Mobility Support in OBIWAN[*]

Luís Veiga Paulo Ferreira
{luis.veiga,paulo.ferreira}@inesc.pt
INESC/IST, Rua Alves Redol N^o 9, Lisboa, Portugal
http://www.gsd.inesc.pt phone: 351 21 3100292

Abstract

The need for sharing is well known in a large number of distributed collaborative applications in a mobile environment. For this purpose, we have been developing a platform called OBIWAN[1] that: i) allows the application programmer to decide the mechanism by which objects should be invoked, remote method invocation or invocation on a local replica, ii) allows incremental (on-demand) replication of large object graphs, iii) provides hooks for the application programmer to implement a set of application specific properties such as transactional support or updates dissemination, iv) supports the migration of execution flows allowing the implementation of mobile agents, and v) supports the concept of a computing dynamic horizon in which resources in a broader sense (memory, disks, printers, internet access, data and even code) can be found in other neighbor devices and used accordingly.

1 Introduction

There is a clear need for data sharing and collaboration support in a large number of applications in different domains. In OBIWAN, we focus on applications in the area of co-operative work within virtual organizations; for example, a virtual enterprise grouping several companies from different countries, a virtual marketplace, a widely distributed software development team, a distributed game involving people anywhere in the world, etc.

This need for information sharing is increasing along two main axis: wide area (i.e., across the Internet) and mobility (i.e., portable computers, webpads, personal digital assistants, smart cellular phones, etc.). As a matter of fact, besides the growing number of desktop computers connected to the Internet, there are other devices, generally called information appliances (info-appliances, for short) that are gaining enormous popularity; personal digital assistants (PDAs) are just one of them.

The role of these info-appliances, currently handling agendas, calendars, etc. will certainly grow as more computing power and communications capability can be included [17, 18]. In particular, the foreseen increase of bandwidth in wireless communication makes the connection of these info-appliances to the Internet a reality [15].

We envisage a general scenario in which a user wants to access data using a PC in his office, using a laptop while in the airport or in the hotel, using a PDA in a taxi, etc. The user wants to live in this "data ubiquitous world" with no other concern besides doing his own work and, as much as possible, to keep on working in spite of any system problem that may occur (e.g. network partitions).

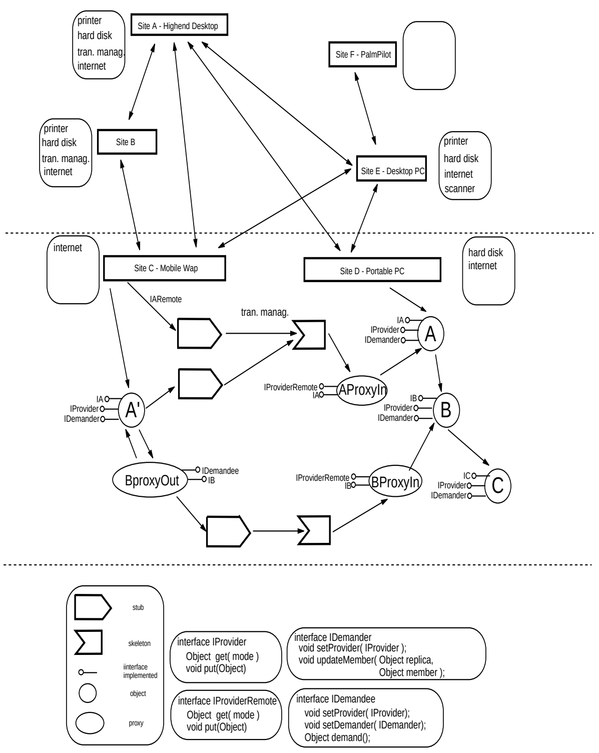
So, there is a constant need to access shared data no matter where you are and the info-appliance you use, and users want the same degree of responsiveness and performance as in a fully high-bandwidth low-latency wired connected environment. Sometimes these requirements may be impossible to fulfill but the system should be able to minimize the number of such occurrences.

As a matter of fact, mobile computing is characterized by significant and rapid changes in its supporting infrastructure and, in particular, in the quality of service available from the underlying communication channels; wireless links provide lower bandwidth, possibly higher error rates than wired networks, and periods of disconnection and intermittent or variable connectivity may occur.

