4

Replication–Support for Advanced Mobile Applications[1]

D. A. Kottmann
Institut für Telematik, Universität Karlsruhe
Kaiserstr. 12, 76128 Karlsruhe, Germany
Tel.: (+49) 721 608-4022, Fax: (+49) 721 388097
e-mail: kottmann@telematik.informatik.uni-karlsruhe.de

Abstract

Applications in mobile computing have to live with massive fluctuations of connectivity, bandwidth, latency, and cost in the underlying communication system. These fluctuations must be masked in order to provide the end user with predictable expenses and functionality. This can be achieved in employing cheap high bandwidth connections to set up a long term usage capability in replicating objects. This capability is afterwards exploited when communication cost increases or its quality decreases. However, off the shelf replication systems that are unaware of the applications they support, often fail to produce sufficient results. In this paper we present an approach for object based replication–support systems for mobile computing which can be tailored to specific applications. After discussing the basic concepts, we present our current prototype system MISTRAL that opens this flexibility to C++–based applications.

Keywords

Mobile computing, application–support systems, replication, advanced mobile applications

1 INTRODUCTION

Untethered communication promises to alter information processing in a way comparable to the emergence of computers from a terminal pool to the desktop (Bagrodia et al, 1995). The core of this paradigm shift towards mobile computing is that applications have to cope with massive fluctuations of connectivity, bandwidth, latency, and cost in the underlying communication system. The level of service can range from a bandwidth of 0 in coverage

[1] This work was partly supported by the German Research Council (Deutsche Forschungsgemeinschaft DFG) under grant SFB 346–A6.

J. L. Encarnação et al. (eds.), *Mobile Communications*

blackspots to broadband access in the order of several hundred Mbps when directly connected to a fixed network. Permitting users to operate as effectively as possible under these challenging conditions is the aim of advanced mobile applications. The key success factor for masking these variations is to replicate information between mobile and stationary devices and to employ available communication facilities to manage the replicas (Bagrodia et al, 1995), (Satyanarayanan, 1996). In this paper we present techniques that provide flexible replication–support for these kind of applications.
The paper is organized as follows. Section 2 gives a brief overview about the prospects of replication–support for mobile applications. Then we discuss techniques for tailoring a replication-support system to object oriented advanced mobile applications in section 3. How these techniques can be realized is shown in section 4 that presents our current prototype MISTRAL. Finally, section 5 concludes the paper.

2 THE CASE FOR REPLICATION IN MOBILE COMPUTING

Applications in mobile computing can be classified into three types (Davies, 1996): stand–alone applications, existing distributed applications, and advanced mobile applications.
Stand–alone applications, like textprocessors, are existing applications that only interact with other applications in exchanging data over a file-system or a database. These are the places where one can introduce replication–support. As the application itself exists, the support has to be transparent. Hence, issues that are specific for the mobility context must be handled completely inside the support system.
Most systems that employ replication for mobile computing today operate at the file system level and have been designed with stand–alone applications in mind. Systems like Coda (Kistler and Satyanarayanan, 1993) or LITTLE WORK (Honeyman and Huston, 1995) allow mobile applications arbitrary operations on each replicated file. This so called **optimistic update** philosophy is complemented with protocols for validating consistency of the replicas such that consistency degrades gracefully with vanishing communication abilities. As soon as the available service becomes too poor or too expensive (just think of a 24 hour GSM–connection) updates are performed locally and reintegrated when service quality and cost improves. Such independent updates bear the risk of conflicts. Sometimes conflicts can be resolved automatically by facilities like ASR (Kumar and Satyanarayanan, 1993), but often manual intervention is indispensable. As long as only stand–alone applications are used, the conflict rates are tolerable. An extensive study in Bagrodia, et al. (1995) reports rates below 0.3 % for applications like software development. On the other hand, the observed conflict rate for commercial applications peaked at 25 %. Considering specific applications, like appointment calendars or physical stocktaking, conflicts are no longer the exception. Hence, the conventional approach to replication–support is insufficient when one wants to use those applications in the mobile field.
It is nowadays a widely accepted fact that for these kinds of applications one should no longer try to keep all effects of mobility completely transparent at the system support layer (Davies, 1996), (Satyanarayanan, 1996). However, those advanced mobile applications, which have been designed with mobility in mind, still have to replicate information to compensate for the above mentioned variations in the underlying communication system. Doing this in an

