Mobility and Wireless 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 [5, 6]. In particular, the foreseen increase of bandwidth in wireless communication makes the connection of these info-appliances to the Internet a reality [4].

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

[1] OBIWAN stands for **O**bject **B**roker **I**nfrastructure for **W**ide **A**rea **N**etworks.

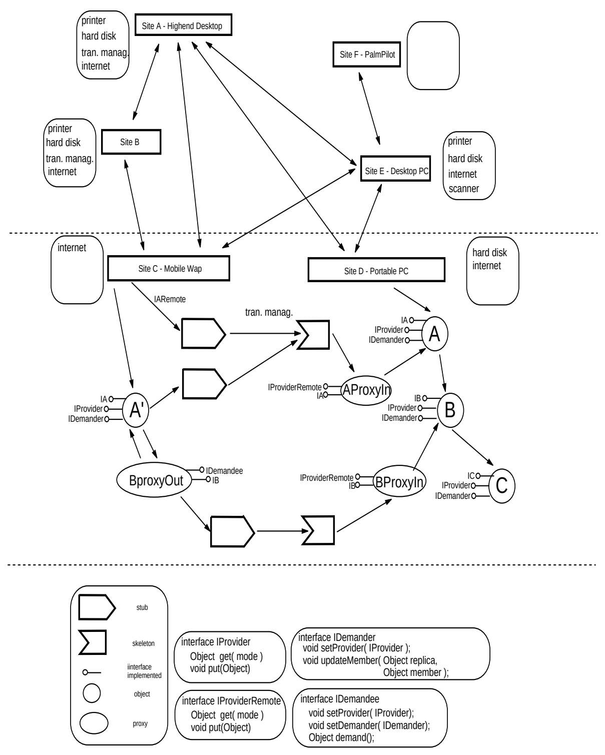

Figure 1: Main structures of OBIWAN.

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.

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 2 illustrates the architecture of OBIWAN. It shows four 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 (Java in the current implementation). 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 2 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;
- 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, 7], or locally via local method invocation (LMI) based on a replication mechanism that brings objects to the info-appliance where an application is running [2].

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 Migration

In addition, OBIWAN also supports the migration of execution flows making possible the implementation of mobile agents. However, since threads and stacks are not first class objects in Java, the programmer must provide synchronization points (checkpoint invocation) 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.

2.4 Hooks

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 sematic 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 [3].

2.5 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.

2.6 Implementation

All these services will be integrated within automatically generated proxies.

These result form the composition of simple, asynchronous policies through interface semantic self-description and automatic adaptable implementation.

This self-descriptive information can be viewed as an extension of the interface contract since it provides semantic information about interface methods. Naturally, such information should be encoded in a rich yet simple, versatile and largely adopted language like XML.

Refusal of classic distributed algorithms that rely on some kind of consensus or agreement should be considered. Emphasis on information exchange and asynchronous behavior within the computing horizon should be preferred as they are more suited to this environment. More so, proximity of other devices should trigger opportunistic behavior in the sense that resources newly available should be used when and only when they become available. This contradicts the classic point of view where resources should be available when application/agent needs them.

References

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

[2] 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 Distributed Systems, Springer Verlag LNCS, Eds. S. Krakowiak and S.K. Shrivastava*, 1752, February 2000.

[3] 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.

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

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

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

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