For example, if accessing data on some remote machine is not possible for some reason, the application should not stop working; instead, it should, at least, automatically propose the user an alternative access to such data from another machine, even if such data is not up to date.

Another example is related to network partitions. While these are rare in stationary local area networks, they occur in greater number in wide area mobile networks. Most applications consider them to be failures

[*]This work was supported by Microsoft Research.
[1]OBIWAN stands for **O**bject **B**roker **I**nfrastructure for **W**ide **A**rea **N**etworks.



```
interface IProvider
   Object  get( mode )
   void put(Object)
```

```
interface IDemander
  void setProvider( IProvider );
  void updateMember( Object replica,
                              Object member );
```

```
interface IProviderRemote
   Object  get( mode )
   void put(Object)
```

```
interface IDemandee
   void setProvider( IProvider);
   void setDemander( IDemander);
   Object demand();
```

Figure 1: Main structures of OBIWAN.

that are exposed to users. In the mobile environment, applications will face frequent, lengthy network partitions. Some of these partitions will be involuntary (e.g., due to a lack of network coverage) while others will be voluntary (e.g., due to a high dollar cost). Mobile applications should handle such partitions gracefully and as transparently as possible. In addition, users should be able, as far as possible, to continue working as if the network was still available. In particular, users should be able to modify local copies of global data.

The rest of the paper is organized as follows. In the next section we describe the most important aspects of the OBIWAN platform: its architecture, replication and migration support. In Section 3 we describe the main steps for programming applications running on top of OBIWAN. Then, in Sections 4 and 5 we present the implementation and some performance results. Sections 6, 7 and 8 describe some related work, future OBIWAN improvements and conlusions, respectively.

2 OBIWAN

The overall objective of the OBIWAN project is to design and implement a system that: (i) is well suited to support distributed applications with strong sharing needs in a mobile environment, and (ii) facilitates application development by releasing programmers from the need to handle complex system issues such as fault-tolerance, memory management, etc., while providing the right level of abstraction and functionality to deal with unexpected situations.

We believe that the notion of a generic object broker infrastructure provides the means to the kind of sharing described above. Intuitively, to describe our object broker infrastructure, we can say that OBIWAN supports applications that manipulate an ocean of objects; these objects are scattered over a variety of locations and info-appliances, can flow among such appliances, and contain innumerable references connecting them.

OBIWAN provides support for the following: i) allows the application programmer to decide the mechanism by which objects should be invoked, remote method invocation or invocation on a local replica, ii) allows incremental (on-demand) replication of large object graphs, iii) provides hooks for the application programmer to implement a set of application specific properties such as transactional support or updates dissemination, iv) supports the migration of execution flows allowing the implementation of mobile agents, and v) supports the concept of a computing dynamic horizon in which resources in a broader sense (memory, disks, printers, internet access, data and even code) can be found in other neighbor devices and used accordingly.

We believe that all this functionality allows the application programmer to deal with situations that frequently occur in a (mobile) wide-area network, such as disconnections and slow links.

2.1 Architecture

Figure 1 illustrates the architecture of OBIWAN. It shows six sites (A....F) with arrows among them meaning that, at some instant, they know each other. Thus, they can exchange information and, for example, use the resources of each other. We present with more detail the objects (and the references among them) for sites C and D. Close to each site we also present information describing the site resources such as hard disk, transaction manager, wired connection to the internet, etc.

Stubs and skeletons are created by the underlying virtual machine. (Currently, both .Net and Java are supported). Objects A, B and C are created by the programmer; their replicas, A', B', and C' are created upon the programmer's request. All other objects, i.e. proxies-in and proxies-out, are part of the OBIWAN platform and are transparent to the programmer. Figure 1 also shows, for each object and proxy, the interfaces implemented:

- IA, IB and IC: these are the remote interfaces of objects A, B and C, respectively, designed by the programmer; they define the methods that can be remotely invoked on these objects.

- IProvider: interface with methods that support the creation and update of replicas; method `get` results in the creation of a replica and method `put` is invoked when a replica is sent back to the process where it came from in order to update its master replica.
- IDemander and IDemandee: methods that support the incremental replication of an object's graph; in other words, the implementation of these interfaces allow OBIWAN to detect an object fault and to service it accordingly.
- IProviderRemote: remote interface that inherits from IProvider so that its methods can be invoked remotely.

