PEMOCO: AN INFRASTRUCTURE FOR PERSONAL MOBILE COMMERCE USING JAVA ENABLED SMART PHONES

Tapan Divekar, Sumi Helal and Steve Moore

Computer and Information Science and Engineering Department,
University of Florida, Gainesville, FL 32611
helal@cise.ufl.edu

ABSTRACT
This paper explains the architecture of PEMOCO which is a middleware infrastructure developed on Java enabled smart phones for the emerging field of Mobile Commerce. We developed a micro HTTP server be used as a service gateway for buying, selling and auctioning tickets. We also developed a web publishing agent on the phone to provides an easy way to create tickets and contents on the phone web server. We present the architecture and implementation of PEMOCO. We present a case study that demonstrates PEMOCO and the importance of location awareness and location naming schemes in Mobile Commerce.

KEYWORDS
Mobile Commerce, Location-based services, mobile servers, Smart Phones, J2ME.

1 Introduction

The emergence of wirelessly connected portable devices has given the power of ubiquity to users, with anytime anywhere access to information and services. These devices have brought computing and information closer to ordinary users, which enable them to transact on e-commerce applications right off their personal devices. Such transactions have been termed *Mobile Commerce (MC)* by many [1]. It is anticipated that commercial services catered to people on the move will constitute a multi-billion dollar industry within the next couple of years.

Location awareness play an important role in differentiating services from one another in MC. Discovering services for a user in a new location [2][3] or any specified position is one of the key requirements for MC.

In this paper we report on a MC Middleware infrastructure that we have designed and implemented to enable the buying, selling and auctioning of events tickets using Java-enabled smart phones. The middleware, which we call Personal Mobile Commerce, or PEMOCO (a pun on DoCoMo, which in Japanese means anytime, anywhere), includes a small version of an HTTP server on the mobile phone (Motorola *i85*s phones [4]), which serves as a universal interface for MC applications. In addition to being an MC enabler, the HTTP server can be used to enable the emerging Peer-to-Peer Computing paradigm. We have used XML [5] based storage structures for storage and building of tickets.

The paper covers the broad level architecture of the PEMOCO framework followed by the design of each of the components involved. In this paper we have assumed football ticket as an MC entity. We also cover the performance analysis of the micro HTTP Server and wrap up with the roadmap to the future.

2 Design

Figure 1 illustrates the conceptual framework of our infrastructure. The phones depict the sellers and buyers who are using their PEMOCO infrastructure to choose between i) Micro HTTP Server ii) Web Publishing Client, iii) Auctioneer and iv) MyTicket applications.

The sellers (at location x), first get onto the Web Publishing client to create their ticket using user friendly tags. The Auctioneer application on seller's phone is used to sell the tickets.

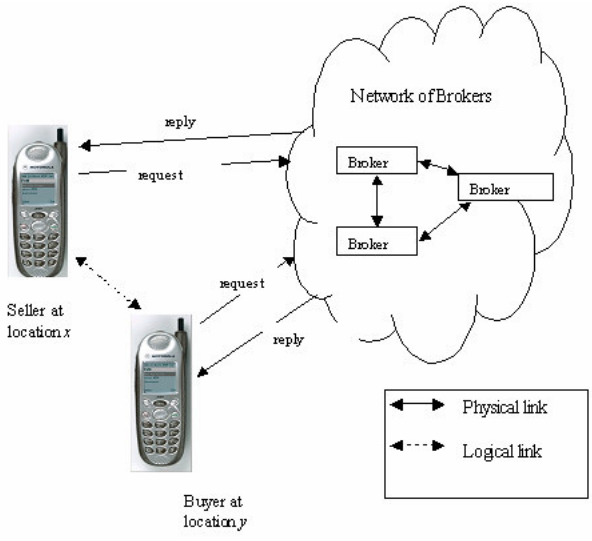


Figure 1: Conceptual Framework

The brokers run on a well-known server on a wired network. The tickets sold by the sellers are stored in the broker's persistent store.

The buyers (at location y) use the Auctioneer application making a query to the local broker including the necessary location dependent information. Whenever the buyers query for a ticket, the local broker contacts the broker in the area specified by the buyer to find if the ticket is on sale.

Buyers have the ability to create a wish ticket and they will be notified whenever a ticket of their choice is being sold. This is done using the MyTicket application. The local broker stores this information and notifies the buyer when an appropriate match for the ticket is found. The micro HTTP Server built on the phones facilitates the buyers to receive tickets from the broker.

