

Copyright (c) 2000 Institute of Electrical and Electronics Engineers. Reprinted from *Proceedings, Tenth International Workshop on Research Issues in Data Engineering*, February 27-28, 2000, San Diego, California. IEEE Computer Society, Los Alamitos, California, pp. 9-16.

This material is posted here with permission of the IEEE. Such permission of the IEEE does not in any way imply IEEE endorsement of any of IBM's products or services. Internal or personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution must be obtained from the IEEE by sending an email message to pubs-permissions@ieee.org.

By choosing to view this document, you agree to all provisions of the copyright laws protecting it.

Enterprise Data Access from Mobile Computers: An End-to-end Story

Maria Butrico, Norman Cohen, John Givler, Ajay Mohindra, Apratim Purakayastha, Dennis G. Shea
I.B.M. T.J. Watson Research Center, Yorktown Heights, NY

Josephine Cheng, Don Clare, Gerry Fisher, Rob Scott, Yudong Sun, May Wone
I.B.M. Santa Teresa Labs, Santa Teresa, CA

Quinton Zondervan
Lotus Development Corporation, Cambridge, MA

Abstract

Currently handheld and palmtop computers are widely used for personal information management. In the near future they will also be used to access enterprise data. There are however, numerous technical challenges in enabling an end-to-end system that provides enterprise data access from mobile computers. The challenges include heterogeneity, various resource constraints, scalability, and security. In this paper, we describe the design and implementation of the Mobile Data Synchronization Service (MDSS), an end-to-end system that provides enterprise data access from mobile computers. Specifically, we address the heterogeneity of devices and data sources, the memory and power constraints of devices, the poor quality of communication, and the need for scalability. Our system achieves interoperability and solves the key technical challenges related to enterprise data access from mobile computers.

1. Introduction

Handheld and palmtop computers are becoming increasingly popular. Currently such mobile devices are mostly used for personal information management. There is, however, a clear trend toward using these devices for business applications that access enterprise data. For example, mobile insurance workers need access to rate quotes and customer data, and mobile salespersons need access to inventory data.

There are numerous technical challenges in enabling an end-to-end system that provides enterprise data access from mobile computers. There are a wide variety of heterogeneous devices, networks, and enterprise data sources. Thus, an end-to-end system must be interoperable across diverse platforms and networks. The client devices are memory and

power-constrained. Thus, the system must optimize footprint and power consumption. Communication is unreliable and expensive. Thus, the system must operate well over intermittent, lossy connections and limit communication costs. Wireless networks are easily snooped and mobile devices are prone to loss or theft. Thus, the system must protect enterprise data in transit and on client devices. Finally, the proliferation of mobile devices requires the server infrastructure to be scalable and able to withstand traffic storms.

In this paper we describe the design and implementation of the Mobile Data Synchronization Service (MDSS). MDSS is an end-to-end system that enables enterprise data-access from diverse client devices to diverse data repositories over diverse networks. In Section 2 we present the overall design rationale of the system, in Section 3 we outline our synchronization protocol, in Section 4 we discuss the actual design and implementation of the system, in Section 5 we present related work, and finally in Section 6 we conclude.

2. Design Rationale

In this section we discuss the major design decisions that shape MDSS. The most significant decisions are: (1) a logical three-tier architecture, (2) a common data interchange format, (3) a high-level open synchronization protocol, and (4) a relaxed correctness model.

One of our critical design goals is to support diverse clients and back-end data sources. If a client needs to communicate directly with numerous data sources the client must know how to communicate with each data source. This solution does not scale for memory-constrained clients, as each client requires additional code to support each data source (Figure 1a). We first consider a three-tier approach to address this problem (Figure 1b). In this approach, logically a client corresponds to a single

