Loosely-Coupled, Mobile Replication of Objects with Transactions*

Luís Veiga, Nuno Santos, Ricardo Lebre, Paulo Ferreira
{lveiga,nsantos,riclebre,pjpf}@gsd.inesc-id.pt
INESC-ID / IST, Rua Alves Redol, 9, 1000-029 Lisboa, Portugal

Abstract

Advances in wireless technology and affordable info-appliances are making mobile computing a reality. However, programmers do have a real hard task while developing mobile distributed applications in which sharing is needed. Such difficulty arises, mainly because programmers are forced to deal with system level issues such as consistency, durability, availability, etc.

We designed, implemented and evaluated an object-based platform called M-OBIWAN that releases the programmer from the above mentioned system-level issues. It supports mobile transactions and replication in an integrated way.

In contrast with other approaches, M-OBIWAN provides an automatic replication mechanism allowing the creation of dynamic clusters of objects which are accessed within transactions. In addition, the transactional model is adapted to mobile environments. A prototype implementation has been developed. Its performance has been measured with PDAs and desktop machines, linked via Bluetooth.

Key-words: Replication, Mobility, Transactions, Dynamic clustering, Middleware, Loosely-coupled systems .

1 Introduction

We witness presently, and for some time now, to a vast dissemination of portable devices (e.g. laptops, tablet PC's, PDAs, etc.) made available by various manufacturers. This makes mobile computing a reality. However, programmers do have a real hard task while developing mobile distributed applications in which sharing is needed. Such difficulty arises mainly because programmers are forced to deal with system level issues such as consistency, durability, availability, etc.

As a matter of fact, mobile networks are characterized by the mobility and/or disconnection of one or more hosts. This raises the issues of availability and consistency among others. When left to the programmer, such system level issues are the cause of errors, low productivity and useless applications. M-OBIWAN[1] releases the programmer from the above mentioned system-level issues. It supports mobile transactions and replication in an integrated and transparent way.

Traditional distributed applications are client-server based. These can either be strongly connected, i.e. need a permanent connected channel or be connection-less. A strongly connected approach is not an adequate solution for economic and technical reasons. Economically, users may not wish to be online all the time due to communication costs. Technically, signal power may be insufficient, in some areas, to maintain constant connectivity and a strongly-coupled approach limits flexibility and scalability. In addition, with such a solution, availability is strictly dependent on connectivity, i.e., connection breakdown forces applications to stop.

On the other hand, in most cases, mobile devices are still regarded as small color-screen, sound-enabled clients mainly used for interface purposes. This is the case with common mobile database-oriented applications where data queries are performed, ultimately, on the servers. Clients running on devices simply ask for user input and present query results in a more or less structured and visually appealing manner. This increases load on the servers and dependency on connectivity.

The automatic replication mechanism allows the creation of dynamic clusters of objects which are accessed within transactions. Replication contributes not only to increased availability but also to performance. Objects are replicated into mobile devices so that even while disconnected, the user can still perform useful work. In addition, accessing local instead of remote data is much faster.

Some systems try to leverage devices computing-power by pre-executing queries locally on replicated data[15] with some guarantees about update-query results. This approach makes better use of device capabilities and reduces dependency on connectivity. However, the same workload is performed twice, since actions are replayed again at the

*This work has been partially funded by FEDER.

[1] Mobile Object Broker Infra-structure for Wide Area Networks.

Proceedings of the Tenth International Conference on Parallel and Distributed Systems (ICPADS'04)
1521-9097/04 $ 20.00 IEEE

servers. Thus, it does not reduce server load. Furthermore, this type of approach reduces the kind of applications supported just to those based on the traditional query-result-update model provided.

M-OBIWAN follows a loosely connected client-server approach in which objects are automatically replicated into mobile devices. The underlying requirements for this option are the following:

1) To leverage mobile devices computing-power, and its resources in general. As a matter of fact, recent evolution in mobile devices power and resources makes them viable platforms to perform a large number of tasks;

2) To minimize and/or prevent dependency on continuous connection to servers. Applications should be able to perform useful work even when disconnected from the network;

3) To allow efficient, productive and broad-based application development for these environments. In other words, to avoid the obligation to use a strict query-based kind of application. Instead, a rich object-orientation approach should be adopted.

In this paper, we focus specifically on its support for transparent, yet adaptive, incremental replication of object graphs and its integration with the optimistic transactional support.

