Towards a World-Wide Civilization of Objects

Michael Condict, Dejan Milojicic,
Franklin Reynolds, Don Bolinger

{**condict**,**dejan**,**bolinger**}**@osf.org**,**fdr@cambridge.apple.com**

The Open Group Research Institute
11 Cambridge Center
Cambridge, MA 02142

(take a look at ~dejan/fr/dmo/coots.fr for how to arrange the naming, will make your life easier)

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

Michael, fill this in. You may transfer some material from the following section, but it may be worthwhile also to retain a "Background" section (probably without "Position Statement" which really belongs to Introduction).

2. Background and Position Statement

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 [Reference?]. The question is how to speed that evolutionary process.

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. Its real innovation, however, will be the support of mobile objects and agents, together with the enhanced security they require. To promote more effective use of scarce and unpredictable network bandwidth, it will support easily designed and composed custom network protocols. Finally, the TIE will provide a high level of configurability.

The TIE will be implemented 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 the 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.

3. The Object Framework

The 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 net-

work. Alternatively, stationary objects can be operated on by remote method invocation (loosely referred to as 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 RPC message is sent to the object, which is then operated on remotely.

The implementation language of the TIE is Java, and its objects are a direct extension of Java objects. The main extensions are that object state is distributed and that objects can migrate and can be *active* (see below). These extensions can be implemented in unenhanced Java, without modifying today's Java run-time environments.

To support RPC, there is an IDL compiler, which, for any class description, generates a client-side and a server-side stub function (in Java) for each operation in that class. (Given its properties, Java itself may be usable as an IDL.) In addition to transmitting remote method invocations across the Internet, the stubs also provide hooks to allow users to extend the semantics of RPC to include such functionality as security and replication.

To support distributed shared objects, there is a hierarchical, persistent object cache. To increase its usefulness for mobile objects, the cache will be security-enhanced, as we explain below under Security. 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 node will have a cache, which may be filled by obtaining copies of objects from a per-LAN cache, which in turn may be connected to a per-organization cache, etc. 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.

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 (see Custom Network Protocols, below).

The safe use of distributed shared objects requires operations that ensure mutually exclusive access. The TIE provides locking functions integrated with the object cache. The particular cache coherency and locking policies used by the TIE can be replaced by the user. The system uses entry consistency [Bers93] for cache coherency, by default. 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 adjustable grace period) to find out whether they still hold a valid copy of the object.

To allow users to locate and invoke remote objects using human-readable identifiers, the TIE incorporates a world-wide name service. Instead of attempting to define a new syntax for names, we note that the Web already has a workable convention, the URL, which we will continue to use as the default naming model. The configurability of the TIE will also allow its federated naming service to be integrated with other existing federated naming systems, e.g. CORBA and XFN (X/Open Federated Naming). Each node of the TIE will have a local name server, linked into a world-wide hierarchy of name servers.

The job of the name service in the TIE is to provide an object reference, given the name of an object. 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, of course, this reference may need to be updated.)

Objects in the 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 (see Mobile Objects and Agents, below). 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.

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. Persistent objects may not have a name, but have a persistent object reference. An example of a persistent, named object is a Web document named by a URL.

The TIE is very 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 RPC and/or the method to send the RPC 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 the TIE is its ability to arrange that when an object and its invoker are in the same address space, the overhead of RPC 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 RPC's over the network.

In the following sections, we describe particular components and features of the TIE in more detail.

4. Mobile Objects and Agents

Mobile Objects and Agents (MOA) serve a number of useful purposes in object civilization. Aside from the obvious example of electronic commerce with mobile agents that travel the Web acting on behalf of users, MOA left behind in the network can help overcome the problems of mobile users who are infrequently connected. MOA can reduce network traffic because the agent moves towards the source of communication, localizing it. Mobile objects and agents are not always the sole means to these goals, but they can provide unified support that would otherwise require multiple mechanisms [Ches94].