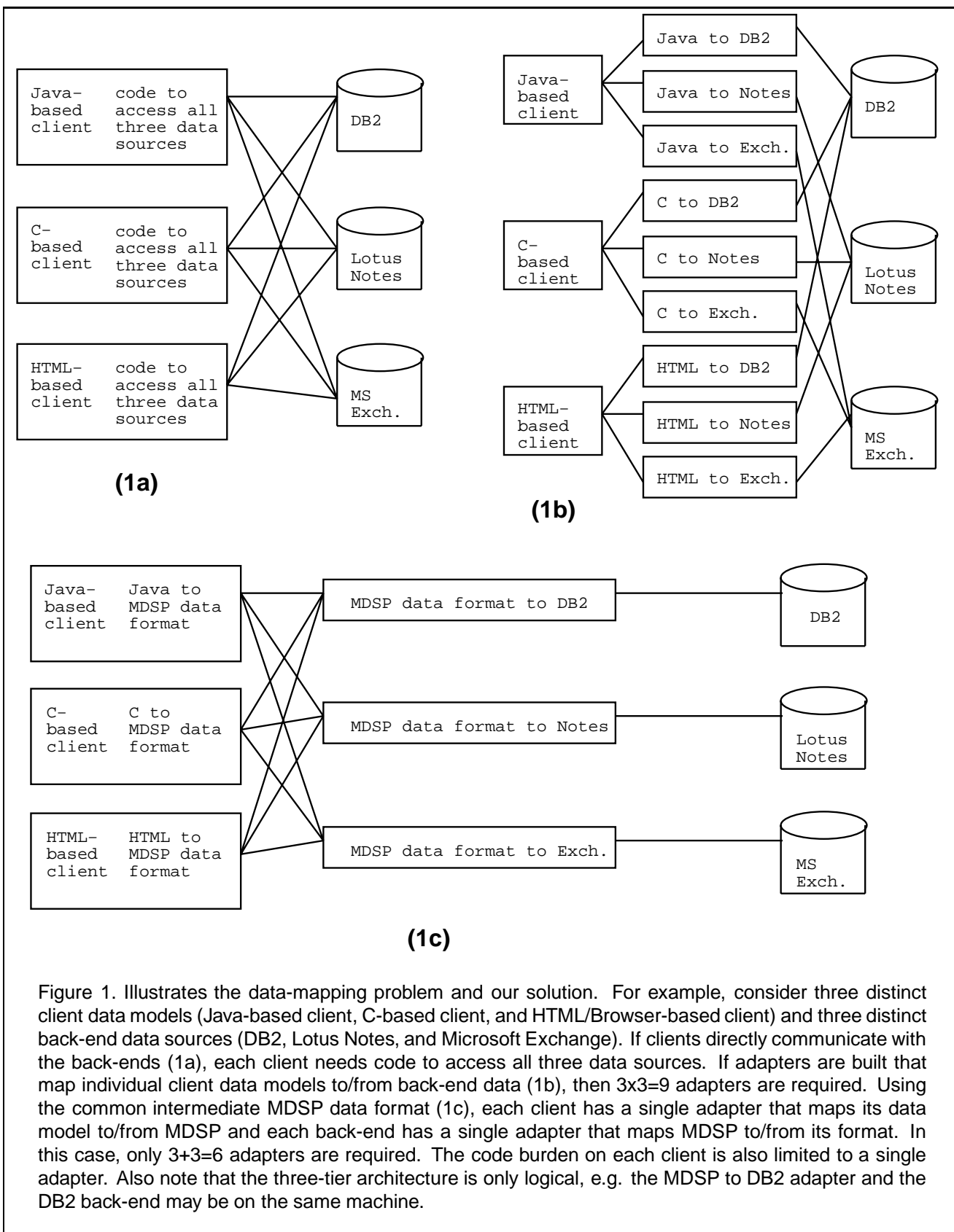


Figure 1. Illustrates the data-mapping problem and our solution. For example, consider three distinct client data models (Java-based client, C-based client, and HTML/Browser-based client) and three distinct back-end data sources (DB2, Lotus Notes, and Microsoft Exchange). If clients directly communicate with the back-ends (1a), each client needs code to access all three data sources. If adapters are built that map individual client data models to/from back-end data (1b), then $3 \times 3 = 9$ adapters are required. Using the common intermediate MDSP data format (1c), each client has a single adapter that maps its data model to/from MDSP and each back-end has a single adapter that maps MDSP to/from its format. In this case, only $3 + 3 = 6$ adapters are required. The code burden on each client is also limited to a single adapter. Also note that the three-tier architecture is only logical, e.g. the MDSP to DB2 adapter and the DB2 back-end may be on the same machine.

data model. The client communicates with an adapter that is aware of the client's data model as well as the back-end data model. During data exchange between the client and the data source, the adapter maps each data model to the other. Although this approach requires no additional code on clients to support various data sources, it causes an explosion in the number of required adapters. To support m client data models and n back-end data source types, mn adapters are necessary.