application specific way is a tedious task when one has to start from scratch. Generic replication–support systems for advanced mobile applications have to offer an interface that allows the provision of application specific information. The sketched approach has already been the primary design philosophy of Bayou (Demers et al, 1994) which provides replication–support for SQL-based advanced mobile applications.
The third application type, existing distributed applications like WWW, lies in between the two discussed cases. The application itself should not be reimplemented completely, but it is possible to interfere the interactions with remote components, e.g. to satisfy requests in consulting cached information. The cache itself can either be taken from a generic support–system for mobile computing or implemented in an application specific way as in the wireless WWW system from Kaashoek et al. (1994). In the latter case, we can once again employ a replication–support system that has the capability to be tailored to specific application needs.

3 A REPLICATION–SUPPORT TECHNOLOGY FOR ADVANCED MOBILE APPLICATIONS

To ease the task of developing advanced mobile applications that employ replication, we devised a replication–support technology for objects like appointment calendars which is especially tailored for the use of mobile communication services.

3.1 Basic operations of replication support systems

First of all, we take a look at the basic operations that have to be offered by a replication–support system in mobile computing. As it is expensive or sometimes even impossible to reach other replicas, the system normally has to operate in disconnected mode (Kistler and Satyanarayanan, 1993). All cited approaches like Code, LITTLE WORK or Bayou stick to this paradigm. The replication system is employed when new replicas are created, when replicas are reintegrated or when two replicas are mutually updated. All these operations are two–party operations on exactly two replicas, as depicted in figure 1.

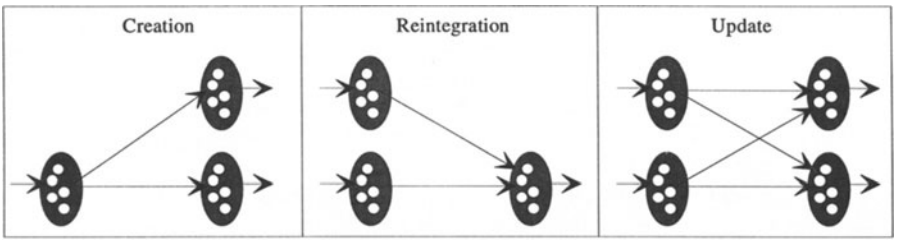

Figure 1 Basic operations of replication–support systems for mobile computing

We use figure 1 to structure replication protocols for mobile computing. We call the ellipsis **virtual partitions**. Inside a virtual partition, operations are performed disconnectedly, unless the replication protocol decides that it has to contact other replicas for reasons of consistency.

3.2 Handling the semantics of objects

A support system has to know what kind of replication–support the application requires. Our approach is to use a description of the objects' semantics via a **serial–dependency–relation** (Herlihy, 1990) and a lean classification of application–specific consistency requirements. Informally, a serial–dependency–relation tells what subset of operations executed on all replicas has to be known when one wants to be sure that another operation returns a consistent value. We discuss the idea behind this relation for the case of a simple stock–item object. The relation is depicted in table 1.

Table 1 Serial–dependency–relation for the stock–item object.

$(op_1, op_2) \in R$	op_2 = **onStock**() $\rightarrow i_3$	op_2 = **addToStock**(i_4) $\rightarrow \varepsilon$
op_1 = **onStock**() $\rightarrow i_1$	false	false
op_1 = **addToStock**(i_2) $\rightarrow \varepsilon$	$i_2 \neq 0$	false

The interpretation of the table is that the predicate in a cell tells when one has to know the operation in the row in order to be sure that the execution of the operation in the column returns a globally consistent value. Hence, one can execute arbitrary modifying **addToStock**()–operations without consulting other replicas, while one needs to know all **addToStock**(i)–operations with $i \neq 0$, executed on any replica to get a consistent result from an **onStock**() call. Thus, it is possible to give several replicas the right to increase the amount of a stock–item. How many replicas have to be contacted before executing is a matter of the consistency requirement. Due to the limited place in this paper we only allow to weaken the requirements for operations that do not alter the state of the object and we only tell the case of consistent operations from operations that return a value based on the state at the time of disconnection and the operations executed since then on that replica. Note, that we can provide two different **onStock**()–operations. One **cOnStock**() that is consistent and is used when physical stocktaking has completed to inquire the overall number of items and one **iOnStock**(), used to inquire intermediate results. Even under these simplified conditions it is possible to perform physical stocktaking in multiple stocks.

