XMIDDLE: An XML based Middleware for Mobile Computing

Cecilia Mascolo and Wolfgang Emmerich
Department of Computer Science
University College London
Gower Street, London WC1E 6BT, UK
{C.Mascolo|W.Emmerich}@cs.ucl.ac.uk

Abstract
An increasing number of distributed applications will be written for mobile hosts, such as laptop computers, third generation mobile phones, personal digital assistants, watches and the like. These applications face temporary loss of network connectivity when they move. They need to discover other hosts in an ad-hoc manner, and they are likely to have scarce resources including CPU speed, memory and battery power. Software engineers building mobile applications need to use a suitable middleware that resolves these problems and offers appropriate support for developing mobile applications. In this paper, we describe the XMIDDLE mobile computing middleware that addresses data synchronization issues using replication and reconciliation techniques. XMIDDLE enables the transparent sharing of XML trees across heterogeneous mobile hosts, allowing on-line and off-line access to data. We describe XMIDDLE using a collaborative e-shopping case study on mobile clients.

Keywords
Mobile Computing, Middleware, XML

1 INTRODUCTION
According to Mark Squires (Nokia) it took 15 years for the TV to reach a critical mass of 50 million users, but it took the mobile phone industry only 18 months to sell 50 million WAP phones in Europe alone. These mobile phones become increasingly computationally powerful, are integrated with PDA capabilities and are equipped with ad-hoc networking technologies (e.g., Bluetooth [18]). These enable new classes of applications, which for example exploit the ability to form ad-hoc workgroups; but they also present new challenges to the mobile application developer. In particular, resources, such as available main memory, CPU speed and battery power are scarce and need to be exploited efficiently. Moreover, network connectivity may be interrupted instantaneously and network bandwidth will remain by orders of magnitude lower than in wired networks.

In distributed systems, network protocols are generally hidden from the application programmer by means of middleware technologies, which raise the level of abstraction. Existing middleware technologies [10], such as remote procedure call systems, distributed object middleware, message or transaction-oriented systems, however, have been built for wired networks and are unsuitable for a mobile setting. In particular, the interaction primitives, such as remote procedure calls, object requests, remote method invocations or distributed transactions that are supported by current middleware paradigms assume a high-bandwidth connection of the components, as well as their constant availability. In mobile systems, however, unreachability and low bandwidth are the norm rather than a failure and therefore different interaction primitives are needed.

Frequent disconnection, dynamic reconfiguration, scarce resources, and need of service discovery, influence the architecture of middleware for these kind of systems. Initially commonly used middleware has been devised for mobile computing. In Bayou [19] disconnection and mobility was contemplated as a rare and occasional event. The system hides mobility from the application layer in the same way as transparency for relocation of object is used in modern middleware systems. Weak reconciliation strategies are used to support data consistency. Rover [17] adapts traditional computational models to mobile computing environments in order to provide programmers with well known paradigms.

It has recently been recognized that wireless applications need different interaction paradigms than distributed systems in fixed networks [21]. Tuple space coordination primitives, that were initially suggested for Linda [14], have been employed to facilitate component interaction for both mobile systems. Tuple spaces achieve a decoupling between interacting components in both time and space by matching the idea of asynchronicity with the mobile computing embedded concept of disconnection and reconnection. Tuple spaces do not impose any data structures for coordination allowing more flexibility in the range of data that can be handled. On the other hand the lack of any data structuring primitives complicates the construction of applications that need to exchange highly structured data.

Middleware systems for mobile computing need to accom-
We now describe the main XMIDDLE operations.

Each host has its own XML tree, which it may share with other mobile hosts. In order to share data, a host needs to explicitly mount another host’s tree. The concept of mounting a tree is borrowed from distributed operating systems, where network file systems can be mounted in order to access and update information on a remote disk. The mount operation is, however, not enough for data sharing among mobile hosts; in order to share data, hosts needs to be connected. A host becomes connected with another host when in “reach”\(^2\). When two hosts are connected they can share and modify the information on each other’s mounted XML trees. Figure 1 shows the general structure of XMIDDLE and the way hosts get in touch and interact.