We address the above problem by using a common data interchange format, called the MDSP data format¹. We logically *split* the adapter into two adapters (Figure 1c). One adapter resides on the client and translates the client's data into the MDSP data format and vice-versa. The other adapter resides on some server and translates the MDSP data format into a particular back-end data format and vice-versa. In this new architecture, although a client needs an adapter to translate between its data model and the MDSP data format, the total number of required adapters² for m client data models and n back-end data source types is only $m+n$.

Mobile devices vary widely in capabilities, programming-language support, and transport-protocol support. Existing application-level data synchronization protocols (see Section 5) make assumptions about one or more of the diverse capabilities of mobile devices, transport protocols, and back-end data sources. Such assumptions limit the applicability of the synchronization protocols. We need a higher-level synchronization protocol that is device, transport, and back-end agnostic. To be truly interoperable, the protocol must also be open and extensible. As part of the MDSS effort, we developed the Mobile Data Synchronization Protocol (MDSP). MDSP is an XML-based application-level synchronization protocol that is device and network agnostic, as well as open and extensible (see Section 3).

Mobile devices are often disconnected. Correctness models such as one-copy serializability [7] or bounded inconsistency [32] are not practical in this domain, as they require communication with other machines for even accessing local data. We use a relaxed correctness model in our system that only guarantees eventual consistency [7], and is appropriate for many applications in this domain. In our model, clients can read and write data when disconnected. Data on client devices is synchronized from time to time with the central enterprise data. Direct client-to-client synchronization is not supported, although the clients can indi-

¹The common data interchange format is embodied by the various data elements in the Mobile Data Synchronization Protocol (MDSP), and hence called the MDSP data format. See section 3.1 for a more detailed description of the MDSP data format.

²Notwithstanding the fact that every client in our system has an adapter, hereafter in this paper, we refer to the entity that translates between the MDSP data format and a back-end data source as an *adapter*.

rectly share data and affect each other via the central server. Due to concurrent updates, synchronization may generate conflicts, which may result in invalidation of a client's modification, and application of compensating updates that are sent by the central server. Eventually, in some steady state (when no updates are being made), all clients and the central server will have the same view of the data. Workload characteristics of our target application domain indicate that client replicas are not heavily write-shared. Heavy write sharing scales up conflicts and resulting invalidations. Also, in comparison with continuous updates, periodic synchronizations in our model better amortizes network connection overheads.

3. The Mobile Data Synchronization Protocol (MDSP) and its Usage

MDSP is a client-server data exchange and synchronization protocol defined using an XML Data Type Definition (DTD) [9]. MDSP contains *elements* defining data and processing instructions. Each element has a regular expression-like structure associated with it. Each element can also have a set of *attributes*. Some attributes may have values that are free-form strings and other attributes may have values that are taken from a fixed set of tokens. We broadly classify MDSP elements into three major classes: data elements, command elements, and information elements (see Figure 2). The data elements embody the MDSP data format that is used as the common data interchange format for data exchange between clients and back-end adapters. The command elements define the data exchange and synchronization *operations* of MDSP. The information elements provide meta information about data and operations, or may contain results of an operation.