2.2 Replication

OBIWAN provides support for objects in the sense that they can be invoked either remotely, via remote method invocation (RMI) [1, 21], or locally via local method invocation (LMI) based on a replication mechanism that brings objects to the info-appliance where an application is running [10].

This replication mechanism is incremental in the sense that only those objects that are really needed are effectively replicated (not the whole graph which can be very large); the application does not have to wait for the replication of every object it needs, as this is done in parallel in the background (with a pre-fetching approach).

The flexibility of the invocation mechanism allows the application programmer to develop his application so that it resists to network failures, and allows the user to work disconnected from the network (either voluntary or not). As a matter of fact, as long as those objects needed by an application (or an agent) are locally accessible, there is no need to be connected to the network. In addition, by replicating objects in the info-appliance where an application using them is running, the overall performance can be improved w.r.t. an approach in which objects are always invoked via RMI.

Finally, note that the programmer can easily replace, in run-time, remote by local invocations on replicas, thus improving the performance of his application and its adaptability.

2.3 Incremental Replication

We now explain in detail the steps involved in the incremental replication of the graph presented in Figure 1. Consider the initial situation in which there is no replica of the graph A-B-C in site C. Then, the application running in site C requests A' by invoking method AProxyIn.get; this method simply invokes A.get. Then, this method executes the following:

1. create A' in site D;
2. for each reference A holds (only to B in this case) create the corresponding ProxyIn objects (only BProxyIn in this case) in site D; in the constructor of BProxyIn, set an internal reference pointing to B;
3. create a ProxyOut object for each ProxyIn created in the previous step (only BProxyOut in this case) in site D;
4. set the internal reference of A' (of type IB) so that it points to BProxyOut;
5. invoke BProxyOut.setProvider(BProxyIn) so that BProxyOut points to BProxyIn;
6. invoke BProxyOut.setDemander(A') so that BProxyOut also points to A'; return A' to site C.

Thus, AProxyIn.get terminates simply by returning A'. As a result, A' and BProxyOut are automatically serialized by the underlying virtual machine and sent to site C. Thus, in site C, object A' points to BproxyOut (that stands for B').

Later, the code in A' may invoke any method that is part of the interface IB, exported by B, on BProxyOut (that A' sees as being B'). For transparency, this requires the system to support a kind of "object fault" mechanism as described now. All IB methods in BProxyOut simply invoke its demand method BProxyOut.demand that runs as follows:

1. invokes method BProxyIn.get (BProxyIn is BProxyOut's provider);
2. BProxyIn.get invokes B.get that will proceed in a similar way as explained previously for A.get: creates B', CProxyOut, CProxyIn and sets the references between them; once this method terminates, B', BProxyOut and CProxyOut are all in site C, CProxyIn is in site D, and BProxyOut points to B'; note that A' and BProxyOut still point to each other;
3. BProxyOut invokes B'.setProvider(this.provider) so that B' also points to BProxyIn; this is needed because the application can decide to update the master replica B, by invoking method B'.put that in turn will invoke BProxyIn.put, or to refresh replica B' (method BProxyIn.get);
4. BProxyOut invokes A'.updateMember(B',this) so that A' replaces its reference to BProxyOut with a reference to B';

5. finally, BProxyOut invokes the same method on B' that was invoked initially by A' (that triggered this whole process) and returns accordingly to the application code;

6. from this moment on, BProxyOut is no longer reachable in site C and will be reclaimed by the garbage collector of the underlying virtual machine.

It's important to note that, once A' and B' are in site C, further invocations from A' on B' will be normal direct invocations with no indirection at all. Obviously, later, if and when B' invokes a method on CProxyOut (standing in for C' that is not yet replicated in site C) an object fault occurs and will be solved with a set of steps similar to those previously described.

The replication mechanism just described is very flexible in the sense that allows each object to be individually replicated. However, this has a cost that results from the creation and transference of the associated data structures (i.e., proxies). To minimize this cost OBIWAN allows an application to replicate a set of objects, i.e. a cluster.