The problems and challenges in implementing mobile objects are similar to those for distributed objects. In both cases, objects move from one site to another, and a modified object may return to the original site. The main difference is the security implications of active objects. Passive distributed objects pose few security problems. Cache coherency does not require taking back a modified object from any site unless the site had the authority to modify the object. But a mobile object may return to the server from an untrusted site, or migrate from one client to another who does not trust the first. When this happens, the object, even if allegedly unmodified, must be checked for validity. If it is an active mobile object, it must be treated like any code downloaded from an untrusted site, and confined to execute within a secure framework called a SAEF (see Security, below). (This is original text)

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. Most of the work on mobility appeared in the form of process migration [Thei85, Bara86, Zaya88, Doug91], programming language object mobility [Jul88], and middleware object mobility [Shap89, Schi93, Amar92]. Despite the overwhelming interest, which continues up to the present [Stee95, Smit96, Rous96], object mobility has never really became widespread.

A frequently-stated belief about mobility is that it is too complex to implement, is costly to use, imposes limitations on the object model (has limited transparency) and is limited to homogeneous nodes. However, if we analyze implementations, we can see that technical solutions have been found for each of these problems. Migration has been supported as the 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, Zaya88]; transparency has been achieved at different levels, from limited to complete; and recent work demonstrates improvements in heterogeneity [Stee85, Smit96].

Instead, we believe that the true impediment is the lack of infrastructure. First, there has been no widely-available operating system that supported object mobility. Second, mobility has not been a requirement for users; solutions that do not require mobility (remote invocation) perform less well in some cases, but they are able to meet expectations. Finally, sociological factors have been important in limiting the deployment of mobile objects. The workstation model, where each node belongs to a user, discourages mobility. Users are reluctant to allow remote objects to visit "their" machines.

The Web environment changes these conditions, both technical and sociological. Instead of a processor pool and workstation model, we have a Web environment with computers connected as interfaces to a "network-as-computer" model. Objects are relieved of UNIX functionality, with limited requirements for transparency (user-specific solutions are preferred over general highly- transparent ones); performance is dominated by network latency and therefore object state transfer is not as visible as it is on a local area network; heterogeneity is enabled via programming languages, such as Java [Gosl95] or Tcl/Tk [Oust95].

In contrast to a distributed operating system, the Web has become ubiquitous, motivating would-be participants to adapt to it. In a wide area network, costs become higher for remote access and solutions with remote invocation are not competitive with solutions based on mobility. Finally, the sociological problem of "allowing access to my computer" is also lessened. Users are ready to allow applets to be downloaded. Safety and some partial security are provided. Nevertheless, the Web raises some other problems. While operating systems already had support for naming and general-purpose security, in a Web environment this needs to be added.

We shall extend the notion of applets to encompass mobile objects and agents. Mobility should be supported as Telescript does with its "go" utility, except that in contrast to Telescript's closed environment, our mobile agents will be applicable to the Web. Some key issues in object mobility are checkpointing mechanism, naming, handling communication channels, an object model, object server environment, and auditing and controlling objects.

The migration mechanism addresses the problem of how to migrate an object from node A to node B. In current Java-enabled browsers it is only possible to download, once, from server to client. In theory this would suffice for a two step algorithm (going 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). XXXX ref Java pickling paper.

An open problem is how to checkpoint and maintain the intermediate state accumulated on previous nodes. The most limited transparency is to 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.

Naming of objects needs to be more dynamic than is possible using the existing Web mechanisms, such as URLs. Even URNs are probably inadequate because mobility requires relatively frequent updates of location information. An obvious improvement would be a hierarchical naming scheme, where objects are named within ma-

chine, cluster, intranet, and finally world-wide. We must be able to integrate our naming scheme with external standards. XFN is a good candidate for integration.

Communication channels that an object has opened on one node need to be reopened after it is moved to another node. The servers and agents on other nodes need not recognize the act of migration. Different approaches provide varying degrees of transparency. The first possibility is for an object to close all connections before migration, reopening them afterward. To prevent the loss of messages during migration, a proxy object is typically left on the original node that forwards incoming messages to the new object location. 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 object model that we propose supports negotiation between agents and the sites they visit using primitives such as *move*, *meet*, *fork*, *stop*, *resume*, etc. We will use a CORBA-like IDL. (Java already supports tools that provide CORBA connectivity, and Java-compliant IDLs.) 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*. We will offer an object browser to enable viewing agents, listing agents by node, etc.