The rest of this paper is organized as follows. The next subsection describes the rational of the M-OBIWAN design. The most important data structures and their functionality are describe in Section 2. Section 3 addresses the object replication in grater detail. In Section 4, we present the most relevant implementation aspects. Section 5 shows some performance results. Then, Sections 6 and 7 relates our work with others and draws some conclusions, respectively.

Design Rationale We consider a scenario in which mobile nodes connect to the fixed network, through base stations. Thus, mobile nodes may connect both to other mobile nodes and to fixed nodes, typically for limited amounts of time, and their connections may fail due to its inherent mobility. In addition, due to its size, such mobile devices are resource-constrained (memory, processing, power, etc.).

While working as a server, a node provides information in the form of objects. Objects contain references to other objects building *graphs* of objects. A node where an object was created is called the object's *home node*.

Nodes access objects, provided by others, by replicating them locally; we call this operation, object fetching. Such access is done within a transaction. Any object, which is part of a graph of objects, may be given a human readable name. Such objects can be seen as *roots* of a (sub)graph. Obviously, an object graph may contain several named objects, i.e. roots. Applications obtain references to such objects, from a name service, given their name.

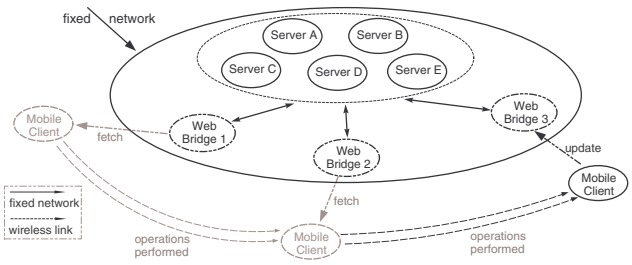

Figure 1. Typical example of mobile computation in M-OBIWAN.

Object graphs are replicated transparently and incrementally, i.e., from a remote node into a local one, within transactions. This means that application code needs not replicate objects explicitly. When they are accessed or invoked for the first time in a node, they are replicated. These replicated graphs cannot be serialized into mobile-devices "as they are" for two main reasons: i) the potential size of the replicated graph may result in memory and bandwidth flooding, and ii) increased application latency. Therefore, replication code builds object replicas with proxies standing in place of referenced objects.

Figure 1 portrays a typical example of a mobile computation with M-OBIWAN. In the example, there are five servers and three web-bridges located somewhere, over the fixed network. A mobile-client can only fetch objects from the servers, through a web-bridge, via wireless link. Initially, the mobile-client connected to Web-Bridge-1 and fetched a number of objects from one server. Following this, the client has moved, re-connected through Web-Bridge-2, and fetched some other objects from a different server. During this two periods, the client has moved once again and performed operations on the replicated objects. Finally, once re-connected through Web-Bridge-3, the client can update replicated objects back on the servers.

2 Architecture

OBIWAN[8, 20] is a middleware platform that provides support for the development and execution of applications. M-OBIWAN is an evolution of OBIWAN. The middleware has been extended in order to improve its performance on resource-constrained devices (e.g. PDAs). More important, in M-OBIWAN, object replication is integrated with optimistic transactional support. Applications are able to access objects according to a distributed transactional semantics. M-OBIWAN allows disconnected clients to proceed computation with already replicated objects. They can reconnect, later, to a different bridge, and further replicate objects or send updates to corresponding servers (e.g. when committing a transaction).

This extension poses different challenges because, in technological terms, there are several limitations in the im-

Proceedings of the Tenth International Conference on Parallel and Distributed Systems (ICPADS'04)
1521-9097/04 $ 20.00 IEEE

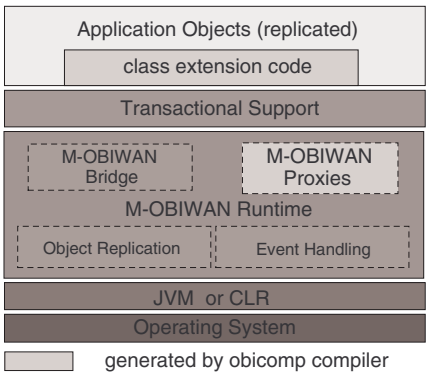

Figure 2. M-OBIWAN architecture.

plementations of virtual machines for compact devices (e.g. .Net Compact Framework and Java 2 Micro Edition). These are due, mainly, to the reduced capabilities of the devices and to minimize memory footprint of the virtual machines themselves. For example, .Net CF does not provide remote method invocation on objects nor general-purpose serialization. These are severe limitations that omit some of the mechanisms that today's distributed computing is based on.