A cluster is a set of objects that are part of a reachability graph. For example, if an application holds a list of 1000 objects, it is possible to replicate a part of the list so that only 100 objects are replicated and a single pair of proxy-in/proxy-out is effectively created and transferred between sites. Thus, the amount of objects in the cluster is determined in run-time by the application. The application specifies the depth of the partial reachability graph that it wants to replicate as a whole. So, these clusters are highly dynamic. This is an intermediate solution between: i) having the possibility of incrementally replicate each object, or ii) replicating the whole graph.

2.4 Migration

In addition to replication, OBIWAN also supports the migration of execution flows making possible the implementation of mobile agents. However, since threads' stacks are not first class objects (both in .Net and Java) the programmer must provide synchronization points in which the agent execution can be freezed, its state serialized and transferred for ulterior reactivation upon arrival on another machine.

Agents activities should be monitored for twofold security reasons. Some machines may not allow certain actions to some agents based on their origin and migration path. On the other side, agents should be able to choose among several available paths from one machine to another.

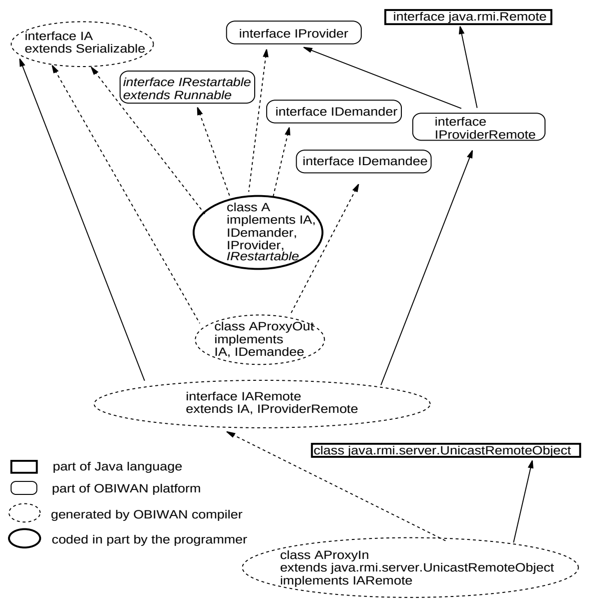


Figure 2: Interfaces and classes of OBIWAN. Inheritance is represented with a solid line; implementation is represented with a dashed line.

3 Application Development

The programming of a distributed application with OBIWAN, when compared to a standard approach based on RMI, is simpler and provides more functionality. As a matter of fact, the programmer only has to write:

- an interface specifying the methods that an object will service (e.g. interface IA), and

- a class implementing the just mentioned interface (e.g. class A).

All the rest, i.e. the code handling the invocations either via RMI or LMI (after the creation of a local replica), incremental replication, object-fault detection and serving, and migration of execution flow is automatically generated. This generation is done with a tool called `obicomp`.

Note that even the above mentioned interface (e.g. IA) can be derived from the corresponding class (e.g. A). This is important because it allows us to automatize the porting of legacy non-distributed applications to OBIWAN.

Finally, the implementation of interfaces IProvider and IDemander on objects written by the programmer (e.g. A) is automatically generated through source code insertion done by `obicomp` .

4 Implementation

The OBIWAN system runs both on top of the Java virtual machine and on top of .Net. Both environments are simple to use, and support the basic functionality required, i.e. RMI, dynamic code loading and reflection. In this section we describe the classes, interfaces and tools that constitute OBIWAN.

Suppose we want to build a distributed application with an object that can be either locally replicated or invoked via RMI. Additionally, we wanted objects to be replicated incrementally, i.e., as and only when they are needed. All the interfaces and classes involved in this example are illustrated in Figure 2. The "rectangular" interfaces and classes are part of the underlying system (Java or .Net). The "rounded rectangles" represent OBIWAN platform interfaces that are constant and therefore pre-compiled. The "dashed ellipses" represent classes and interfaces automatically generated by the `obicomp` compiler. Finally, the solid "ellipse" represents the class that the programmer must write. The programmer only has to worry with the so-called "business-logic". The implementation of interfaces IDemander, IProvider and, if so desired, interface IRestartable is automatic through source code augmentation of the class the programmer has written.

