure Application Execution Framework is more systematic than the current Java design but more lightweight than a pure virtual-machine approach.

There are obvious parallels between our "middleware" approach to distributing objects and existing middleware products, e.g. DCE [Rose92] and CORBA [Sole95], that support distributed applications. However, the granularity and target domains of the two are very different. DCE is aimed at what we might call "intranets," networks (or sets of networks) on which the default behavior of hosts, users and applications over time is mostly assumed to be orderly and stable, whereas CORBA is an all-inclusive specification for heterogeneous object-oriented applications. Both systems are much more complex and difficult to use than TIE, and neither system addresses distributed object caching or mobile objects and agents nor are they integrated with the Web to the same extent.

6. Conclusions

We have shown how distributed and mobile objects have a real chance to advance the state of Internet civilization, and have explored various pieces of infrastructure needed to enable them to prosper. We believe that, by integrating scalable, hierarchical object naming and caching with current Java-based environments, and by adding object persistence, our TIE project makes the widespread use of such objects much more achievable. To make their use safe, as well, we have also presented key enhancements needed in single-node and inter-node security.

From a pragmatic standpoint, we believe that the best way forward is not to innovate at the level of local-system technologies, such as extensible microkernels, but to concentrate on applying and integrating existing partial solutions to the overall problems created by the ever-increasing (and ever more desirable) use of the Web. Hence we do not propose to create new single-node OS technology, nor new portable application programming environments, but rather to build powerful distributed OS abstractions as middleware atop already-available ones.

References

[Bern94] Berners-Lee, T., R. Cailliau, A. Luotonen, H.F. Nielsen, A. Secret, "The World Wide Web", *Communications of the ACM*, August 1994, 37(8):76-82.

[Bers93] Bershad, B., M.J. Zekauskas, W.A. Sawdon, "The Midway Distributed Shared memory System", *Proceedings of the 1993 COMPCON Conference*, pp 528-537, 1993.

[Boeb85] Boebert, W.E., R.Y. Kain, "A Practical Alternative to Hierarchical Integrity Policies", *Proceedings of the 8th National Computer Security Conference*, Gaithersburg, MD, pp. 18-27, 1985.

[Card95] Cardelli, L., "A Language with Distributed Scope", *Proceedings of the 22nd Annual ACM Symposium on Principles of Programming Languages*, pp. 286-297, 1995.

[Chan96] Chankhunthod, A., P. Danzig, C. Neerdaels, M. F. Schwartz, K.J. Worrell, "A Hierarchical Internet Object Cache", *Proceedings of the 1996 USENIX Annual Technical Conference*, San Diego, 1996.

[Ches95] Chess, D., et al., “Itinerant Agents for Mobile Computing”, *IEEE Personal Communications Magazine*, October 1995.

[Dean96] Dean, D., E. Felten, D. Wallach, “Java Security: From HotJava to Netscape and Beyond,” *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pp. 190-199, 1996.

[Epst92] Epstein, J., et al. “Evolution of a Trusted B3 Window System Prototype”, *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pp. 226-239. 1992

[Gene95] General Magic, “An Introduction to Safety and Security in Telescript”, *http://www.genmagic.coms/Develop/Telescript*.

[Gibs84] Gibson, W., “Neuromancer”, ISBN 0-441-56959-5, Ace Books, New York, 1984.

[Gosl95] Gosling, “The Java Language: A White Paper”, Sun Microsystems, 1994.

[Gued93] Guedes, P., M. Castro, “Distributed Shared Object Memory”, *Proceedings of the Fourth Workshop on Workstation Operating Systems (WWOS-IV)*, 1993.

[Hami93] Hamilton, G., M.L. Powell, J. Mitchell, “Subcontract: A Flexible Base for Distributed Programming”, *Proceedings of the 14th SOSP*, Asheville, NC, 1993, pp. 69-79.

[Joha95] Johansen, D., R. van Renesse, F. Schneider, “Operating System Support for Mobile Agents”, *Proceedings of the 5th Workshop on Hot Topics in Operating Systems*, 1995.

[Karg90] Karger, P., M.E. Zurko, D. Bonin, A. Mason, C. Kahn, “A VMM Security Kernel for the VAX Architecture”, *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pp. 2-19, 1990.

[Loep96] Loepere, K., “Javelin - A Java-Compliant, High Assurance, Real-time Runtime Environment”, *http://www.osf.org/www/java/javelin*.

[Milo93] Milojicic, D., et al., “Task Migration on Top of the Mach Microkernel”, *Proceedings of the third USENIX Mach Symposium*, April 1993 pp 273-290.

[O'Mal92] O'Malley, S. O. and L. Peterson, “A Dynamic Network Architecture”, *ACM Transactions on Computer Systems*, 10(2):110-143, 1992.

[Oust94] Ousterhout, J. K., *Tcl and the Tk Toolkit*, Addison-Wesley, 1994.

[Pyth95] Python home page: *http://www.ccs.neu.edu/Cobwebs/Python.*

[Rigg96] Riggs, R., J. Waldo, A. Wollrath, K. Bharut, “Pickling State in the Java System,” *Proceedings of the 2d Usenix Conference on Object-Oriented Technology and Systems (COOTS)*, pp. 241-250, 1996.

[Rive78] Rivest, R. L., et al., “A Method for Obtaining Digital Signatures and Public-Key Crypto Systems”, *Communications of the ACM*, 21(2):120-126, 1978.

[Rose92] Rosenberry, W., et al., “Understanding DCE”, O’Reilly & Associates, Inc., 1992.

[Rous96] Rousch, E., R. Campbell, “Fast Dynamic Process Migration”, to be presented at the *International Conference on Distributed Computing Systems*, 1996.

[Schi93] Schill, A., M. Mock, “DC++: Distributed Object Oriented System Support on top of OSF DCE”, *Distributed Systems Engineering*, 1(2):112-125, 1993.

[Smit96] Smith, P., N. Hutchinson, “Heterogeneous Process Migration: The Tui System”, TR,96-04, University of British Columbia, 1996.

[Sole95] Soley, R., “Object Management Architecture Guide”, Third Edition, Wiley, 1995.

[Sper96] Spere, S., “SolidOak - A Standard for Digitally-Signed Java Classes,” *http://sunsite.unc.edu/ses/SolidOak.html*.

[Stee95] M. van Steen, et al., “Towards Object-based Wide Area Distributed Systems”, *Proceedings of the International Workshop on Object Orientation in Operating Systems*, pp. 224-227, 1995.

[Sun94] “The Java Language: A White Paper”, Sun Microsystems, 1994.

[Thei85] Theimer, M., et al., “Preemptable Remote Execution Facilities for the V System”, *Proceedings of the 10th SOSP*, pp. 2-12, 1985.

[Woll96] Wollrath, A., R. Riggs, J. Waldo, “A Distributed Object Model for the Java System,” *Proceedings of the 2d Usenix Conference on Object-Oriented Technology and Systems (COOTS)*, pp. 219-231, 1996.

[Whit94] White, J., “Telescript Technology: An Introduction to the Language”, General Magic White Paper, General Magic, 1995.

[Xope95] X/Open Company Limited, “The X/Open Federated Naming Specification”, CAE Specification C403 ISBN 1-85912-052-0, 346p., Reading, UK, 1995.

[Zurk96] Zurko, M.E., R. Simon, “Adage - Authorization for Distributed Applications and Groups”, *http://www.osf.org/www/adage*.