M-OBIWAN supports the incremental replication of large object graphs into mobile nodes. Additionally, it allows the creation of dynamic clusters of data, while providing hooks (through events) for the application programmer, to implement a set of application specific properties. An example usage of such hooks is the transactional support. It is built upon the basic infra-structure and provides relaxed transactional semantics and updates dissemination.

M-OBIWAN is structured as a set of runtime libraries on top of either the JVM[1] or the CLR[14]. It is comprised, mainly, of five parts (see Figure 2):

1) Base runtime services. These include, mainly, object replication and event-handling;

2) A mobile-device bridge, for communication abstraction and flexibility among nodes;

3) Outward/inward proxy pairs[18]. An outward proxy stands in for an object that is not yet locally replicated. An inward proxy mediates and controls remote accesses to a local object;

4) Transactional support for optimistic concurrency control of replicas;

5) An open-compiler that analyzes classes and automatically generates code for proxies and classes source-code extension.

Base Runtime Services Base run-time services include object replication, registration and name service, object repository discovery and connection, and custom event-handling.

Object replication is managed through tables of entries. Each entry stores object information, namely, replication state (replicated in, out, or proxy-ed) and a weak-reference pointing to the object. This prevents that, in the same context, more than one replica of the same object co-exist. When a proxy is accessed, replication tables are first checked for an already replicated copy of the object and if so, a reference to it may be returned.

To simplify design, increase modularity and allow different, more sophisticated replication techniques, every step of object replication (into a mobile node) and update (back to server nodes) is performed by handling specially defined events (e.g., before-replica, after-replica, before-update, etc.) triggered by M-OBIWAN. This way, proxy and class-extension code do not include communication related code. This increases flexibility during object replication and updating.

Default handlers for these events are implemented in the base runtime services, performing the basic replications semantics described later (see Section 3). Nonetheless, the application, either explicitly, or in its declarative setup/configuration, may define different handlers with added versatility, flexibility, different QoS and fault-tolerance. This can be applied either during object replication or when performing their update. These primitives are the basis upon which, transactional concurrency control, and transactional policy mechanisms are built.

Communication Bridge Communication between outward and inward proxies in M-OBIWAN is performed, indirectly, with resort to two different transports. Between the communication bridge and inward proxies, remoting services are used. Since these are lacking in compact devices, web-service support is used instead, between outward proxies and the communication bridge. Therefore, communication among nodes is performed using a bridge based on a set of web-services. These web-services simply relay requests on to other nodes.

In each node, specific end-points (client and/or server) modules are set-up to encapsulate all communication. These end-points invoke, and are invoked by, the web-services. Client end-points are invoked by outward proxies and runtime services. Mediated through the web-bridge, servers deal with mobile-devices just like with any other (desktop) machine.

The communication bridge is instrumental to mobility. It enables client and server to communicate without the need of a direct connection between them. Since it is stateless, it may mediate access to any server, from any client. This way, it is possible for a client to fetch objects using one bridge, at a certain location, and later, update them back in the server, through a different bridge, in a different location.

Proxies Proxies stand-in as, and mediate remote access to objects. Outward proxies behave as local stand-ins,

Proceedings of the Tenth International Conference on Parallel and Distributed Systems (ICPADS'04)
1521-9097/04 $ 20.00 IEEE

in lieu of actual objects not yet replicated, until they are accessed/invoked. Proxies automatically implement the same functional interfaces present in application code, using polymorphism.

When a not yet locally replicated object is invoked for the first time, the corresponding outward proxy interacts with its counterpart inward proxy (residing at the remote node) to perform the replication of the corresponding object. This enables the incremental replication of object graphs. Once objects are locally replicated, invocations are direct, i.e., with no indirection at all. Proxies also mediate object updating, i.e., when local replicas are sent back to remote nodes. This happens when, for example, a transaction commits.

Transactional Support The transactional support provides a reliable form of accessing objects, ensuring that data is consistently manipulated even during node disconnections. It adopts a distributed optimistic concurrency control for ensuring serializable histories and uses a two phase commit protocol that guarantee atomicity of transactions. In addition, since the mobile environment is highly dynamic and the transaction semantics is highly dependent on the application requirements, each transaction is configurable. This means that the application developer not only implements the transaction code, but he is able to specify a transaction policy. By means of a transaction policy, the developer defines a set of rules and attributes which will determine how the transaction should behave. For example, it is able to specify if objects should be pre-fetched before executing the transaction or if objects should be fetched on demand; to specify relaxed consistency and atomicity requirements; to drop unnecessary updates if it is not possible to commit the transaction on all the participants; etc. These aspects are out of the scope of this paper and are detailed in[17].