Security is a major issue on the Web, and has not been adequately addressed for mobility. We shall extend security with digital signatures and more sophisticated authorization tools, such as by applying and extending OSF RI's Adage project. Some of the basic support may be soon available by Java environment; Sun has announced the support for digital signatures and key management. Nevertheless, the hardest part will be how to apply the security in an intelligent manner and how to integrate it with the existing standard tools such as Web browsers. XXX reference solid oak

5. 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 RPC and the identity of objects migrating between nodes, and the need for capable but flexible distributed authorization. The TIE will address authentication and confidentiality problems with standard methods such as public key cryptography and digital signatures [Rive78]. 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.

Many issues around the deployment and use of signed objects need to be resolved before they can be used in our distributed middleware. The simplistic approach of having trusted compilers signing objects seems impractical. For a compiler to sign the object it produces, it must itself have a private key. If all instances of a compiler were trusted, then anyone with access to one would have access to the private key. A more likely approach is for objects to be signed by a person responsible for them (e.g., an author or team leader). We will also need an infrastructure for producing and disseminating public/private key pairs.

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 TOG 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 RPC'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.

Higher-level security concerns are handled by Java applications such as the HotJava browser. The browser has 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. 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 the Javelin environment (a freely-available replacement for Sun's Java runtime under development at TOG 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.

Finally, as we noted above under The Object Framework, to improve performance we view it as essential to be able to cache objects, even (or especially) mobile objects. The challenge, particularly for mobile objects, is to do so in a secure manner. How can an application determine when it is safe to use such a cached object? We propose to apply digital signatures to "sign" objects. When an object is initially dereferenced and discovered in the cache, its signature can be used to determine whether it is an unmodified version of the intended object. Naturally, this secure object cache will be integrated with the SAEF.

6. Custom Network Protocols

Though the standard Internet protocols have been indispensable to the success of the Internet, their weaknesses have lead to the creation of many more specialized protocols. However, the prevalence of the standard protocols, and the difficulty of deploying replacements or extensions, has made the use of new protocols problematic, even when clearly technically warranted.

As an example, the TIE provides default cache update functions to pull objects into the distributed object cache and to flush a modified object back to the server, which simply copy the state of the object across the network using a simple protocol such as HTTP. It is often possible to construct much more efficient network protocols for transmitting object state, given knowledge of the semantics of objects. For example, an MPEG stream can be viewed as an efficient way to keep a distributed object (a constantly changing image) consistent, since it attempts to transmit only the changes in the images. And it can be layered on top of an unreliable transport-level protocol, since we don't want to retransmit lost data (it will be out of date).

In response to this, we have developed a toolkit for the development of protocols, called CORDS (Communicating Objects for Real-Time, Dependable Systems), which is derived from the xKernel [O'Mal92]. CORDS provides protocol objects that can be composed in asymmetric, acyclic graphs as well as the more restrictive protocol stack. Porting our CORDS technology to a Java-based secure, distributed object technology such as the TIE will greatly simplify the problem of deploying application-specific protocols.

7. Related Work

The 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 [Genm95]. 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 VM 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. DCE naming supports location transparency, but migration is difficult.

CORBA is an ambitious set of standards for an object-oriented programming environment that is independent of any particular architecture or OS. In the most recent version (2.0) different implementations can interoperate across the Internet. There are several important differences with our work. First, the TIE is intended to be much simpler and easier to use. We are basing the system on one programming language (Java), allowing optimizations and functionality not possible if number of programming languages and legacy servers must be supported. Second, the use of distributed shared objects and mobile objects is central to the TIE, whereas CORBA does not address the issue. Finally, unlike the TIE, CORBA is not integrated with the WWW, which may affect the ease of its world-wide deployment.

In any case, the configurability of the TIE will allow it to connect to DCE and CORBA services (especially naming services) when and where it is later found to be useful.

Conclusion

Michael, fill this in

References

[Amar92] Amaral, P., et al., "Transparent Object Migration in COOL-2", Proceedings of the ECOOP, June 1992.