The programmer only has to write class A, the corresponding interface IA can be derived from it, and, obviously, the code of the client that invokes an instance of A. Note that the interfaces `IProvider` and `IProviderRemote` are constant, thus they do not have to be generated each time an application is written. The interface `IARemote`, and classes `AProxyOut` and `AProxyIn` are generated automatically.

To summarize, when a new application is developed the programmer does the following steps:

- write the interface `IA`;
- write the class `A`;
- run the `obicomp` tool.

The last step is to automatically generate the other interfaces and classes needed, and to extend class A implementing interfaces IProvider and IDemander.

Currently, `obicomp` uses a mix of: i) reflection mechanism to analyze classes and generate the corresponding proxies, and ii) source code insertion to augment the classes written by the programmer with the methods that implement interfaces IDemander and IProvider.

Finally, the support for the migration of execution flow is achieved simply by having class A to implement the interface IRestartable (provided by OBIWAN that derives from Runnable); OBIWAN generates automatically the code that implements IRestartable except the method run and the special implementation of interface IProvider that, besides creating and updating replicas, makes use of IRestartable methods to freeze and restart the object, prior to departure from, and after arrival to any machine; this method, run, must be written by the programmer.

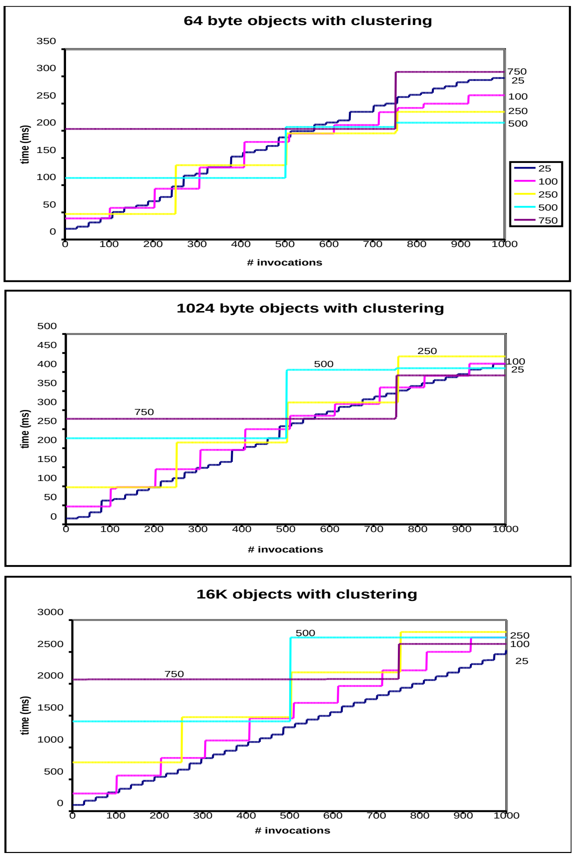


Figure 3: Incremental replication of clusters of objects.

5 Experimental Results

In this section we study the performance of incremental replication with object clustering. These results were obtained in a 100 Mb/sec local area network, connecting Pentium II and Pentium III PCs with 128 Mb of main memory each, running JDK 1.3. We already have a protoype running on .Net; preliminary results indicate a much better performance than with Java. We will have full results of the .Net implementation for the workshop.

We use several lists, each made of 1000 objects with the same size. Different lists have objects with differente sizes: 64 bytes, 1024 bytes and 16 Kbytes. The list are created in site D. This list is then replicated into another site C, in several steps, each step replicating 25, 100, 250, 500, or 750 objects. Then, the application running in site C invokes a method on each object of the list. When the object being invoked is not yet replicated the system automatically replicates the next 25, 100, 250, 500, or 750 objects. So, objects are replicated in groups, i.e. clusters with several sizes. This means that, for each of these clusters, all objects are replicated as a whole, thus there is only one proxy-in/proxy-out pair being created. Consequently, each object can not be individually updated.

The results are presented in Figure 3. Note that, in each case, the time values include the creation and transference of all the replicas along with the single corresponding proxy-out/proxy-in pairs. We can conclude that:

- for larger objects, there is a bigger gain on having small clusters;
- for any number of invocations, smaller clusters are normally better.