Transactional support actually runs on top of M-OBIWAN. It makes extensive use of the hooks M-OBIWAN provides. It does so by implementing the event-handlers related to object replication (i.e., fetching of objects being accessed within transactions) and to object update back into the remote nodes (i.e., when the transaction commits). These event-handlers allow specific transaction mechanisms to intercept objects manipulated by M-OBIWAN and to manage them according to the transaction policy specified by the transaction developer.

Obicomp An open-compiler, called obicomp, is used to automatically generate code for proxy classes and augment application classes. This augmentation process does not interfere with application-logic methods. It simply implements, automatically, special-purpose code so that classes are able to create replicas of their instance objects.

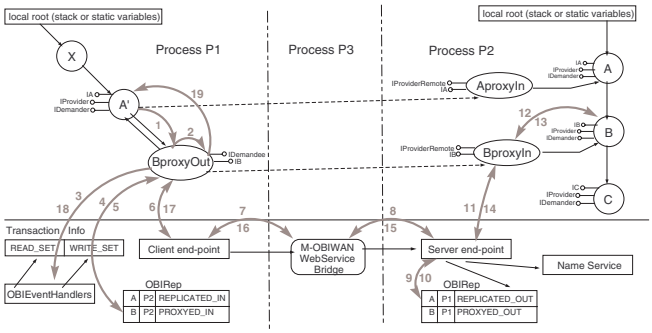

Figure 3. Example of incremental replication: Phase One.

3 Incremental Replication

To address the incremental replication mechanism in detail, a prototypical example is provided in Figure 3. There are three processes involved. Process P1 runs the client application, P2 the server application, and P3 hosts a specific web-service performing the bridge function. P1 could be running on a mobile device (e.g. PDA), P2 on a desktop machine or a server, and P3 either on the same machine as P2 or on a different one.

In process P2, there are three objects: A, B and C. Each object reveals the interfaces it implements. Interfaces IProvider, IDemander and IDemandee, are automatically implemented. Their implementation performs transparent object replication and object-fault handling. This detailed aspect is out of the scope of this paper. It is thoroughly described in [8].

Preceding the situation portrayed in the figure, the application executing on process P1 has connected to server at P2, and initiated a transaction. Object X is local to P1. Object A' is a local replica of object A in P2. A' references an outward proxy (BProxyOut) that stands in as a replica (not yet created) of object B. Since BProxyOut implements the same functional interface as B itself, A' can invoke any method or access any property of BProxyOut as if it was indeed B. Nevertheless, when that happens, the actual content of B must be replicated to P1 for execution to proceed. We now address, in detail, the necessary phases and steps to replicate B into P1.

1. P1 is executing a method -we will name it mA()- of replica A'. When mA() executes some method or accesses some property on BProxyOut, that method/property has a dummy implementation that simply triggers the replication mechanism.
2. Therefore, BProxyOut invokes BProxyOut.demande() (interface IDemandee).
3. BProxyOut triggers an event OBI.Get.Begin that enrolls BProxyOut/B', initially, in the current transaction read-set . Once objects are modified, they are also enrolled in the transaction write-set.

Proceedings of the Tenth International Conference on Parallel and Distributed Systems (ICPADS'04)
1521-9097/04 $ 20.00 IEEE

4. BProxyOut tests if already exists a replica (created through a different proxy) of object B in P1. This information is stored in structure OBIRep, that manages object replication.
5. In this situation, B is registered only as being proxy-ed in and this status is returned to BProxyOut. Thus, object B must be actually replicated.
6. BProxyOut initiates the creation process of B' into P1. It invokes method GetObject on the client end-point, the runtime sole communication gateway.
7. The client end-point forwards the call to the web-bridge. It invokes an homonymous web-service method of the bridge.
8. The web-bridge, in its turn, serves only as a an intermediary between clients and servers. Thus, clients and servers never need to be directly connected. It invokes GetObject on the corresponding server end-point.
9. The server end-point consults the OBIRep to find the entry corresponding to BProxyIn. This entry has been previously inserted when object A was replicated to P1.
10. A reference to BProxyIn is returned to the server end-point.
11. Server end-point invokes BProxyIn.get() -a method of interface IProviderRemote- to obtain a replica of the proxy-ed object.
12. BProxyIn forwards the call to B.get(). Object B, through automatically generated code, creates a replica of itself. Outward and inward proxies are created for every object referenced by B (in this case, object C).
13. The just-created replica B' (not shown) and CProxyOut, are returned to BProxyIn.
14. BProxyIn returns, both B' and CProxyOut, to the server end-point.
15. Replica B' and CProxyOut are binary-serialized, returned to the web-service bridge.
16. The web-bridge exits execution of method GetObject, invoked in step 7, returning, this time via XML serialization, B' and CProxyOut to the client end-point.
17. Client end-point returns a reference pointing to replica B', to BProxyOut.
18. BProxyOut triggers an event updating transaction info and updates replication state associated with B (from proxy-ed in to replicated in). Transaction info and replication state now refer to the actual replica, directly.
19. BProxyOut updates reference, to itself in A', so that it points directly to B'. Then, it invokes the corresponding method (initiated in step 1) in B' and returns execution to A.mA().