3.1. The MDSP data format

Each instance of data is enclosed in a DATA_ELEM element that identifies the *target*, *source*, *format*, and size of the data (among other things) via attribute values. A DATA_ELEM may contain zero or more DESCRIPTION elements that qualify the data, at most one DATA_ID element (data identifier), at most one VERSION element (data version), and at most one of DATA or STRUCTURED_DATA elements. The DATA element in MDSP is used to capture inline text or encoded binary data:

```
<DATA>
  <!-- Inline text or encoded binary -->
</DATA>
```

The STRUCTURED_DATA element can contain any XML data. The enclosed data may use an external DTD

DATA ELEMENTS

- **Container**
 - DATA_ELEM, DATA_COL
- **XML data**
 - STRUCTURED_DATA, PROPERTY_MAP, PROPERTY
- **inline data**
 - DATA

COMMAND ELEMENTS

- **data exchange**
 - GET, ADD, UPDATE, COPY, REMOVE, REFRESH
- **synchronization**
 - SYNCHRONIZE, MAP
- **auxilliary**
 - ATOMIC, SEARCH, EXECUTE, ALERT

INFORMATION ELEMENTS

- **Metadata**
 - DATA_ID, VERSION, DESCRIPTION
- **Status & Results**
 - STATUS, ERROR, ERRORLIST, GET_RESULT, REFRESH_RESULT

Figure 2. The figure illustrates several elements in MDSP. The elements in MDSP can be broadly classified into data, command, and information elements. The data elements constitute the MDSP data format, and are designed to accommodate both structured and unstructured data. Command elements signify various data exchange and synchronization operations. Information elements provide meta information about data or an operation, or may contain operation results.

such as vCalendarML (industry standard XML notation for calendar entries), or may specify property-value pairs using the PROPERTY_MAP and PROPERTY elements. The PROPERTY elements provide a means to easily express rows of tabular data (see examples below).

```
<STRUCTURED_DATA>
<VCALENDAR>
<VEVENT>
  DTSTART: 199902280830
  DTEND: 199902280930
  SUMMARY: Boring Meeting
  DESCRIPTION: Really boring meeting
</VEVENT>
</VCALENDAR>
</STRUCTURED_DATA>
```

```
<STRUCTURED_DATA>
<PROPERTY_MAP>
<PROPERTY name="empName">
  <DATA>Louis V. Gerstner</DATA>
</PROPERTY>
<PROPERTY name="jobTitle">
  <DATA>Chairman and CEO</DATA>
</PROPERTY>
</PROPERTY_MAP>
</STRUCTURED_DATA>
```

3.2. Data Exchange and Synchronization using MDSP

The command elements of MDSP typically specify an operation invoked by the recipient of the message. MDSP makes an effort to capture common operations such as fetching, updating, inserting, and removing data using commands such as GET, UPDATE, ADD, and REMOVE. In MDSS, instead of using individual data exchange operations, we batch numerous data exchange operations in one *synchronization session*³. For this purpose, we use the SYNCHRONIZE element in MDSP that provides an enclosing context for a batch of data exchange operations. A single synchronization session is divided into two distinct phases. In phase 1, a client initiates synchronization by sending the following message to a server (adapter):

```
<SYNCHRONIZE
target = http://us.anycorp.com/employeeDB
user = foo
password = foobar>
  <UPDATE>
    <DATA_ELEM>.....</DATA_ELEM>
    <DATA_ELEM>.....</DATA_ELEM>
  </UPDATE>
  <REMOVE>
    <DATA_ID>.....</DATA_ID>
```

³This section outlines *our use* of MDSP for synchronization. The MDSP operations can be used in numerous other ways.

```
<DATA_ID> . . . . . </DATA_ID>
</REMOVE>
</SYNCHRONIZE>
```

The message typically identifies a server side data store, provides authentication information, and lists the client's updates since the last synchronization. After phase 1, the client has an opportunity to disconnect to save communication costs, power down to save battery, or continue to only read elements from its local replica while the server processes its updates. The server discards the client updates that *conflict* with its own updates⁴, or client updates that violate integrity constraints. Upon processing the client updates, the server constructs a SYNCHRONIZE message similar to the one above that contains new server updates, compensating updates for conflicting/rejected client updates, and an ERRORLIST element that lists the rejected client updates. After some time, the client initiates phase 2 of synchronization. In phase 2, the client simply receives the SYNCHRONIZE message from the server and processes the updates. The time interval between phase 1 and phase 2 is a configurable application parameter.