Finnally, it's worthy to note that the experiments done, if performed using just standard serialization mechanisms (in which all the graph is replicated at once) on computers with limited memory (e.g. PDAs), would not be feasible as the memory would be completely consumed before the graph being totaly replicated.

6 Related Work

The OBIWAN platform is related to several other systems that support distributed invocation, replication, automatic detection and resolution of object-faults. An important difference is that all these systems do not provide an integrated platform supporting all the mechanisms as OBIWAN does. This integration is an advantage to the programmer as he may decide what functionality is best adapted to his application scenario.

Javanaise [3, 11] is a platform that aims at providing support for cooperative distributed applications on the internet. In this system the application programmer develops his application as if it were for a centralized environment, i.e. with no concern about distribution. Then, the programmer configures the application to a distributed setting; this may imply minor source code modifications; a proxy generator is then used to generate indirection objects and a few system classes supporting a consistency protocol. Javanaise does not provide support for incremental replication. Javanaise clusters are defined by the programmer and are less dynamic than in OBIWAN. In other words, the frontier of the clusters in OBIWAN are defined in run-time by the application in order to improve its performance and to allow disconnected work.

There has been some effort in the context of CORBA to provide support for replicated objects [9]. There has been also similar efforts in the context of the World Wide Web [3]. However, most of this work addresses other specific issues such as group communication, replication for fault-tolerance, protocols evolution, etc. None seems to address the issue of distributed application development for networks of info-appliances with mobility in mind. OBIWAN attempts to minimize bandwidth and connection time to address this issue. With OBIWAN, the programmer has the means to make his application to decide, in run-time, if an object should be invoked via RMI or if a local replica should be created. We believe that this is a very important aspect when developing distributed applications for info-appliances given the significant and rapid changes in the quality of service of the underlying network.

The issue of object caching has been addressed by many systems. This is different from what we propose, replication: in OBIWAN objects can be replicated freely among sites. However, there are some common aspects between caching and replication. An important distributed system with object and page caching is Thor [14]. This system provides a hybrid and adaptive caching mechanism handling both pages and objects. In addition, Thor provides its own programming language. Most object-oriented databases (OODBs) [22], for example such as O_2 [7], GemStone [2], do have some kind of caching. However, they are very heavyweight, and often come with their own specialized programming language. There has been some work on object caching in CORBA as well. For example, Chockler [8] proposes an hierarchical cache system in which servers cache objects that clients will then ask for. When compared to OBIWAN, it is clear that we do allow objects to be replicated in the client and there is no hierarchical caching system.

Much work has been done regarding object-fault handling [13, 20]. However, most of it has been centered on persistent programming languages or related to adding transparent, orthogonal persistence to existing programming languages. Nevertheless, it is useful, since it introduces well-known and widely accepted designations for relevant existing techniques and/or concepts, e.g. swizzling. Our object-fault handling is done without modifying the underlying virtual machine. This makes our solution more portable.

7 Future Improvements

We provide, in OBIWAN, a series of hooks through which, a number of mostly orthogonal facilities can be provided to applications/agents. These include memory management, policies for coherence of replicas, security and privacy, interaction with other objects outside OBIWAN.

Memory is at premium in mobile environments. Although memory capacity of mobile devices increases steadily, it will always be relatively limited compared to portable and desktop computers. With this premiss in mind, some old axioms must fall or at the very least, be relaxed. Due to severe memory limitations, even live data should be reclaimed in order to provide free memory so that applications/agents can continue to function. This data, however, should not be simply discarded but instead swapped-out whenever possible, e.g. saved elsewhere in some more capable, portable or desktop computer. Naturally, this raises some old issues, like trashing, but in a new environment. Efficient policies should be developed based on a mix of adaptable behavior and application programmer's hints.

In a replicated environment with possibly long disconnected periods, coherence of replicas poses some difficulties. Pessimistic and synchronous models should be provided to maintain old applications semantic but with the unavoidable performance penalties. These should not be encouraged in the next generation applications/agents. They should be based in optimistic, mostly asynchronous models that allow computation to proceed, even in the presence of old data, and perform ulterior conciliation of data at merging time. This should be achieved in an automatic fashion for common data manipulations, with application specific treatment or even with user intervention. Several transactional models should be provided and could be combined in the same application/agent to access data with different freshness and exclusiveness requirements.