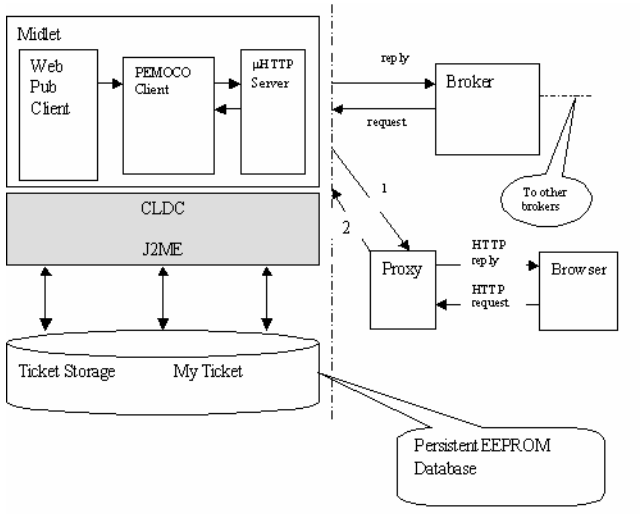


Figure 2: Overall Framework

Figure 2 depicts the overall framework of PEMOCO. The left hand side of the figure is the client side (smart phone) and the right hand side is on a fixed network. The above figure shows the overall architecture of PEMOCO. The client contains the Web publishing client, the micro HTTP Server that is built over CLDC. The client also contains persistent storage for tickets and My-Ticket. Browsers can access the smart phone using a proxy as shown above.

The following subsections discuss the individual components in detail.

2.1 The Micro HTTP Server

We have designed a small HTTP[6] server on the smart phone. The J2ME implementation supports a limited number of APIs for networking.

The current implementation of i85 [4] doesn't support server side sockets. This absence of server side sockets has spawned a need to develop a new kind of protocol, called as U-H protocol.

HTTP is placed over TCP in the protocol stack. In our design we have embedded HTTP packets over UDP. U-H stands for HTTP packets over UDP.

It has been shown [7][8] that TCP does not perform well for smaller files. Instead, UDP is proved to have a better performance, since it does not require opening and closing a connection, like TCP.

2.1.1 Communication Details

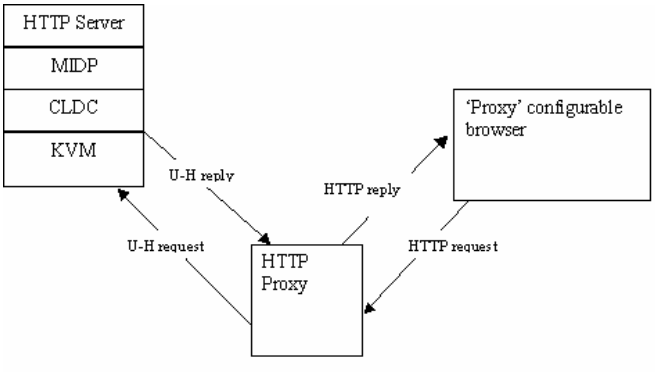


Figure 3: Communication details (Client is a proxy enabled browser).

Figure 3 cites the communications that take place between the browser and the phone. The browser should have the capability to manually set the HTTP proxy server's address. The HTTP proxy takes the browser's request and converts it into a U-H request, which the micro HTTP Server can understand. The proxy also receives the HTTP Server's U-H reply and sends it to the browser. The handshake protocol is discussed in the next section.

2.1.2 Handshake

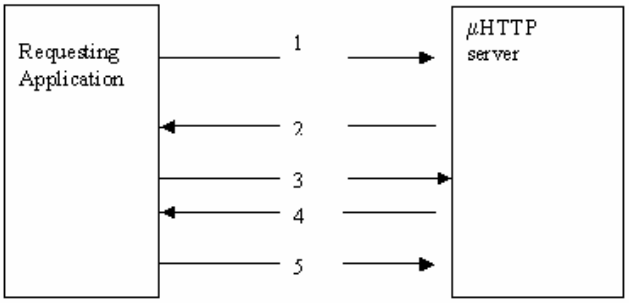


Figure 4: U-H Protocol Sequence

The following are the sequence of operations that take place in the U-H protocol:

1 The requesting application makes an UDP request to the micro HTTP Server. This is the initial CONNECTION request.

2 The micro HTTP Server replies with a REPLY CONNECTED message.

3 The application makes a REQUEST SESSION to the micro HTTP Server.

4 The micro HTTP replies with RESPONSE SESSION back to the application.

5 The application makes a FINISH request to the micro HTTP Server.

5.1.1 Opcodes

In this architecture we have used Opcodes to define requests and replies. We have based our design on IrDa's OBEX [9] protocol.

Table 1: Opcodes

Opcode	Definition	Meaning
10	Connect	Establish Connection
20	Reply	Reply connect
30	Get	Get parameters
40	Inform	Send data to the Server
50	Response	Response to the request.
60	Finish	Abort connection

2.2 Web Publishing Client

Along with the micro HTTP Server, we have developed a small publishing client, which will allow users to make web pages as well as tickets using user-friendly tags. Mainly used by the sellers it provides user friendly tags which can be used to create tickets easily and quicker. A user has the ability to add custom tags and can then add the tags to the list of predefined tags. In case of tickets the common tags are the name of the game, the participating teams and the location of the game. A user can also store multiple types of tickets or pages. Using the Auctioneer application the seller browses the file names to choose the right ticket to sell and then the ticket is sent to the brokers for sale.

2.3 Auctioneer

Auctioneer is the application used by both the sellers and the buyers. Using the buy operation in the Auctioneer the buyer can query for availability of tickets. The buyer creates a location based query using any of the four location notations described in section 2.4.1 to search for available tickets.