The data are structured as XML trees and XMIDDLE applications can modify the trees using operations defined in the DOM API [3] to traverse the tree and add or delete nodes. Each host has full control over its own trees, however it is obliged to notify other connected hosts that mount some part of its trees of the changes that it made.

If a host wants to modify a tree mounted from a remote host to which it is connected, it requests the remote host to perform the desired changes. The remote host then broadcasts the changes to the host and all other hosts that mount the tree, too.

Hosts may explicitly disconnect from other hosts, even though these hosts may be “in reach”. XMIDDLE supports explicit disconnection to enable a host to save battery power, to perform changes in isolation from other hosts and to not

\(^2\)The definition of in reach depends on the network protocols and hardware devices used.
receiving updates that other hosts broadcast. The disconnected host retains replicas of its mounted trees and continues to be able to access and modify the data.

The ability to retain tree replicas when not connected allows a broad range of applications to be implemented on XMIDDLE as off-line manipulation of data may be useful in conditions of battery limitations or expensive bandwidth. The mount operation allows a tree to be mounted even when its owner is unreachable. When the first connection between the two hosts occurs, a replica of the tree is transferred from the owner to the mounting host. When a connection drops the copy of the tree is retained and may even be modified off-line. Such off-line modifications, however, necessitate reconciliation of replicas with their master during reconnection.

Reconciliation of XML tree replicas in XMIDDLE exploits the tree differencing techniques developed in [24]. We note, however, that reconciliation cannot in all cases be done by XMIDDLE alone. Similarly to merging text files, tree updates may lead to differences which can be solved only using application-specific policies or may even need end-user interaction. The use of XML as an underlying data structure, however, enables XMIDDLE to both highlight the differences in a way and define reconciliation policies that are specific for particular types of document elements.

3 A CASE STUDY
In order to show how the XMIDDLE primitives support building a mobile application we describe collaborative electronic shopping as a case study. Assume that a family has three members and that the family owns a PC and each member of the family has a PDA. Assume further that PC and the PDAs have embedded Bluetooth technology to establish a.hoc networks. The family does its weekly shopping electronically. To do so, the PC maintains a replica of the shop’s product catalogue that is encoded in XML as sketched in Figure 2. The catalogue on the PC is updated whenever a price or the portfolio of the shop changes. Family members replicate subsets of the product catalogue on their PDAs. We suppose that the different members hold replicas of different parts of the catalogue as they are interested in different product categories. For example, the mother may have an interest in beauty products, the father in hardware and the child in sweets and toys. The product categories however may overlap among the members. To show this in our example, we assume that Member1 is only interested in dairy products, Member2 in vegetables, while Member3 is interested in both dairy and vegetables. Furthermore, each family member has a replica of a joint shopping basket. They shop by dragging items of the catalogue into their shopping basket and by selecting quantities for these items. Reconciliation of product catalogues and shopping baskets happen whenever the PDAs establish connection with each other or with the PC.

An application engineer uses the XMIDDLE mount primitive for both mounting replicas of the PC’s catalogue subtrees and the PC’s basket. When a PDA gets within reach of the PC for the first time after the mount operation, it reconciles the

Figure 1: a. Host 2 and Host 3 are not in reach. b. Host 2 and Host 3 connect and Host 2 receives a copy of the XML tree that it has mounted from Host 3.

Figure 2: The XML representation of the product catalogue.

Figure 3: The tree representation on Member3’s PDA.
Figure 4: The tree representation on Member1’s PDA.

Figure 5: The tree representation of the data on the Member1’s PDA after orders have been entered.

PDA’s version with the PC’s version by transferring shop catalogue subtrees and the empty basket to the PDA. Member3, for example, may decide to mount the whole catalogue in addition to the empty basket. Figure 3 above shows the DOM tree on Member3’s PDA. To mount only the dairy category on Member1’s PDA, it specifies the path of the DOM tree of that category and also mounts the empty shopping basket. The resulting XML tree on Member1’s PDA is shown in Figure 4.

The applications on each PDA can now use DOM primitives to traverse the catalogue in order to display different categories and products to the family members. To implement the addition of new items to the shopping basket, DOM operations are used to create new child nodes of the shopping basket node. Let us suppose that Member1 begins to put products into the basket; a sample configuration is shown in Figure 5, where an order for one bottle of milk has been added to the basket. If these orders are entered while the PDA is connected to the PC, the implementation of the DOM operations will request updates of the DOM tree from XMIDDLE middleware on the PC. Let us now assume that the PDA was either out of reach or explicitly disconnected from the PC while these updates occurred.