3.3. The pros and cons of MDSP

MDSP offers a number of benefits. First, its use of XML allows for a rich, hierarchical structure of data/operations compared to relatively flat structures in plain text or binary protocols. The use of XML also facilitates wide acceptance and availability of good programming tools. Second, it is open and extensible: e.g. features like capability negotiation between devices and servers can be added to it easily. Third, it accommodates general data types since new data types can be used simply by making external DTD references. Fourth, it is transport agnostic; it can be mapped onto a wide variety of transports such as HTTP or TCP/IP. Finally, it allows operations to be batched to make better use of bandwidth. There are also several concerns regarding MDSP. First, it is verbose and inefficient. Verbosity can be handled with smaller tags and attribute names, or during transmission, an efficient binary encoding such as WBXML [5] can be used. Second, XML parsers are hard to fit in small devices. Since we expect MDSP to be generated automatically, it does not require a validating parser. Using a non-validating parser can alleviate footprint concerns, and if required, a small custom MDSP parser can always be built. Third, XML cannot handle binary data. Binary data can be included as encoded (e.g. base64 encoding) text, or if WBXML is used for transmission, binary data can be transmitted using WBXML binary extension features [5].

⁴Discarding conflicting client updates only reflects a policy decision. It is not mandated by our design.

4. System Design and Implementation

In this section we discuss the design issues related to various parts of the system and explain the implementation choices we make. We discuss the client, the adapter, and the transport protocol.

4.1. The Client Platforms

We have implemented two data models for clients. The *object-based* data model facilitates access to unstructured or semi-structured data; the *relational* data model facilitates access to uniform tabular data. The object-based data model is Java-centric (although it can be extended to other programming languages). It follows the Mobile Network Computer Reference Specification (MNCRS) data synchronization framework [6]. The data model consists of a persistent object store that contains Java objects identified by keys. The keys themselves are Java objects. The stored objects belong to application-defined classes and have the flexibility to implement application-specific policies to resolve update conflicts during synchronization. The framework defines interfaces for a pluggable versioning mechanism, a pluggable persistent layer, and a pluggable synchronizer. We have implemented various forms of versions including dirty bits, integer versions, and version vectors. For clients that interact with a single server we found that the dirty-bit versions work well, but integer versions simplify recovery from failures. We have implemented two persistent storage layers that use log-based and malloc-style algorithms to optimize storage space and access time for objects.

The relational data model on the client is centered on DB2 Everywhere (DB2e) [11], a lightweight implementation of DB2. A table on the client side represents an appropriate view of the corresponding table on the server side. The view may be limited to a projection of a set of selected columns. The basic version of DB2e supports a subset of SQL comprising of simple queries. Java or C/C++ programs on clients may use DB2e via standard interfaces such as JDBC/ODBC. This data model is well suited to simple client-side operations such as lookup and update, and fast synchronization where most of the synchronization logic resides on the server side. The object model above is well suited to handle non-uniform heterogeneous data. The footprint of the object-based client implemented in Java is 4 to 5 times more than the lightweight relational client implemented in C, and hence the latter is more suitable for severely memory-constrained clients (e.g. Palm Pilot, cell phones). A native implementation of the object-based client will reduce its footprint.

The object-based Java client and the relational client both use synchronizers based on MQSeries Everywhere (MQe) [12], a lightweight implementation of IBM's queue-

based messaging software MQSeries [12]. This approach separates the model of synchronization from individual client data models; clients simply put and get MDSP messages to/from named queues.

4.2. The Adapter

The *primary* role of an adapter in our system is to map MDSP data to/from back-end databases such as DB2, Oracle, Lotus Notes, and Microsoft Exchange. We have currently implemented an adapter for DB2; adapters for other databases are planned. The DB2 adapter is implemented in Java such that it can run on a variety of server platforms including WindowsNT and AIX. The DB2 adapter also performs additional roles of authentication and client administration.