The sellers use the sell option of the Auctioneer and choose the file created using the Web Publishing client to publish the ticket to sell to the broker.

2.4 Location Awareness

Location [10][11] is an important ingredient in the domain of mobile computing and moreover in the MC domain. Various location aware schemes have been developed.

We define location space as a data model that can adequately represent locations of fixed and mobile objects. We can have two ways of naming this model: One is an n dimensional co-ordinate system – geometric and another is symbolic which can be used to specify relationships.

The symbolic model uses an abstract way to represent objects and uses relationship between different entities. The advantage of using symbolic naming scheme is its ability to access locations by name. This facilitates location awareness and access control.

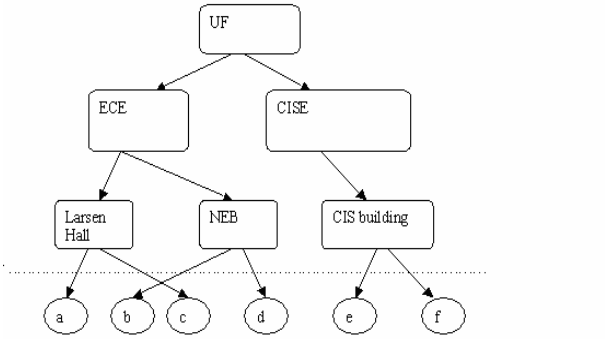


Figure 5: Hierarchical Representation of Symbolic Models

Figure 5 is an example of Symbolic Models. The Sybolic Model depicts a "contained within" relationship for a particular location. For example: Larsen Hall lies in the ECE (department) inside UF.

In the Geometric Model, locations are represented as points, areas or volume within the co-ordinate systems. Such locations are described by sets of co-ordinate tuples. Geometric models are advantageous as far as accuracy of information unless there is loss of data during conversion from one co-ordinate system to another.

At present we mainly use combination of the symbolic and the geometric models to give the users the choice of selection of a variety of locations taking advantage of the naming scheme of symbolic models as well as the accuracy of the geometric models.

5.4.1 Notations for Locations

It is essential to develop a notation [10] for the textual representation of the above addressed naming schemes.

The notations [10] defined shall be consistent and be easily understood by a common user. The naming schemes should be hierarchical, consistent to represent mobile objects, which are changing locations frequently. Also they should be able to represent abstract locations and geometric positions. We can use the following expressions to represent simple locations.

The symbolic naming schemes could be specified in the form of a well-known position or either hierarchically. Geometric position can be specified in the form of well-defined co-ordinates. Common examples are E309 @Gainesville/UF/CIS, <5m@Gainesville/UF or <1m@WGS: 84(0.04 W, 51.3 N, 0).

We have used four naming schemes in our application. The absolute naming scheme is used to specify a particular location with respect to a well-known location. For example

CIS department in UF, Gainesville could be specified as Gainesville/UF/CIS.

The contained within naming scheme allows the buyer to specify a certain perimeter with respect to a well-known location. This naming scheme can be best expressed assuming our geographical unit is a city. In a city like Gainesville avenues run parallel to streets and this forms a matrix like structure when seen from the top. When a user specifies the boundaries of the areas we build a rectangle which encompasses the area of choice. We then define closeness factors, which will be used by the brokers to pass on requests depending on requests specified by the buyers.

Another naming scheme is used to specify a region. A common example is say the northern part of a city. Example North Gainesville can be considered a covered area for this particular schema.

The last naming called 'All' is used to select all the brokers. This will be used by the buyers if they want to increase their chances of finding tickets which may be at the cost of long distances.

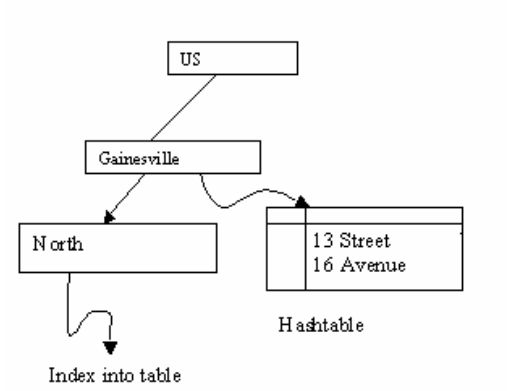


Figure 6: Data Structures used for the Naming Conventions

The above tree is the way the naming scheme is internally handled. For example to represent Gainesville/13Street in absolute naming scheme we traverse the tree until Gainesville from the root node and then hash into the hash table. This table provides index into the metadata table, which gives information about broker addresses. Information whether the location is on a row or column is also provided.

For specific area locations like North Gainesville, 'North' provides a table with indices to specific locations in a table.

2.5 Transactions

The following subsections explain the transactions between different members of our MC infrastructure.

5.5.1 Seller - Broker

The sellers use the Auctioneer application to sell the tickets. The Auctioneer application sends to the broker the ticket in XML format. The broker uses the B-tree storage structure to store the tickets.

5.5.2 Buyer - