XMIDDLE

Information Sharing Middleware for a Mobile Environment

Stefanos Zachariadis

Dept. of Computer Science

University College London

http://www.cs.ucl.ac.uk/staff/s.zachariadis/

Joint Work with Licia Capra, Giles Taylor, Cecilia Mascolo and Wolfgang Emmerich.

Motivation

- Proliferation of mobile devices
  - Limited processing power
  - Limited memory
- Built in networking hardware
  - e.g. 802.11b, Bluetooth, IrDA
  - Error prone, battery hungry, (some are) short ranged, perhaps expensive (GSM modems)
- Need to share information with other hosts
  - From classical PDA synchronisation paradigms (e.g. PalmOS hotsync)
  - Exploitation of limited peer resources

Motivation (2)

- Problems with existing middleware
- Fixed network (wired) infrastructure middleware too heavy
- Existing mobile networking middleware problematic
  - Bayou (Xerox PARC): mobility is an exception
  - Lime & TupleSpace Based Systems: unstructured data

The Goal

- Provide a middleware for data sharing in mobile computing environments, supporting disconnected operations through replication and providing for the dynamic nature of ad hoc networks
- Provide a framework for the automatic reconciliation of shared information
- The application should be able to guide the middleware as to how to resolve conflicts

The XMIDDLE Approach

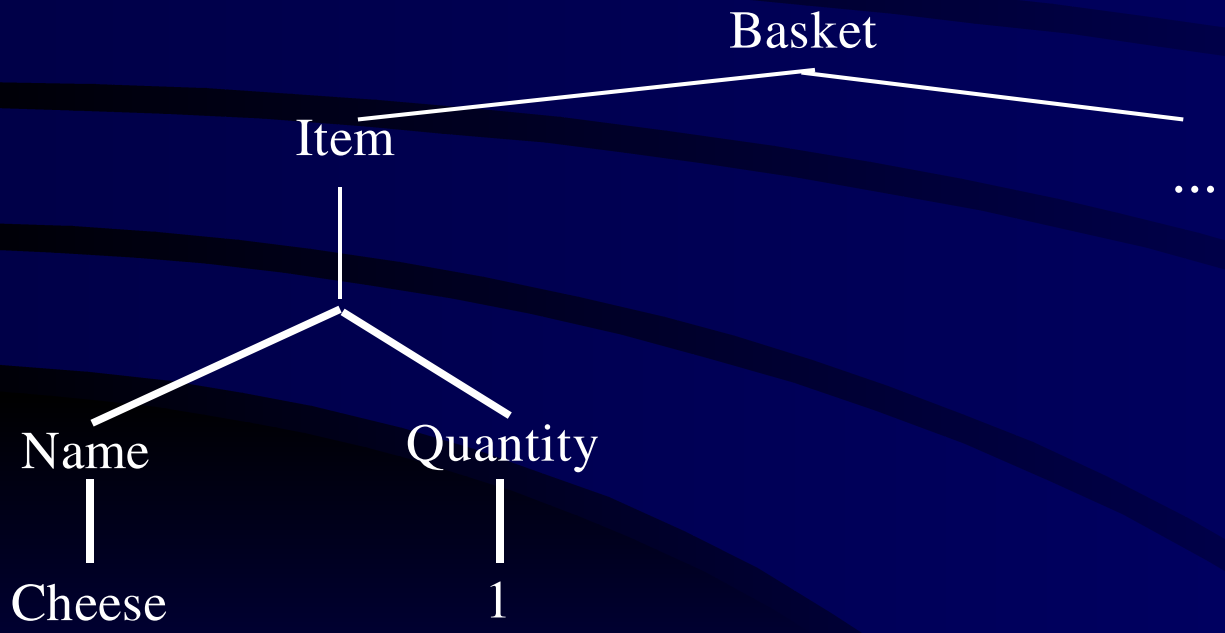
- Completely decentralised (peer to peer) operation
- Geared for non-continuous & low bandwidth networking hardware
- Fine-grained application control of what is being shared
- Concurrently supporting multiple applications
- Applications store their data in XML format (i.e. a tree)
- Provides an API to access data, hiding any replication information
- Lightweight (~120KB compressed)

A Simple Example

- A household, with one home computer (A) and a PDA (B), both running XMIDDLE and an e-shopping application.
- Connected, when in reach, via a wireless link.
- Family wishes to use both A and B to do their shopping

A's Data

- Encoded as XML



Exporting and Linking

- A *exports* basket.
  - Data available for sharing
  - Can either export just to B or globally
- B *links* to the exported branch
  - The data are copied to the local shopping tree and are readily available to the application

Remote Access

- A & B are separated
- Both have access to the catalog and the shopping basket
- B changes quantities and adds another item...