Since the adapter serves a potentially large number of clients, the focal design goal is scalability and performance. Enterprise data is usually stored in a back-end “master” database. During synchronization, accessing and updating the master database directly (e.g. using JDBC) for each client update will result in excessive competition for connections to the master database and will likely degrade the performance of the adapter. To avoid such contention for master database access we use the DB2 replication facility DataPropagator [10], that periodically reconciles client updates with the master database by using “mirror tables” that reflect actual master database tables.

Each client synchronization session is first assigned a lower-priority service thread from a dual-priority thread pool. Each service thread is associated with a synchronization *window*. This window remains open for a tunable duration and has numerous service threads associated with it. Only one such window is open at a time. The service threads stage the client data in *staging* tables associated with each database and thereafter reenter the thread pool with higher priority. Each window has a *window monitor* thread that monitors the cumulative state of the service threads. When all service threads complete staging of their data, the monitor closes the window and marks it ready to proceed with the next stage of synchronization and blocks. In the next stage, a per-database thread first analyzes staged entries from all clients in a given window for readily detectable conflicts and marks them as such. Then it processes remaining entries by performing appropriate operations to mirror tables. Next, it invokes the DataPropagator processes, analyzes results of replication, notes additional conflicts, records new master database activities as updates to versions in staging table rows, and finally wakes up the window monitor thread. The window monitor thread wakes up higher-priority service threads that extract data for a client, construct an MDSP response message and place the message in the appropriate MQe output queue (see Sec-

tion 4.3), and thereafter reenter the thread pool with lower priority. Once all the service threads in a window are done, the window monitor thread exits.

Clearly, the adapter is designed to maximize scalability and performance. Use of DataPropagator reduces contention for master database access. The prioritized service thread pool efficiently handles a large number of clients in peak traffic. Finally, the synchronization window batches updates from clients such that invoking the relatively expensive replication operation is justified.

4.3. The transport protocol

Mobile environments are often associated with unreliable communication, such as wireless communication. Although TCP/IP provides reliable application-to-application transport, it is not suitable for wireless networks. First, it cannot make incremental progress on message transmission. In a lossy environment, TCP/IP can send all but one packet of a long message and still fail, forcing the application to retry sending the whole message. Second, TCP/IP is synchronous in nature. Applications simply have to wait until the whole message is delivered, or TCP/IP declares failure. This requirement may not fit the “instant on” or “instant access” promises made by the handhelds.

A store-and-forward messaging mechanism can eliminate the need for applications to wait until a message is delivered or a delivery failure is declared. If the transport layer can be entrusted with safekeeping of the message until it is delivered, the application can proceed after handing the message over to the local transport layer. It is better yet if the store-and-forward messaging mechanism can also make incremental progress on individual messages. We use MQe, which provides persistence via queuing and guarantees at-most-once message delivery. Incremental message delivery mechanisms suitable for wireless environments can be implemented under the MQe APIs. Use of store-and-forward messaging also improves response time for clients. When all the adapter service threads are busy handling heavy load from clients, the adapter queues can act as buffers for outstanding client requests. The clients do not have to wait for synchronous message handoffs to adapter threads.

We believe that another key to success in the mobile/wireless domain is application-assisted communication. If the application can break down large messages in a meaningful manner and then hand the smaller parts to the transport layer, the overall reliability and efficiency of the system is likely to improve. This application level message restructuring has the added benefit of not requiring large buffer spaces, which is critical for space-constrained clients.

5. Related Work

Disconnected operations are addressed in systems such as Coda [15] and Ficus [25]. These systems focus only on files. They also either involve manual conflict resolution or coarse-grain automatic conflict resolution based on file types [16]. Enterprise data access however, involves finer-granularity data such as rows in a database table or fields in a document. Policies suitable for files are not directly applicable in our domain. The Bayou [30] project addresses replication and synchronization between tuple stores. It is geared toward collaborative applications and hence its mechanisms are overweight or unnecessary for typical enterprise data-access applications. Recent work by Phatak and Badrinath [23] and Stanoi *et al* [27] consider data partitioning and data warehousing for mobile enterprise data access. These systems indeed address some server side issues for mobile enterprise data access but are not end-to-end systems. The DataX system by Lei *et al* [17] addresses targeting and subsetting data for clients, and rendering data on clients; it complements our synchronization framework.