In such a complex new environment, security can no longer consist in a series of simple access control and authentication permissions for hardware, data, programs and communication media. Information flows through different, possibly scattered machines. Security related information must obviously be secured, as well. More so, security concepts should be enlarged to bear obligation policies, i.e., permissions are no longer just a function of application/agent identity and desired resources but also of past execution. Previous actions should be denied and their results relegated if they are not followed by other demanded actions. This can be achieved by using transaction rollbacks. However, this security information must travel untouched through a series of machines. In order to accomplish this, security data and application/agent logs should be successively encrypted with several machines private keys. Any machine could consult these logs to uphold security obligation policies but none of them, and more importantly the application/agent, could tamper with the information produced by others.

To leverage existing applications and document formats, some form of interaction with other objects outside OBIWAN should be provided. This can easily be incorporated in ProxyIn and ProxyOut objects that have automation code to load, save, manipulate and convert most document types, e.g. Word, Excel and PowerPoint documents [12].

7.1 Resource Discovery and Usage

Traditionally, applications' resources have always been seen as the set of hardware resources used by the application/agent from those available in the running machine or from a well-known set of neighbor machines as those in a LAN. The latter usually include printers and hard disk storage, sometimes internet access, seldom memory and processing power and rarely code.

We purpose the adaptation of the concept of known horizon to computing. Thus, we think that the resources available to an application/agent should not be restricted to those installed in the running device (computer, PDA, etc.). They should include all that are accessible within acceptable time frames (the horizon) from all devices the application/agent is aware of. Proximity-triggered notification should be used to frequently update current horizon definition in each device.

Furthermore, resources should be considered in a broader sense and include as much computation elements as possible. They should include CPU power, specific code in the from of services, extended memory, communication media, etc.

Whenever required hardware becomes available, logged application/agent requests to it should be activated. Additionally, every time fresh data or more recent and sophisticated code comes near, they should be transparently acquired for improved results and functionality.

This relaying-based access to resources and propagation of computation results raises new issues, namely security ones, that we want to pursue our research on. Applications/agents and the runtime should be able to monitor possible vulnerabilities as malicious resources usage, un-trusted code location, data relay paths through machines without the desired trust level.

8 Conclusions

In wide area networks, distributed applications must be capable of dealing with variable quality of service and disconnections. The mechanism of object replication supported in OBIWAN allows the programmer to deal with such situations; applications may decide, at run-time, what is the best way to invoke an object: via remote method invocation (RMI), or locally via local method invocation (LMI) based on a replication mechanism that brings objects to the info-appliance where an application is running.

The flexibility of the invocation mechanism allows the application programmer to develop his application in such a way that the user can continue to work disconnected from the network (either voluntary or not). As long as the objects needed by an application (or an agent) are locally accessible, there is no need to be connected to the network. In addition, by replicating objects in the info-appliance where an application using them is running, the overall performance can be improved w.r.t. an approach in which objects are always invoked via RMI.

We showed how incremental replication and object faulting resolution can be done without modifying the underlying (Java or .Net) virtual machine.

The number of objects being replicated can be changed in run-time by the application. This allows the application to balance latency, bandwidth, invocation performance and memory used. All these aspects are of utmost importance in a mobile wide-area network of info-appliances. The performance results of our prototypes, even with no special optimizations, are very encouraging.

We plan to test our prototype on several info-appliances under different network conditions (wide-area and wireless). We will study how the performance numbers presented in Section 5 depend on the relative speed of the processors involved, for example, between a hand-held PC such as Compaq iPAQ, and a desktop PC.

References

[1] Andrew Birrell, Greg Nelson, Susan Owicki, and Edward Wobber. Network objects. *Software Practice and Experience*, S4(25):87–130, December 1995.

[2] P. Butterwoth, A. Otis, and J. Stein. The GemStone object database management system. *Communications of the ACM*, 34(10):64–77, October 1991.