Reconciliation

- A & B come in reach again.
- The middleware detects that they are sharing data
- It determines that there have been changes to the data
- A diff of the changes is computed and transferred.
  - Any conflicts are resolved via the use of policies
- A & B end up with the same consistent tree

Complexity

- Scenario can be more complex
- Multiple hosts sharing information
- Hosts reconciling shared data even if none of the involved hosts “own” the data
- Larger data
- Online collaboration

Basic Assumptions

- Availability (non constant) of networking hardware (TCP/IP stack) and network connection
- Each application stores its data on an XML document, accessible via DOM operations (tree based format)
- XML validation a non-issue for the middleware (disabled).
  - Deemed an application issue.
  - Can be enabled (work in progress)

Sidenote: Why DOM?

- Fast random access to all XML Elements
- Tree structure
  - Easy to manipulate and traverse
  - Mature algorithms exist
- Data are kept in main memory, but in PDAs, data are kept in main memory in any case!

XMIDDLE Primitives

- **Connection**: hosts can be connected to/participating in the ad hoc network, allowing them to share data and to trigger the transparent data reconciliation process
- **Disconnection**: hosts can choose to explicitly disconnect from the network, in which case they will not have access to the networked facilities that the middleware provides, although they will still be able to access shared data that has already been replicated. A host can explicitly request disconnection, to conserve battery power, etc.

XMIDDLE Primitives (2)

- **Exporting**: applications can specifically define which local data elements they allow to be shared with other devices, by exporting elements of their data tree.
- **Linking**: In order to share information, applications need to link to data exported by applications in other hosts. Once linked the middleware allows disconnected operation on that data, and the reconciliation of changes when connected & in reach
- **Unlinking**: data can also be unlinked when sharing and reconciliation is not needed anymore

Hosts

- Hosts are defined by two ids
  - PrimaryID: Assumed to be unique. Used to identify hosts.
  - SecondaryID: Human-readable identification of a host (does not have to be unique)
- Receive information in two ways
  - IP Multicast (for information broadcast to all hosts)
  - UDP Sockets (each host has a socket open for receiving information specifically addressed to it)

Versioning & Reconciliation

- In order to reconcile information, XMIDDLE uses the concept of versioning
- A version is a "snapshot" of the shared data tree.
  - Can either be a full snapshot, or a diff pointing to a previous version
- The first version (version 0) is created when a branch is first exported by the owner, and is always transferred to the host that links to it (along with the latest version available).
  - So all hosts sharing a branch have at least one common tree to reconcile data from

Versioning & Reconciliation (2)

- New versions are released when two hosts reconcile a branch that they are sharing
  - The “snapshot” of the reconciled branch
- Each version has a version identifier attached, which contains three elements:
  - The PrimaryIDs of the two hosts involved, and a version number starting from 0 (when the branch is first exported)
- The hosts reconciling the branch automatically compute the same version identifier
- Reconciliation always starts from the most recent (i.e. largest version number) version common to both hosts

Installing Applications

- Applications must *register* with the *Application Manager* in order to have access to the services of the middleware.
- When registering, they are assigned a unique ID (integer).
  - Used to distinguish which application is exporting which element
  - IDs are unique on each platform – The same application can be assigned a different ID when installed on a different platform

Installing Applications (2)

- The Application Manager creates an *Application Profile* for each application.
- The application accesses its data via the Application Profile through the Application Manager

XMIDDLE, DOM and XPath

- Exported elements are encoded using XPath
  - Advertising and requesting such elements is done via XPath
- XMIDDLE provides an API where applications can request local data using XPath and are returned DOM Elements

Peer Discovery

- Upon *connecting* to the network, XMIDDLE hosts join an IP Multicast group
- Send information like their ids, their exports, items they're linking to, private port number…

```
<online><primaryID>1352591906</primaryID><secondaryID>musakas</secondaryID><address>128.16.10.65</address><port>1439</port><segmenSize>2048</segmentSize><linktable><export><appID>0</appID><branch>/basket</branch></export></linktable><services></services></online>
```

- Amount of data currently sent needs optimisation
- Other hosts detect and analyse this
  - If they stop receiving it, they consider that the host is off-line or out of reach

Protocols

- Reconciliation, Linking (can be extended)
- The host that wishes to initiate a session (e.g. Linking) with another host, sends a protocol request to the UDP socket of the latter…

  - For reconciliation this is done automatically
- Both hosts then launch what we call the ProtocolChooser…

Protocols, the ProtocolRegistry & the ProtocolChooser

- What is a protocol?
  - Linking, reconciliation… But there can be many ways to do those, depending on the context etc…
  - How about new ideas such as simple copying of data?
- On a basic level, a Class implementing a specified Protocol interface
- Protocols need to *register* with the XMIDDLE ProtocolRegistry