Various systems use the notion of common data interchange formats to deal with heterogeneity. The Abstract Syntax Notation defines a common data interchange format in the domain of digital networks and public data networks [13]. Elmagarmid *et al* explore the use of intermediate languages in heterogeneous database systems [4]. Drashansky *et al* [8] use intermediate data formats in the area of mobile scientific computing. Although MDSP is similar to such existing approaches, its focus is mobile data synchronization and it substantially differs in scope from the other approaches.

The industry also offers a number of data replication and synchronization systems. Systems such as Palm Computing's HotSync [22] and Microsoft's ActiveSync [20] are geared only toward synchronizing PIM⁵ data with the desktop and are each targeted to a single client platform. Systems such as Advance Systems' ASL Connect [1], AvantGo's AvantGo.com [2], Puma's Intellisync [24], Riverbed's ScoutMTS [26], Synchrologic's RealSync [29], WaveWare's WaveSync [31], Oracle's Oracle Lite 8i [21], and Sybase's SQL Anywhere [28] provide some form of enterprise data-access from handheld devices. The key difference between these systems and ours is our focus on interoperability, which is realized by our three-tier architecture and an open synchronization protocol. We also believe that our use of efficient replication support (already provided by many databases) on the server side yields performance and scalability benefits over the ODBC/JDBC approach used in many of the above systems. Oracle's Oracle Lite 8i [21]

⁵Applications such as calendar, address book, etc. that are found on Palm Pilots or Windows CE devices are called Personal Information Management (PIM) applications.

has a hybrid connected/disconnected model that it requires the client to run a web server and Java servlets when connected. We speculate that this approach is unsuitable for space-constrained devices. Systems like Lotus Notes [18] and Microsoft Exchange [19] provide replication facilities but do not scale down for small handheld clients and do not address interoperability issues. Likewise, synchronization protocols such as IrMC [14] and MAL [3] are data type and/or transport specific. They do not interoperate in a heterogeneous environment.

6. Conclusions and Future Work

We have designed and implemented a complete end-to-end enterprise-data-access framework. We have identified the key technical challenges in this domain and addressed them. We believe that interoperability is the strongest feature of our system. The three-tier adapter-based model, the common data interchange format, and the open synchronization protocol are the cornerstones of interoperability in the system. We have addressed footprint, connectivity, and power issues in designing the object-based and relational clients; addressed efficiency and reliability concerns in designing synchronizers; and addressed performance and scalability issues in designing the replicator-based adapter.

We have a first-cut implementation of the system. Measurements and performance tuning must follow to validate our design choices. Implementation of other adapters (e.g. Lotus Notes and Microsoft Exchange) will likely provide valuable insight into server-side design issues for non-relational systems. Although we provide basic authentication and access-control, we must address security more comprehensively, including encryption of data on the network and client devices. Design enhancements such as multiple MDSP messages for one phase of synchronization, flexible message delivery priorities, support for advanced query mechanisms and transactions for the relational client, and query and indexing support for the object-based client are under consideration.

References

- [1] Advance Systems. *Connecting Mobile Workers to Enterprise Information Systems*. <http://www.asl.com>, 1999.
- [2] AvantGo Corporation. *AvantGo Developer Guide*. <http://www.avantgo.com/DevCorner/DevGuide>, 1999.
- [3] AvantGo Corporation. *Mobile Application Link*. <http://www.mobilelink.org>, 1999.
- [4] A. Bougettaya, B. Benatallah, and A. Elmagarmid. *Interconnecting Heterogeneous Information Systems*. Kluwer Academic Press, 1999.
- [5] Bruce Martin and Bashar Jano, editors. *WAP Binary XML Content Format*. <http://www.w3.org/TR/wbxml>, 1999.