Whenever a PDA establishes a connection with the PC or with another PDA a reconciliation process is started, which aims at merging updates performed at the master trees or any of its replicas so as to obtain a state of consistency. If in our example we assume that the PDA of a family member estab-

Figure 6: Schema definition of the application specific reconciliation policy.

ishes a connection with the PC, it will reconcile any updates that the PC has received via the Internet from the shop. The XMIDDLE reconciliation primitives will identify which version of the catalogue subtrees that the PDA has mounted have been modified since the last reconciliation, and it will then send the differences to the PDA so that a patch can be applied. Likewise, any updates that the member has performed on the shopping basket will be incorporated into the basket on the PC. As a second example for reconciliation, assume that members reconcile their shopping catalogues and baskets whenever they meet in different rooms of the house or in town. This allows the members to have a more up to date version of the catalogue even without connecting with the PC. It also allows a member to communicate his or her part of the shopping list to the other member so that if one of them goes home, it is immediately copied into the PC. Eventually (once a week) the home PC can then fire off the shopping list to the shop. The shopping basket on the PC is gradually filled in through synchronization with the different members of the family.

The ability for every host to update a replica opens the possibility of conflicting updates. As an example, let us now consider Member3 who also buys milk from the dairy category, this time however ordering two bottles. When the two hosts Member1 and Member3 connect their PDAs, the reconciliation process has to compute a consistent new version of the mounted branches, both the basket and the shop (as the PDAs may have two different version of the shop catalogue if they synchronized with the PC at different times). We now focus on the basket differences: the reconciliation algorithm (which is described in the next section in detail) identifies that a conflict occurred as the quantity for the milk has different values in the two basket replicas. Unfortunately, XMIDDLE cannot resolve this conflict without any application specific knowledge. XMIDDLE therefore supports the definition of conflict resolution policies for reconciliation as part of the XML schema definition of the data structures that are handled by XMIDDLE. To enable XMIDDLE to resolve the above conflict of the shopping basket, the application designer determines an additional conflict resolution policy in the XML schema. The definition of element resolver in Figure 6 shows the conflict resolution policies that are available for XMIDDLE. In particular, the schema of the
XMIDDLE namespace defines type Resolutor with values add, last, random, first, greatest.

Figure 7 shows how applications select conflict resolution policies. In our case add means that three bottles of milk will be included in the reconciled version of the shopping basket. The reconciliation of the shop catalogue among the different PDAs is also performed in a similar way. For the shop catalogue, resolutor can be set to last, to distribute the latest versions of product catalogue entries.

4 XMIDDLE ARCHITECTURE

We now present an overview of the XMIDDLE architecture, which follows the ISO/OSI reference model. Figure 8 shows the protocol stack: the mobile application layer uses the XMIDDLE primitives described in Section 5 above to connect, disconnect, mount, traverse and update its own DOM tree, as well as remote replicas. The XMIDDLE implementation that we discuss in the next section implements the session and presentation layers. The presentation layer implementation maps XML documents to DOM trees and provides the mobile application layer with the primitives to mount, unmount and manipulate replicas of remote trees. The session layer implementation manages connection and disconnection. The XMIDDLE implementations of the session and presentation layer build on top of standard network protocols, such as TCP or UDP that are provided in mobile networks on top of a Bluetooth data-link layer (i.e., Logical Link Control and Adaptation Protocol) and MAC and physical layer (i.e., Bluetooth core which is based on radio communication).

Figure 9 refines the presentation and session layer implementations of XMIDDLE. The Xmiddle Primitives API provides mobile applications with operations implementing the XMIDDLE primitives, such as mount, unmount, connect and disconnect. The Xmiddle.DOM component provides XMIDDLE’s implementation of the DOM to mobile applications. It uses the off-the-shelf DOM implementation of Apache and adds version management and update requests from connected remote hosts.