When a replication–support system knows the serial–dependency–relations R of all objects and it knows what operations are up to be performed in a virtual partition, it has enough information to test whether it can pessimistically guarantee that all operations are allowed to be performed disconnectedly. When operation op_1 is allowed in virtual partition p_1, and there is an operation op_2 allowed in virtual partition p_2 with $(op_1, op_2) \in R$, then the support system can only guarantee that one can execute op_2 disconnectedly, when there is no op^2 allowed in p_2 and op^1 allowed in p_1 with $(op^2, op^1) \in R$. When the system avoids such cycles, that can also span multiple virtual partitions, it can guarantee that the requested operations can be executed disconnectedly. This approach relies on the repeatedly reported observation that mobile users approximately know in advance what operations they need on a replica (Satyanarayanan et al, 1993); i.e. that they can tell the replication–support system what operations they disconnectedly like to execute in a virtual partition.

We use two special locks to detect such cycles efficiently: a **past**-lock for operations that have no successor in the serial–dependency–relation R, a **future**-lock for operations that have no

predecessor in R, and both locks for the remaining operations. Then we say that a virtual partition p_1 **precedes** a virtual partition p_2 when there is an object with a past–lock in p_1 and a future–lock in p_2. It is easy to verify that when we avoid cyclic precedes between partitions, also no of the above discussed cycles can result. Note, that although the lock–representation is only a coarse approximation of the serial–dependency–relation, it still suffices to give several replicas the right to increase the amount of a stock–item.

3.3 Alternatives in case of insufficient guarantees

The so far discussed approach allows to perform operations on the local replica as long as the allocated rights (i.e. locks) are sufficient. Thus, it keeps the illusion to work with a single object without the necessity of permanent expensive access to a single copy allocated a server. When an operation beyond the allocated right is requested, we can offer the application the choice among four alternatives. The superiority of one of the alternatives depends on the currently available communication system. The alternatives are:

Permanent reconciliation: Consistency is preserved via contacting other replicas. This is appropriate for high–bandwidth connections or to perform a bulk reconciliation over wireless services that charge for connection time, regardless of the amount of data that is actually transmitted.

Reassigning rights: Rights are redistributed among the replicas. This requires only minimal data transfer in the order of 20 Bytes per object. Hence it is suited for transfer over wireless links that charge the amount of transmitted data.

Switch to optimism: The operation is performed optimistically. This alternative is appropriate for coverage blackspots, when the conflict rate is low, or when automatic reconciliation is possible. We discuss below how optimistic operations are seamlessly integrated.

Defer: Try the operation later. This should be chosen when one is in a small coverage blackspot, as in a railway tunnel, or when conflict rates are high and only manual conflict resolution is possible.

Our support–technology allows applications to choose the right alternative for each situation. As an example, permanent reconciliation is the right choice as long as the mobile device is directly connected to an Ethernet or via an ISDN link.

3.4 Integrating optimism

A novel aspect of our approach is that anomalies that might arise from optimistic operations are only visible for the initiating user who in performing the operation beyond the allocated rights has accepted the risk of failure in the first place. All operations that are performed without violating the allocated rights are guaranteed to converge into a common state, even if optimistic operations have been performed on some replicas. We call this seamless integration of pessimistic guarantees and optimism **mixed–mode–replication**.
Mixed–mode–replication is achieved in logging operations that have been performed optimistically and in postponing reconciliation until the replica can contact another one so