From this moment on, further invocations of B' will be performed without any indirection. This is important to preserve application performance. Subsequent invocations amortize the cost of object replication as they are performed at "full-speed". Incremental replication overhead is minimal, since it is dominated by the actual object network transfer (a phase that cannot be avoided). Thus, its increased flexibility comes at a reduced marginal cost.

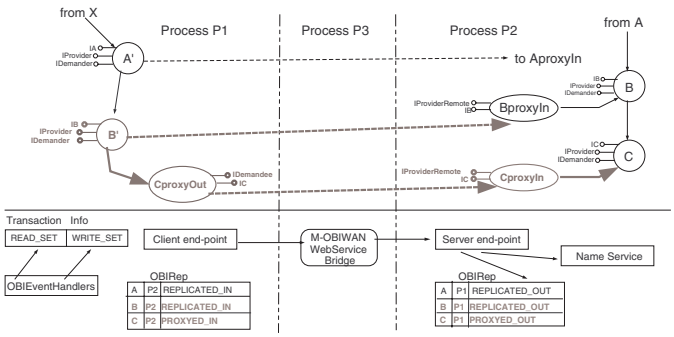

Figure 4. Incremental replication: Phase Two.

The situation resulting from creating replica B' can be seen in Figure 4. BProxyOut no longer exists and has been replaced by replica B'. In fact, for a certain period of time BProxyOut will still exist, i.e., until it is garbage collected by the underlying (Java or .Net) virtual machine. B' references CProxyOut that stands in as object C, until the first-time it is accessed/invoked (in the same manner as BProxyOut previously). Updated OBIRep structures, both at the client and server, reflect the successful replication of B and the creation of proxies to C.

Dynamic Clustering The replication mechanism just described is very flexible. It allows each object to be individually replicated. This way, only the objects that are actually needed by the application are replicated. However, in mobile environments, this approach still has its drawbacks. As far as availability is concerned, the application is still somewhat dependent on frequent connectivity. Each time an object not yet replicated is needed, the application needs to connect to the network to download it. With regard to performance, by replicating one object at a time, the application incurs in high latency costs and the available bandwidth, already low, is further wasted.

To mitigate the costs associated with these issues, M-OBIWAN allows an application to replicate a variable sub/graph of objects as a whole, i.e. a cluster. A cluster, in this context, is a set of objects that are part of a reachability graph, i.e., transitively referenced by the object being replicated. This way, a whole cluster can be replicated in a single step instead of replicating a single object individually. This is an intermediate solution between: i) having the possibility of incrementally replicate each object, and ii) the unfeasible alternative of replicating the whole graph, given the limited memory available in mobile devices.

With clustering, the dependency on connectivity can be avoided, or at least temporarily tolerated, if a cluster can be replicated in advance, i.e., pre-fetched instead of downloaded on-demand. More so, clustering minimizes latency costs, since each latency penalty is amortized by all the objects being replicated in the cluster. The depth of this prefetching can be changed in run-time: i) programmatically

Proceedings of the Tenth International Conference on Parallel and Distributed Systems (ICPADS'04)
1521-9097/04 $ 20.00 IEEE

by the application, specifying the depth of the partial reachability graph that it wants to replicate as a whole; ii) defined by simple policy-files, on a per-user, per-application, and/or per-type basis, or iii) by the run-time itself when certain conditions are detected, e.g., shortage of memory or the imminence of disconnection. So, these clusters are highly dynamic.

4 Implementation

To demonstrate and evaluate its functionality, we have developed a prototype implementation of M-OBIWAN. It was developed for the .Net and .Net Compact frameworks. The primary programming language used is C#.