- [6] N. H. Cohen. *Application Programmer's Guide to Mobile Network Computing Data Synchronization*. http://www.oadg.or.jp/activity/mncrs/dsync/pgmguide/tutorial-1_1.pdf, 1999.
- [7] S. Davidson, H. Garcia-Molina, and D. Skeen. Consistency in a partitioned network: A Survey. *ACM Computing Surveys*, 17(3):341–370, September 1985.
- [8] T. T. Drashansky, S. Weerawarana, A. Joshi, R. A. Weerasinghe, and E. N. Houstis. Software architecture of ubiquitous scientific computing environments for mobile platforms. *ACM Journal on Mobile Networks and Applications*, 1(4), 1996.
- [9] C. F. Goldfarb and P. Prescod. *The XML Handbook*. Prentice Hall, 1998.
- [10] International Business Machines Corporation. *Data Replication Solution*. <http://www.software.ibm.com/data/dbtools/datarepl.html>, 1999.
- [11] International Business Machines Corporation. *DB2 Everywhere*. <http://www.software.ibm.com/data/db2/everywhere>, 1999.
- [12] International Business Machines Corporation. *The IBM MQSeries Family*. <http://www.software.ibm.com/mqseries>, 1999.
- [13] International Telecommunications Union. *Series X Recommendations — Data Networks and Open System Communications*. <http://www.itu.int/itudoc/itu-t/rec/x/index.html>, 1999.
- [14] John Stossel, editor. *Specifications for the IR Mobile Communications (IRMC)*. <http://www.irda.org>, 1999.
- [15] J. J. Kistler and M. Satyanarayanan. Disconnection operations in the Coda file system. *ACM Transactions on Computer Systems*, 10(1):3–25, February 1992.
- [16] P. Kumar and M. Satyanarayanan. Supporting application-specific resolution in an optimistically replicated file system. In *Proceedings of the fourth workshop on Workstation Operating Systems*, pages 66–70, October 1993.
- [17] H. Lei, K. Lee, M. Blount, and C. Tait. Enabling ubiquitous database access with XML. In *Proceedings of the first International Conference on Mobile Data Access*, December 1999.
- [18] Lotus Development Corporation. *R5 Notes Home*. <http://www.lotus.com/home.nsf/tabs/lotusnotes>, 1999.
- [19] Microsoft Corporation. *Microsoft Exchange Server*. <http://www.microsoft.com/exchange>, 1999.
- [20] J. Murray. *Inside Microsoft Windows CE*. Microsoft Press, 1998.
- [21] Oracle Corporation. *Oracle8iLite*. <http://www.oracle.com/mobile/o8lite/index.html>, 1999.
- [22] Palm Computing. *Development Documentation*. <http://www.palm.com/devzone/docs.html>, 1999.
- [23] S. H. Phatak and B.R.Badrinath. Data partitioning for disconnected client server databases. In *Proceedings of the workshop on Data Engineering for Mobile and Wireless Access*, pages 102–109, August 1999.
- [24] Puma Technology. *Puma Technology Developer Zone*. <http://www.pumatech.com/developer>, 1999.
- [25] D. Ratner, G. J. Popek, and P. Reiher. Peer replication with selective control. In *UCLA Technical Report CSD-960031*, July 1996.
- [26] Riverbed Technologies. *The Freedom to go Mobile*. <http://www.riverbedtech.com>, 1999.
- [27] I. Stanoi, D. Aggarwal, A. E. Abbadi, S. H. Phatak, and B.R.Badrinath. Data warehousing alternatives for mobile environments. In *Proceedings of the workshop on Data Engineering for Mobile and Wireless Access*, pages 110–115, August 1999.
- [28] Sybase Corporation. *Sybase Mobile and Embedded Computing*. <http://www.sybase.com/mec>, 1999.
- [29] Synchrologic Corporation. *Solving the Mobile Computing Challenge: Data, Documents, and Software Distribution to Mobile Users*. http://www.synchrologic.com/images/whitepapers/mobile_computing_whitepaper.html, 1999.
- [30] 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 System Principles*, pages 172–182, December 1995.
- [31] Waveware Communications Inc. *Wavesync: The Premier Synchronization Server*. <http://www.waveware.net/syncindex.htm>, 1999.
- [32] M. H. Wong and D. Aggarwal. Tolerating bounded inconsistency for increasing concurrency in database systems. In *Proceedings of PODS*, pages 236–245, 1992.