The Xmiddle Controller is a concurrent thread that communicates with the underlying network protocol and handles new connections, triggers the reconciliation procedures and handles reconciliation conflicts according to application specific policies. As XMIDDLE is entirely implemented in Java, it relies on a Java Virtual Machine (JVM). A large variety of JVMs have been implemented for mobile devices. The Symbian operating system for the third generation of mobile phones, for example, has a Java virtual machine built in. Likewise, Sun provides a minimal kernel virtual machine (KVM) implementation for Palm PDAs.

5 XMIDDLE IMPLEMENTATION

The ability to mount trees from other hosts introduces a client/server dependency between mobile hosts. We refer to the host from which a tree is mounted as the server host and the host that mounts the tree as a client host. The XMIDDLE implementation maintains this client/server relationship in a mount table and a mountedBy table that are kept on each host. It is necessary to keep track of which host is the owner of which subtree in the mount table to be able to request updates from those hosts. It is also necessary to know which host has mounted a tree for being able to broadcast updates.

The principal data structure that XMIDDLE maintains for each host, however, is a tree where each node contains a version graph of DOM trees from potentially different hosts. The XMIDDLE implementation maintains this client/server relationship in a mount table and a mountedBy table that are kept on each host. It is necessary to keep track of which host is the owner of which subtree in the mount table to be able to request updates from those hosts. It is also necessary to know which host has mounted a tree for being able to broadcast updates.

The principal data structure that XMIDDLE maintains for each host, however, is a tree where each node contains a version graph of DOM trees from potentially different hosts. Figure 10 shows an example of such a version history tree, where Host1 is mounting Host2’s and Host3’s trees: the version history of the trees are kept. It is necessary to keep a number of different versions of each tree mounted from another host in order to be able to reconcile updates. We discuss below how reconciliation is achieved. Our implementation does not use physical duplication of version trees.

Figure 9: The XMIDDLE architecture.
in order to store the version history graph; this would waste scarce memory on mobile hosts. Instead, we only store the “deltas”, that is, the differences to the predecessor version, which enable us to reconstruct predecessor versions from the current version. The computation of deltas is supported by XMLTreeDiff [2] the differencing package that we use for XML trees.

**Mount and Unmount:** Mounting a tree from a remote host is achieved by calling a XMIDDLE operation called $\text{mount}()$. The $\text{mount}()$ operation has three parameters, one to indicate the host from which to mount, one for the path of the subtree of that host, and one for the local mounting point. The XMIDDLE implementation of the mount operation records the mounting details in the $\text{mountedBy}$ table. When mounted and connected to a remote client host, the server host records the name of the client host mounting its tree in the $\text{mountedBy}$ table, which branch it is mounting, and the mounting point. This is used for broadcasting changes by the server host during connection with the client host.

**DOM Operations:** XMIDDLE provides all operations specified for tree traversal and manipulation in the DOM Level1 Recommendation. All operations access and manipulate the latest tree version in the version history graph. For all access operations, such as operations $\text{firstChild}$, $\text{parentNode}$, and $\text{nextSibling}$ in DOM type $\text{Node}$, that return data from either the own tree of the host, or any tree that is mounted, the XMIDDLE DOM interface just accesses the local DOM tree (replica) using the Apache DOM implementation. As access operations are usually much more frequently used, no remote communication is needed.

For updates of trees, we have to distinguish several cases. If we update the local tree that is owned by the host, update is performed by calling the underlying Apache DOM implementation and then broadcasting the change to all hosts that have mounted the tree (and that are connected). If we update a remote tree that is mounted from another host we again have to distinguish two cases. If the owner host is within reach, we request it to perform the update and wait until we receive the notification of the changes. If the owner is not within reach and it is the first change since we disconnected from the owner host we create a new version. Then we perform the changes on that version locally using the Apache DOM implementation.

**Disconnect and connect:** Each entry in the $\text{mount}$ table and the $\text{mountedBy}$ table identifies a remote host. We store the connection status to those remote hosts in each entry of the $\text{mount}$ table. All entries of the $\text{mountedBy}$ table of a host indicate connected remote hosts that have mounted the hosts tree. The implementation of the explicit disconnect and connect operations, therefore, modify the connection status of the $\text{mount}$ table and delete and insert entries into the $\text{mountedBy}$ table, so that these tables can be used by the implementation of the DOM. Implicit connection and disconnection due to movement of the host within or out of reach is handled by the implementation of the XMIDDLE controller.