M-OBIWAN base runtime, i.e., object replication and event handling resorts only to packages available in both frameworks. Server end-points were developed using Remoting services, .Net remote method invocation mechanism. The Web-Bridge was developed as a web-service, and runs on top of Internet Information Server. The client end-point runs on .Net CF as a web-service client.

The obicomp compiler is coded in C#. It makes use of reflection to analyze classes, and automatically generates proxies in C#. Proxy code invokes solely event-triggers and object replication services. It does not handle communication explicitly. This feature is key for proxy portability so that they can be used in the two frameworks, .Net and .Net CF.

Parsing of applications code currently only accepts C# source code and extends classes with replication-specific code. Due to this, application code must still be developed in C#. This is not a major drawback. M-OBIWAN runtime services and generated proxy code need not be changed, since .Net CLR executes byte-codes, regardless of the source code language used.

Regarding application code, programmers needs only to insert instructions to discover repositories; initiating and committing transactions. From that on, the programmer only needs to develop so-called application-logic. No explicit communication code, i.e., RMI, web-services, etc., is required. Replication is transparently handled by proxies. Once objects are replicated, their local proxies are replaced and discarded by the local garbage collector. Thus, the programmer never needs to invoke object replication explicitly.

5 Experimental Results

The prototype qualitative evaluation was done testing applications correct functionality in the presence of mobility, replication and optimistic transactional support.

Besides this, we analyzed prototype performance with a micro-benchmark: series of iterations were executed on a list of hypothetical appointments with 300 elements with

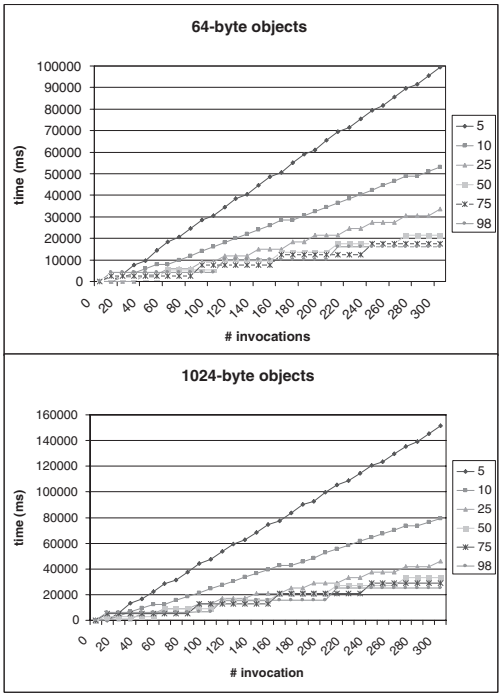

Figure 5. Performance results of incremental replication of objects with 64-byte and 1024-byte payload.

different payloads: 64 and 1024 bytes each. Therefore, as the list of appointments is iterated, in each element object, an empty method is invoked. When the object is not yet replicated, the replication mechanism takes over and replicates the object where the fault occurred. Additionally, a configurable number of other objects is also pre-fetched. In the end of each test, 300 objects have been replicated.

This performance tests were executed with the following infrastructure: a Pentium 4, 2.8 Ghz, 512 MB PC, an IPAQ 3360 Pocket PC connected through a USB Bluetooth adapter at 700Kbps. The replication mechanism was configured, by means of different policies, to replicate objects, on-demand, with a depth of 5, 10, 25, 50, 75 and 98 objects each time. This way, every time a proxy is replaced and the corresponding object is replicated, a number of others, referenced by it, are also pre-fetched. The limit depth, 98, is imposed by stack restriction on .Net CF. The graphs in Figure 5 show that replication performance is mostly latency-bound. It is more efficient when several (more than 25) objects are replicated each time. Additionally, when object payload is raised from 64 to 1024 bytes (a sixteenfold increase), performance drops only 60%.

These are rather encouraging results for various reasons. Naturally, on-demand object replication of objects masks communication latency and minimizes memory usage by applications. The number of objects pre-fetched for near optimal results needs not be too large (25 or 50). Best results are achieved with higher replication depths (75 or

Proceedings of the Tenth International Conference on Parallel and Distributed Systems (ICPADS'04)
1521-9097/04 $ 20.00 IEEE

98) but these could waste more memory if only a few of the objects pre-fetched are actually accessed. Once more, XML-base serialization, imposed by .Net CF current limitations, is responsible for some wasted bandwidth and increased processing-time due to parsing.