[Bara86] Barak, A. and Shiloh, A. "A Distributed Load-Balancing Policy for a Multicomputer", Software-Practice and Experience, Sep 1985, 901-913.

[Bern94] Berners-Lee, T., R. Cailliau, A. Luotonen, H.F. Nielsen, A. Secret, "The World Wide Web", Communications of the ACM, August 1994, vol 37, no 8, pp 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", Proc. 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", 1996 USENIX Annual Technical Conference, San Diego, 1996.

[Ches94] Chess, D.M., et al., "Mobile Agents: Are They a Good Idea?", IBM Research Report, RC 19887, October 1994.

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

[Doug91] Douglis, F. and Ousterhout, J., "Transparent process migration: Design alternatives and the Sprite Implementation", Software-Practice and Experience, August 1991, 21(8):757-785.

[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

[Fini95] Finin, T., et al., "KQML as an Agent Communication Language", CIKM'95, Intelligent Information Agents Workshop, 1995.

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

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

[Gray95] Gray, R., "Agent Tcl: A Transportable Agent System", CIKM'95, Intelligent Information Agents Workshop, 1995.

[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.

[Jul88] Jul, E., et al., "Fine-Grained Mobility in the Emerald System", ACM Transactions on Computer Systems, February 1988, 6(1):109-133.

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

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

[O'Mal92] O'Malley, S. O. and L. Peterson, "A Dynamic Network Architecture", *ACM Transactions on Computer Systems*, Vol 10, no 2, pp 110-143, May 1992.

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

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

[Pein95] Peine, H., "Ara - Agents for Remote Actions", Presentation, Distributed Systems Group, University of Kaiserslautern, Germany.

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

[Rive78] Rivest, R. L., et al., "A Method for Obtaining Digital Signatures and Public-Key Crypto Systems", *Communications of the ACM*, vol 21, no 2, February 1978, pp. 120-126.

[Rose92] Rosenberry, W., et al., "Understanding DCE", O'Reilly & Associates, Inc., 1992.

[Rous96] Rousch, E., Campbell, R., "Fast Dynamic Process Migration", to be presented at the Intl. Conf. on Distributed Computing Systems, 1996.

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

[Shap89] Shapiro, M., et al., "Persistence and Migration for C++ Objects", ECOOP '89 European Conference on Object-Oriented Programming, July 1989.

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

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

[Sowa92] Knowledge-Based Systems, vol. 5, no. 3, 1992, ed. by John F. Sowa.

[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", Proc. of the 10th ACM SOSP, December 1985, pages 2-12.

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

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

[Zaya88] Zayas, E., "Attacking the Process Migration Bottleneck", Proc. of the 11th SOSP, pages 13-24, November 1987.

General Comments by Dejan

1. I spoke already to Don about the use of TIE and referencing to Javelin work (the way it is referenced). I have strong opinion on this and wouldn't approach it the way Don did. However, since I am not the only author (and not the first one), I am more flexible on this issue, and can live with it (although I'd still urge you to reconsider it, or soften it at least). The same goes for TOG. Ira is too dynamic that we should blindly follow his changes in names of projects and organization. I would simply stick to OSF RI instead of TOG RI (Ira himself mentioned that TOG RI may change again soon); and ISE instead of TIE.
2. We need a usual introduction, since this is already a typical paper instead of the short position paper. I am pretty sure that all other paper will have similar format. This is definitely true if are aiming at later publication in SIGOPS (in addition to proceedings of the workshop).
3. Similarly, and even more important, we need a proper conclusion.
4. We need some good figures. This may be hard, but nevertheless you should consider adding some. You can possibly pick up some from my presentation at COOTS workshop.
5. When speaking about thick/thin proxies, you should also mention intermediate ones. In general, you should refer to them as custom proxies.
6. Change all type of cross referencing of the form (see below, or see Custom Network Protocols, etc.) into (See Section 6).
7. You have to mention RMI and reference their paper at COOTS.
8. In general you have too much forward referencing. This is not good for paper. Get rid of it.
9. When you mention naming in CORBA and XFN, you need references + be more explicit on what kind of naming you want to introduce.
10. Turn VM into Virtual Machine. Also add reference to Mez' paper.