[3] Steve J. Caughey, Daniel Hagimont, and David B. Ingham. Deploying distributed objects on the internet. *Recent Advances in Dist. Systems, Springer Verlag LNCS, Eds. S. Krakowiak and S.K. Shrivastava*, 1752, February 2000.

[4] S. Chiba. Javassist — a reflection-based programming wizard for java. In *Proc. of OOPSLA'98 W'shop on Reflective Programming in C++ and Java*, October 1998.

[5] Geoff Cohen, Jeff Chase, and David Kaminsky. Automatic program transformation with joie. In *Proc. of the 1998 USENIX Annual Technical Symposium*, 1998.

[6] M. Dahm. Byte code engineering with the javaclass api. Technical report b-98-17, Freie Universität Berlin, Institut für Informatik, 1998.

[7] O. Deux et al. The O_2 system. *Communications of the ACM*, 34(10):34–48, October 1991.

[8] G. Chockler el al. Implementing caching service for dist. corba objects. In *Proc. of the IFIP/ACM Int. Conf. on Dist. Systems Platforms and Open Dist. Processing (Middleware'2000) - Springer Verlag*, Heidelberg, April 2000.

[9] Pascal Felber, Rachid Guerraoui, and André Schiper. Replication of corba objects. *Recent Advances in Dist. Systems, Springer Verlag LNCS, Eds. S. Krakowiak and S.K. Shrivastava*, 1752, February 2000.

[10] Paulo Ferreira, Marc Shapiro, Xavier Blondel, Olivier Fambon, Jo ao Garcia, Sytse Kloosterman, Nicolas Richer, Marcus Robert, Fadi Sandakly, George Coulouris, Jean Dollimore, Paulo Guedes, Daniel Hagimont, and Sacha Krakowiak. PerDiS: design, implementation, and use of a PERsistent DIstributed Store. *Recent Advances in Dist. Systems, Springer Verlag LNCS, Eds. S. Krakowiak and S.K. Shrivastava*, 1752, February 2000.

[11] Daniel Hagimont and F. Boyer. A configurable rmi mechanism for sharing dist. java objects. *IEEE Internet Computing*, 5, January 2001.

[12] Christopher K. Hess, Francisco Ballesteros, Roy Capmbell, and M. Dennis Mickunas. An adaptive data object service for pervasive computng environments. In *Proceedings of the Sixth USENIX Conference on Object-Oriented Technologies and Systems (COOTS'01)*, San Antonio (USA), January 2001.

[13] Anthony L. Hosking and J. Elliot B. Moss. object fault handling for persistent programming languages: a performance evaluation. In *ACM Conf. on Object-Oriented PRogramming Systems, Languages and Applications, 288-303*, September 1993.

[14] Barbara Liskov, Mark Day, and Liuba Shrira. Distributed object management in Thor. In *Proc. Int. Workshop on Distributed Object Management*, pages 1–15, Edmonton (Canada), August 1992.

[15] Malcom W. Oliphant. The mobile phone meets the internet. *Software Practice and Experience*, 36(8):20–28, August 1999.

[16] David S. Platt. *Introducing Microsoft .Net*. Microsoft Press, 2001. ISBN: 0-7356-1377-X.

[17] John Muray Reuter. *Inside Windows CE*. Microsoft Programming Series. Microsoft Press, 1998. ISBN 1-57231-854-6.

[18] Bill Venners. *Inside the Java Virtual Machine*. Java Masters Series. McGraw-Hill, 1997. ISBN 0079132480.

[19] Ian Welch and Robert J. Stroud. Kava using byte code rewriting to add behavioural reflection to java. In *Proc. of the Sixth USENIX Conf. on Object-Oriented Technologies and Systems (COOTS'01)*, San Antonio (USA), January 2001.

[20] Seth J. White and David J. Dewitt. A performance study of alternative object faulting and pointer swizzling strategies. In *18th VLDB Conf. Vancouver, British Columbia, Canada*, 1992.

[21] Ann Wollrath, Roger Riggs, and Jim Waldo. A dist. object model for the java system. In *Conf. on Object-Oriented Technologies*, Toronto Ontario (Canada), 1996. Usenix.

[22] S. Zdonik and D. Maier. *Readings in Object-Oriented Database Systems*. Morgan-Kaufman, San Mateo, California (USA), 1990.