6 Related Work

Our work is related with others in several fields. These include mobile computing, replication, caching and reflective middleware. Initial work done regarding object-fault handling can be found in [9]. However, most of it has been centered on persistent programming languages or related to adding transparent, orthogonal persistence to existing programming languages. Nevertheless, it introduces widely accepted designations for relevant existing techniques, e.g. swizzling. Our object-fault handling is done without modifying the underlying virtual machine. This makes our solution rather portable.

Pro-Active[2] is a Java library for distributed programming with mobility in mind. It is originally based on a programming-model described in [3]. Some of its goals are common to ours: to facilitate programmer's lives in dealing with distribution issues. Neither M-OBIWAN nor Pro-Active impose any changes on the underlying virtual machines. However there are several differences: Pro-Active emphasis is on process synchronization, group communication and the absence of specific compilers/source code extenders. M-OBIWAN provides relaxed synchronization through optimistic transactions and uses an open compiler.

Javanaise [4] 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; clusters are defined by the programmer, less dynamic than in M-OBIWAN. Cluster frontiers, in M-OBIWAN, can be defined in run-time by the application in order to improve its performance and to allow disconnected work.

Some effort has been done in the context of CORBA to provide support for replicated objects [7] as well as in the context of the World-Wide-Web [4]. Nonetheless, most of this only addresses specific issues such as group communication, replication for fault-tolerance, protocols evolution, etc. Class introspection and open-compilers, used in the initial OBIWAN approach and extended in M-OBIWAN, have since also been used to portably serialize CORBA objects among different platforms[11].

Rover[10] allows both mobile-transparent and mobile-aware applications. It introduces the notions of relocatable data objects(RDO), comprising of data and code. These are the main components to define in mobile applications. RDOs communicate among each other by means of queueing RPC. This allows applications to proceed, making non-blocking remote invocations, even with disconnected hosts.

Bayou [19] presents an architecture based on mobile-aware databases, used for data-sharing among mobile users. Two basic operations are provided: Read(query) and Write(update). Clients need only to be able to access one server. Replica consistency is enforced by performing Write operations in the same, well-define order at all servers. This achieves eventual consistency among servers. Application-specific conflict resolution is performed by special types of methods: dependency checks and merge procedures.

Mobisnap[15] is a database middleware system designed to transparently support applications running on mobile environments. It allows different clients concurrently updating the database by usage of mobile transactions and reservations, though modification of persistent data is only done at the central server. Mobile transactions are expressed in unmodified PL/SQL. It allows caching of relational data in the clients and semantically infers from client transactions, the necessary constraints (reservations) that, with confidence, prevent conflicts and allow transaction completeness, independently, when they are replayed at the server.

A considerable effort has also been done in developing transaction systems suitable for mobile networks. Most of these solutions, such as Kangaroo[6], Moflex[12] and Toggle[5], provide handover protocols, allowing the execution of transactions while the mobile hosts move across the wireless cells. These systems, however, assume a different physical infrastructure than M-OBIWAN does.

Clustering[13]and Pro-Motion[16] are two mobile transaction systems that have common goals with M-OBIWAN. In both systems, data is locally fetched and replicated from its home nodes, the transaction is locally executed and later, during commit, updates are written back to its home nodes. In Clustering, objects, when fetched, can present some inconsistencies, whose degree can be directly specified by the developers. In Pro-Motion, developers have more freedom for specifying the way objects should be locally manipulated by defining compacts. However, it is not possible to dynamically and incrementally fetch graphs of objects according to the needs of applications and the contingencies of the environment.

M-OBIWAN shares some design-goals and/or practices with each of the former systems. It aims at mobility support through transparent, yet flexible replication of objects, optionally within optimistic transactions. However, in comparison with object-replication systems, it provides greater flexibility for mobile environments.

Proceedings of the Tenth International Conference on Parallel and Distributed Systems (ICPADS'04)
1521-9097/04 $ 20.00 IEEE

Regarding most mobile database-oriented systems, M-OBIWAN also allows optimistic transactional manipulation of data repositories. Moreover, it allows so, without changes to the VM and making use of a richer, object-oriented programming-model (instead of the query-update model), with no limitations concerning how data is structured or the operations that may be performed on them.

7 Conclusions

We designed and implemented M-OBIWAN, an object-based platform. It provides an automatic replication mechanism allowing the creation of dynamic clusters of objects. These clusters are accessed within transactions. The transactional model used is adapted to mobile environments.

Developers of applications for mobile devices are able to use a broad-based, productive programming-model, already familiar on centralized and classic distributed systems. Yet, they are also able to profit from the advantages inherent to mobility, and be released from error-prone, system-level issues such as consistency. The evaluation of our prototype has provided encouraging results. As presented, dynamic clustering of objects, using incremental replication, is able to mask communication-latency significantly. The presented architecture and implementation do not require any change to the underlying VM.