that the sum of the guarantees covers all logged operations. On reintegration, a fine–grained conflict analysis is performed. It uses the serial–dependency–relation to verify whether an operation on the other replica may invalidate one of the optimistically performed ones. In this case, the optimistically executed operation has to be rolled back. As other optimistically performed operations might depend on the deleted operation, we use a **forward–dependency–relation** to detect these conflicts. Informally, a forward–dependency–relation tells what operations must not be deleted in order to guarantee that another operation stays valid. To see the difference between those two relations, consider an account whose balance is not allowed to drop beyond zero. Here we have to know all **credit**()–operations to make sure that we can apply another **credit**(). Hence **credit**()–operations are related by the serial–dependency relation. On the other hand, we can roll back arbitrary many credit() operations without compromising the validity of a **credit**(). Hence **credit**()–operations are not related by the forward–dependency–relation.
We can employ this fine–grained analysis to perform a flexible conflict classification. Take an appointment calendar as an example and consider the case that the same appointment is deleted independently on two replicas. Note, that on the application level the conflict is harmless and can be ignored, as both users who issued the deletions get what they wanted when a single deletion is reflected in the reintegrated object. Conventional conflict detection algorithms that are unaware of the semantics would detect an update conflict and invoke some kind of repair action. Our replication–support system detects that the conflict was caused by two deletions of the same appointment and infers from the conflict classification that no specific repair action is required.

4 REALIZATION

The discussed replication-support approach has been realized in the system **MISTRAL**. MISTRAL replicates C++–objects. The standard implementation of a C++–class has to be complemented by a specification of the internal structure to enable MISTRAL to create new objects and specifications of the serial–dependency–relation and the forward–dependency–relation. A preprocessor generates a wrapper along the lines of DC++ (Schill and Mock, 1993) to trap and log invocations. Complementing a conventional implementation of a stock–item object comprises 8 lines of description. 7 lines are necessary to build the wrapper for logging invocations and creating replicas. Only one line describes the semantics of the object. Based on this description, MISTRAL allows to dynamically create replicas on new mobile devices, to join replicas, and to assign flexible rights — e.g. to allow several replicas to increase the quantity on stock. The implementation uses XDR (RFC 1014, 1987) to code all exchanged information and currently rests on TCP/IP as a lean communication platform.
The application development process in MISTRAL is depicted in figure 3. It shows that the inputs that have to provided by the programmer consist of the implementation of the C++–class (UserClass), the specification of the used parameters in XDR (UserXDR.x), and the discussed descriptions for MISTRAL (User.def).

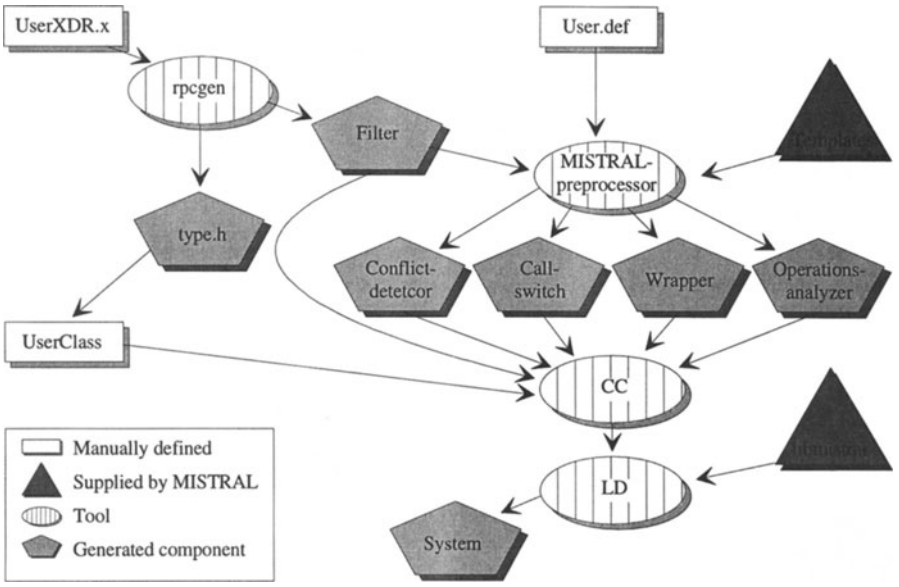

Figure 3 Application development in MISTRAL.

MISTRAL has successfully replicated objects among DECstations, DEC–AXPs, and LINUX–PCs. Figure 4 gives the performance of reintegration. The results have been obtained for the stock-item replicated on two AXP 3000/500 connected via a lightly loaded Ethernet.