**XMIDDLE Controller:** The underlying mobile network protocol notifies the XMIDDLE Controller that a connection with a remote host has been established. The controller then checks the $\text{mount}$ table to see whether the remote host is a client or server host. If this is the case, the connection status to the remote host is updated in the $\text{mount}$ or $\text{mountedBy}$ tables. When either an explicit connect is invoked from an application or an implicit connect is detected by the XMIDDLE controller, it starts the reconciliation process.

**Reconciliation:** The aim of reconciliation is to obtain a consistent version of the replicas of the same tree once two hosts become connected. Our reconciliation approach is composed of two main parts, one of which is application independent and one application dependent. The former is based on general techniques for XML tree comparison and merging, while the latter allows us to tune reconciliation parameters for resolving conflicts in an application-specific way. For reconciliation, consider the following situation: Host1 is mounting a branch of Host2’s tree (or a branch of some other host’s tree that is also mounted by Host2). Further assume that the two hosts have just re-established connection. Then the following reconciliation algorithm is started for the tree that Host1 and Host2 have in common. It will produce a new common version that incorporates both the changes of Host1 and Host2 as is shown in Figure 11.

![Figure 10: Tree of Version History Graph of DOM Trees.](image-url)

![Figure 11: Reconciliation of versions.](image-url)
1. **Host1** sends the list of version numbers that it has for the common tree and requests **Host2** to return the list of changes it made from the last version that the two hosts had in common.

2. **Host2** computes this list of differences using XML-TreeDiff primitives, locks access to the tree, and sends the changes to **Host1** together with the version number of the last common version.

3. **Host1** uses the differences to compute **Host2**’s version locally thus creating a successor of the common version.

4. **Host1** then computes a common successor version of its latest version and the latest version of **Host2** by merging the differences.

5. **Host1** sends the differences between the new common version and the version of **Host2** to **Host2** together with the new version number.

6. **Host2** uses the differences to establish a new current version and gives it the common version number assigned by **Host1**.

Step 4 (merging two versions) may produce conflicts if both hosts have changed or deleted the same element or attribute. These conflicts need to be reconciled.

**Conflict Resolution:** Unfortunately it is not possible to devise generally applicable conflict resolution strategies that could resolve conflicting updates between replicas without assuming application specific knowledge. XMIDDLE therefore provides the mobile application engineer who designs the underlying schemas with primitives to specify how conflicts can be resolved in an application specific way. More precisely, the XMIDDLE namespace defines an element type `Resolutor` from which the schema designer can choose the primitives that we discussed with Figure 6.

During the execution of the merge operation, the XMIDDLE controller, identifies conflicts by finding replacements of attribute or element values. If such a conflict has been detected, the controller consults the XML schema for the tree and identifies which conflict resolution strategy has been determined for the attribute or element in question. If the mobile application designer has not defined a type specific conflict resolution strategy, XMIDDLE chooses the latest change, otherwise XMIDDLE determines the attribute or element value by executing the conflict resolution strategy.

6 CASE STUDY IMPLEMENTATION
We now revisit the collaborative shopping case study of Section 3 in order to see how engineers for mobile applications make use of XMIDDLE. Figure 12 shows how Member1 and Member3 mount the shop category (i.e., dairy products for Member1 and the whole catalogue for Member3) and the empty basket, which the applications on their respective PDAs will fill with products to be purchased.

The first parameter of `mount()` is the host name from which the tree is to be mounted, in this case the PC. The second parameter is the XPath [8] expression for the root of the branch to be mounted. Consider the mounting expression for the “Dairy” products branch in Figure 12, for which we use a predicate XPath expression to select that category element, whose value of attribute `name` equals `Dairy`. The third parameter of the `mount()` is the mounting point on the local host.

When the hosts of Member1 and Member3 connect, they consider their mounting tables to identify whether they have common replicas, which they have in the above example. The reconciliation method is called by the initiator of the connection (let us say Member1’s host) in order to obtain the latest version of the catalogue (or transmit his one to the other host in case it is the last), and to reconcile the shopping list, i.e., the basket. Reconciliation of the shop catalogue is trivial, as the members would only read, but not modify it. Let us therefore focus on reconciling the basket, and let us assume that Member1 has the shopping basket that was shown in Figure 7 and that Member2’s basket contains the orders shown in Figure 13.