In future work, we intend to address the following issues:

1) To enhance M-OBIWAN performance on compact devices, we intend to use binary-serialization in the communication between compact devices and bridges;

2) To extend the M-OBIWAN bridge to run on mobile devices. This will allow devices to host object servers to other devices when there is no need, or no possibility, of connection to fixed network;

3) To extend obicomp compiler to parse VB.Net code. This will allow adaptation of a larger number of applications. We also intend to extend class byte-codes;

4) To harvest, in runtime, user and application behavior parameters to automatically determine, on user and application basis, efficient object replication strategies.

References

[1] K. Arnold and J. Gosling. *The Java Programming Language*. Addison-Wesley, 1996.

[2] L. Baduel, F. Baude, and D. Caromel. Efficient, flexible, and type group communication in java. In *Proc. of ACM Joint ACM Java Grande - ISCOPE 2002 Conference (JGI'02)*, 2002.

[3] D. Caromel. Towards a method of object-oriented concurrent programming. *Comm. of the ACM*, 36-99:90–102, 1993.

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

[5] R. A. Dirckze and L. Gruenwald. A toggle transaction management technique for mobile multidatabases. In *Proceedings of the CIKM 98*, pages 371–377, Bethesda, MD, USA, 1998.

[6] M. H. Dunham, A. Helal, and S. Balakrishnan. A mobile transaction model that captures both the data and movement behavior. *Mobile Networks and Applications*, 2(2):149–162, 1997.

[7] P. Felber, R. Guerraoui, and A. Schiper. Replication of corba objects. *Recent Advances in Dist. Systems, Springer Verlag LNCS, Eds. S. Krakowiak and S.K. Shrivastava*, 1752, Feb. 2000.

[8] P. Ferreira, L. Veiga, and C. Ribeiro. Obiwan - design and implementation of a middleware platform. *IEEE Transactions on Parallel and Distributed Systems*, 14(11):1086–1099, November 2003.

[9] A. L. Hosking and J. E. 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*, Sept. 1993.

[10] A. D. Joseph, J. A. Tauber, and M. F. Kaashoek. Mobile computing with the rover toolkit. *IEEE Transactions on Computers*, 46(3):337–352, 1997.

[11] M.-O. Killijian, J.-C. Ruiz, and J.-C. Fabre. Portable serialization of corba objects: a reflective approach. In *Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 68–82. ACM Press, 2002.

[12] K.-I. Ku and Y.-S. Kim. Moflex transaction model for mobile heterogeneous multidatabase systems. In *Proceedings of the 10th International Workshop on Research Issues in Data Engineering*, San Diego, California, 2000.

[13] E. Pitoura and B. Bhargava. Maintaining consistency of data in mobile distributed environments. In *Proceedings of 15th International Conference on Distributed Computing Systems*, Vancouver, Canada, 1995.

[14] D. S. Platt. *Introducing the Microsoft.NET Platform*. Microsoft Press, 2001.

[15] N. Preguiça, J. L. Martins, M. Cunha, and H. Domingos. Reservations for conflict avoidance in a mobile database system. In *Proc. of the 1st Usenix Int'l Conference on Mobile Systems, Applications and Services (Mobisys 2003)*, 2003.

[16] K. Ramamritham and P. K. Chrysanthis. A taxonomy of correctness criterion in database applications. *Journal of Very Large Databases*, 4(1), 1996.

[17] N. Santos, L. Veiga, and P. Ferreira. Transaction policies for mobile networks. In *5th IEEE International Workshop on Policies for Dist. Systems and Networks(Policy 2004)*, 2004.

[18] M. Shapiro. Structure and encapsulation in distributed systems: the proxy principle. In *Proc. of the 6th Intl. Conf. on Dist. Computing Systems*, pages 198–204, Boston, May 1986.

[19] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in bayou, a weakly connected replicated storage system. In *Proceedings of the fifteenth ACM symposium on Operating systems principles*, pages 172–182. ACM Press, 1995.

[20] L. Veiga and P. Ferreira. Incremental replication for mobility support in OBIWAN. In *The 22nd International Conference on Distributed Computing Systems*, pages 249–256, Viena (Austria), July 2002.

Proceedings of the Tenth International Conference on Parallel and Distributed Systems (ICPADS'04)
1521-9097/04 $ 20.00 IEEE