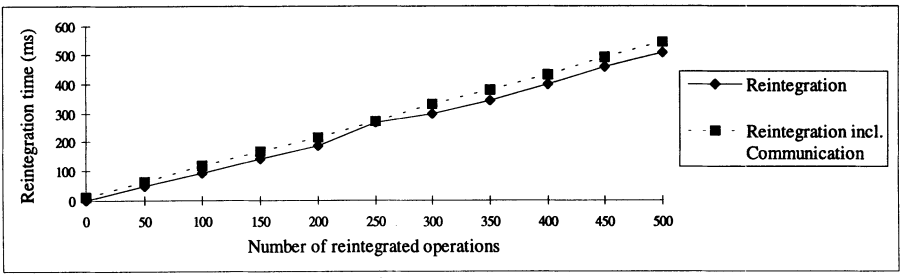

Figure 4 Reintegration Performance of MISTRAL.

5 CONCLUSIONS

In this paper we presented an approach to provide generic replication–support for advanced mobile applications. The approach exploits the specific semantics of objects, the approximate knowledge of mobile users about what they are up to, and offers four different reaction alternatives for the case that the preallocated guarantees are insufficient. Those alternatives are seamlessly integrated into the concept of mixed–mode–replication. Those features ease

the task of developing replication strategies that exploit the specifics of objects, they result in a system that can make best use of the cost structure of the currently available communication link, and they provide the mobile user with the flexibility to trade between consistency and cost while he is on the move.

REFERENCES

Bagrodia R., Chu W. W., Kleinrock L., Popek G. (1995): Vision, Issues, and Architecture for Nomadic Computing}, *IEEE Personal Communications*, 2(6), Dec. 1995, 14–27

Davies N. (1996): The Impact of Mobility on Distributed Platforms, *in International Conference on Distributed Platforms,* Dresden, Feb. 1996, 18–25

Demers A., Petersen K., Spreitzer M., Terry D., Theimer M., Welche B. (1994): The Bayou Architecture: Support for Data Sharing among Mobile Users, *Workshop on Mobile Computing Systems and Applications,* Santa Cruz, Dec. 1994, 2–7

Herlihy M. P. (1990): Concurrency and Availability as Dual Properties of Replicated Atomic Data, *Journal of the ACM*, 37(2), April 1990, 257–278

Honeyman P., Huston L.B. (1995): Communications and Consistency in Mobile File Systems, *IEEE Personal Communications*, 2(6), Dec. 1995, 44–48

Kaashoek M. F., Pinckney T., Tauber J. A.: Dynamic Documents: Mobile Wireless Access to WWW, *Workshop on Mobile Computing Systems and Applications,* Santa Cruz, Dec. 1994, 179–184

Kistler J. J., Satyanarayanan M. (1992): Disconnected Operation in the Coda File System, *ACM Transactions on Computer Systems*, 10(1), Feb. 1992, 3–25

Kumar P., Satyanarayanan M. (1993): Supporting Application–Specific Resolution in an Optimistically Replicated File System, in *Fourth Workshop on Workstation Operating Systems*, Napa, CA, Oct. 1993, 66–70

RFC–1014 (1987), Sun Microsystems, XDR: External Data Representation, June 1987

Satyanarayanan M. (1996): Mobile Information Access, *IEEE Personal Communications*, 3(1), Feb. 1996, 26–33

Satyanarayanan M., Kistler J.J., Mummert L. B., Ebling M. R., Kumar P., Lu Q. (1993): Experiences with Disconnected Operations in a Mobile Computing Environment, in *USENIX Symposium on Mobile and Location–Independent Computing*, Cambridge, Aug. 1993, 11–28

Schill A.B., Mock M. U. (1993): DC++: Distributed Object–Oriented System Support on Top of OSF DCE, *Distributed Systems Engineering Journal*, 1(2), 1993

BIOGRAPHY

Dietmar Kottmann graduated from Karlsruhe in 1992. Since then he has been a research assistant at the institute of telematics at the computer science department of the university of Karlsruhe. He is working in the interdisciplinary research group SFB 346 and is currently pursuing his Ph.D. His research interests include mobile computing, mechanisms for distributed platforms and distributed object–oriented databases.