We now discuss the steps of the reconciliation process that will establish identical replicas of a merged basket on both hosts. To achieve this reconciliation, the host of Member1 starts by sending the list of all version numbers from the current version to the root to Member3 (in this case we suppose
Member1 never reconciled after having copied the empty basket from the PC, so he has version 1.0 only in addition to the current version. The PDA of Member3 realizes that version 1.0 is the last common version and computes the changes made from that version: she added 2 bottles of milk and 3 apples (as she is also mounting vegetables).

Member3’s host sends the updates done to version 1.0 of the basket branch, after it has calculated them using XMLTreeD-iff [2] and locks the tree. The updates are shown in Figure 15 as XMLTreeDiff differences. They are returned in such a way that the merge operation of XMLTreeDiff can take the differences and turn version 1.0 into 1.1 on Member1’s host.

The host of Member1 locks the basket branch and establishes Member3’s update in a new successor version of 1.0. It then uses XMLTreeDiff to compare Member1’s most recent version (the one shown in Figure 7) with the newly established version of Member3’s basket. XMLTreeDiff returns the difference shown in Figure 14. The merge operation then analyzes these XMLTreeDiff results and identifies that there are two differences. The first one graft is a new order that is to be inserted. This can be merged into Member1’s basket by XMLTreeDiff without causing a conflict. The second difference is a replace, which indicates a conflict. The conflicting node is the howmuch element identified by the XPath expression of the match attribute. Instead of applying the replace operation as is, the merge operation consults the application specific conflict resolution strategy in the schema and as a resolution changes the value element of the replace node to 3 (to cater for the additional bottle of milk that was in the howmuch element of Member1’s basket).

The merge operation then applies the differences to Member1’s shopping list by calling XMLTreeDiff’s merge operation. Finally, it computes the differences between the result and Member3’s list to be sent back to Member3 together with the new common version number. In this way we have fully reconciled the two versions on the PDA.

7 DISCUSSION AND RELATED WORK
We have described XMIDDLE and shown its architecture. Through a case study we have illustrated how it is used and shown the details of the tree reconciliation algorithm and mounting used for data synchronization among the mobile devices. Synchronization and data locking has been described as one of the main problems in wireless environments by Imielinski and Badrinath in [16]. XMIDDLE offers a possible solution.

We focused our interest on ad-hoc networks where host configurations are relative and dynamic. No discovery services are set-up as in Jini [4] as all the hosts are equal and have the same capabilities. They are able to reconfigure their own connection groups while they move through connection and disconnection with the other hosts.

Tuple space based systems for logical and physical mobility such as JavaSpaces [13], Lime [20], Limbo [9], TSpaces [15], and Mars [6] exploit the decoupling in time and space of these data structures in the mobility context where connection and disconnection are very relevant and frequent operations. However, tuple spaces are very general and loose data structures which do not allow complex data organizations and therefore do not fit all the application domains. XML allows us to introduce hierarchy of data and to address specific paths in the structure so that more elaborated operations can be performed by the applications. The value of XML in structuring data has already been recognized and some work has also been carried out to integrate tuple spaces and XML: in [7] a mobile agent system based on tuple spaces is integrated with XML for the encoding of data. This allows a more structured way of dealing with data communication, while introducing flexibility in the data treatment. In that paper, however XML is only used for data formatting. Tuple are translated into XML files and stored into a data-space. In [1] XML is used to create a lightweight repository of XML documents, based on IBM’s TSpaces. This repository supports XML (DOM) oriented queries. XML documents are somehow stored as tuples in the tuple space. TSpaces recently offered direct support for storage and indexing of XML documents. This is done by transforming XML documents into a tree of TSpaces tuples, linked internally via pointers.