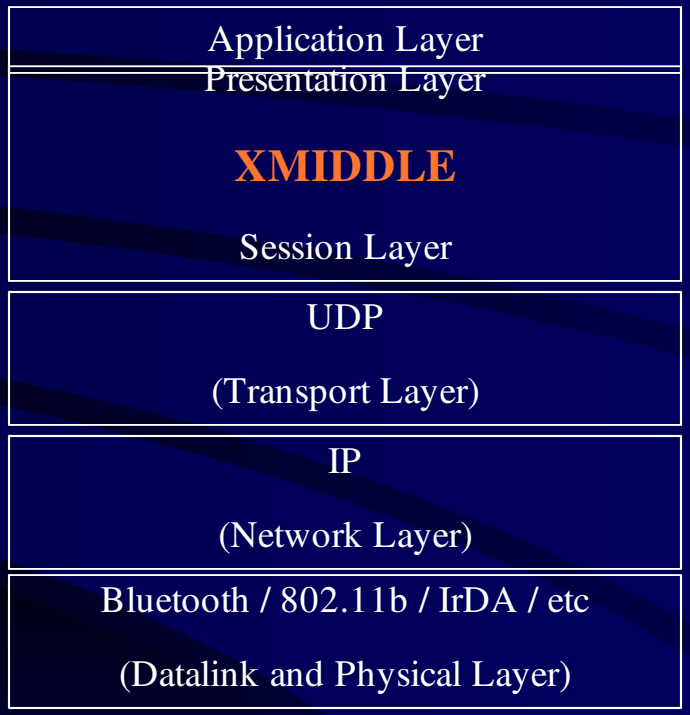
Protocols, the ProtocolRegistry & the ProtocolChooser (2)

- ProtocolRegistry is a repository of Protocols (algorithms)
- Protocols have a unique (String) id.
  - e.g. LINK
- They register with the ProtocolRegistry giving their full class name and their id.
  - e.g. edu.UCL.xmiddle.lib.protocols.Linking

Protocols, the ProtocolRegistry & the ProtocolChooser (3)

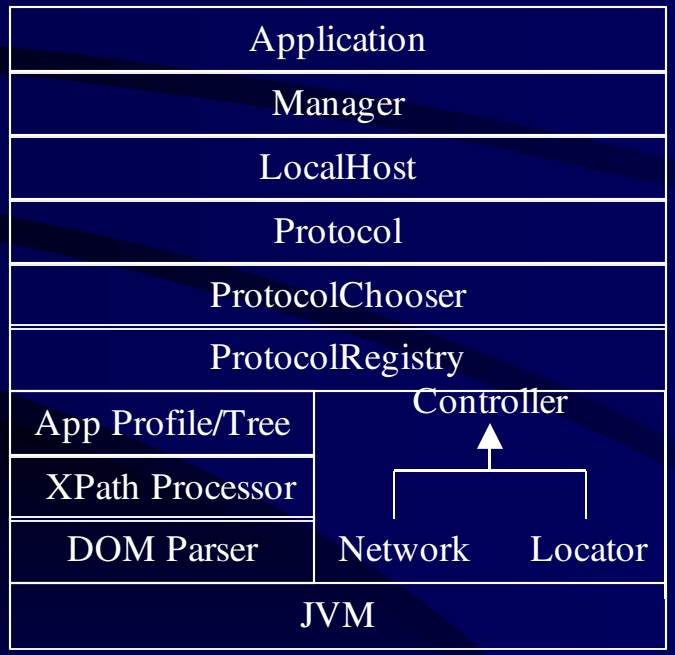
- When the ProtocolChooser is launched, it checks the registry for the specified protocol.
  - If it is not found then it has the option of requesting the given protocol from the other host
  - Dynamic code update
- It then negotiates dedicated UDP sockets (which we call Listener and Sender objects) for communicating directly with the other host for this session
  - Faster processing, possibilities of encryption etc
- Synchronises the hosts involved and
- Finally, using the Java Class Loader, it launches the Protocol giving it any parameters and data it requests

XMIDDLE and the Protocol Stack



Brief Overview of Architecture

- Modular & Reusable



Some Numbers

- XMIDDLE has been installed on Compaq iPAQ H3660 (64MB memory) running Linux (familiar 0.4+, kernel 2.4.3+)
  - Intel StrongARM CPU (206MHz)
- Lucent Orinoco Silver cards (802.11b)
- Java (1.3.1-rc1 from Blackdown ~15MB) & XML Parser & Xpath Processor(Xerces 1.4.1, Xalan 2.2.D6, ~2.5MB)
  - No JIT for ARM (yet)
- XMIDDLE (~125KB)

Evaluation

- Biggest bottlenecks were found to be Apache Xerces and Xalan.
- Fast but can become faster
  - e.g negotiation of UDP sockets for transferring information during protocol sessions needs to be faster
- Ongoing work to port to J2ME and Pjava
  - Also considering porting to NanoXML or KXML

Future Work

- Reconciliation of reconciliation policies!
- Dynamic Code Update
- Access rights
  - Framework already exists
- Optimisation