XMIDDLE uses XML tree only as data structure and exploits the power of the nature of the data structure with specific operations; for instance, the mounting primitive allows the off-line sharing of information which is very valuable in mobile computing contexts where hosts have the need to move away from the source of information even if they may want to continue to work on the downloaded data. Reconciliation mechanisms are needed to maintain a certain level of consistency and to support synchronization. Existing mobile computing middleware systems do not address this point and a consortium has just been built in order to provide standards for mobile computing data synchronization; XML is considered as playing an important role is the definition of the data exchange format for that [23].
The XMIDDLE strategy for data synchronization exploits well established techniques and tools for replication and reconciliation on trees [24, 2]. In [22] a technique for application-independent data reconciliation for nomadic computing settings is described. The technique does not focus on particular data structures, instead it defines a generally applicable strategy to synchronize replicated data on mobile hosts. XMIDDLE allows the combination of application dependent and independent techniques, exploiting semantic knowledge about element types; a set of reconciliation primitives are defined in XMIDDLE, as described in Section 3, and the mobile application engineer can specify the way these primitives are combined to determine an application-specific reconciliation policy.

Tanenbaum is addressing data reconciliation in distributed object middleware in the Globe project [26]. The aim of Globe is to provide an object based middleware that scales to a billion users. To achieve this aim, Globe makes extensive use of replication. Unlike other replication mechanisms, such as Isis [5], Globe does not assume the existence of an application independent applicable replication strategy. It rather suggests that replication policies have to be object-type specific, and therefore they have to be determined by server object designers. Thus, Globe assumes that each type of object has its own strategy that pro-actively replicates objects. XMIDDLE policies definition follows this approach.

The XMIDDLE reconciliation algorithm is relying on versioning mechanisms. Like text-based versioning systems, such as RCS [25], we store and transmit differences as shown in Figures 15 and 14 to minimize the transmission load during reconciliation: only the updates from the last common version are exchanged between the hosts. Unlike text-based versioning, however, the differences the XMIDDLE implementation is able to obtain from XMLTreeDiff are more precise and semantically richer. This is because the differing algorithms are able to take attribute and element, as well as their arrangements in trees into account. This generally leads to less conflicts than when we used text-based differencing tools.

8 CONCLUSIONS

The growth of the recent mobile computing devices and networking strategies call for the investigation of new middleware that deal with mobile computing properties such as disconnection, low/expensive bandwidth, scarce resources and in particular battery power, in a natural way. XMIDDLE is one possible answer to these needs that focuses on data replication and synchronization problems and solves them exploiting reconciliation strategies and technologies.

The implementation of XMIDDLE is on-going and we plan to continue also considering the new and very frequent advance in hardware technology and network protocols. The reconciliation and mounting policy can be refined and we are considering more case studies to deal with the conflict resolutions in a mixed application/non-application oriented fashion.

The definition of policies for inconsistency resolution during the reconciliation process may also be considered as quality of service specification. By defining the level of consistency the application needs on specific data it is possible to specify different “quality of reconciliation”. We intend to investigate this approach further.

Security policies can also be established in order to limit the access of hosts on XML trees. For instance, specific branches of the trees may be defined as accessible to all the hosts while other branches may be accessible only to particular hosts. This can be done using an “exports” file which allows a host to make only some subtrees remotely accessible while retaining exclusive access to other subtrees. Digital signatures and common security strategies (i.e, passwords and public/private keys) could be applied as well in order to guarantee further levels of security.
The use of XML for the data formatting has, as we said, advantages at the level of the tree structure and at the use of readily available technologies, but also at the information rendering level as XSL and WAP could be integrated in order to custom the display of the data for different mobile devices.

In [11] we used XML for the implementation of a fine-grained code mobility approach which allows single lines of code to be transferred among hosts in an incremental manner. XMIDDLE allows data sharing through XML, however, using the approach presented in the mentioned paper we could provide code sharing and mobility using the same XML format. This feature would power XMIDDLE with more flexibility and extensibility: we plan to look into this aspect.

Tuple spaces based systems allow notification of events on the tuple spaces through different ways like transactions and the tuple spaces through different ways like transactions and reactions. We plan to extend XMIDDLE by introducing some event notification mechanisms that allow hosts to register for events on trees. At the moment a basic event notification mechanism is in place for connected hosts to be notified about the modification of mounted tree branches, but some extensions can be developed.

ACKNOWLEDGEMENTS
We would like to thank Anthony Finkelstein and Engin Kirda for the helpful discussions on the topic.

REFERENCES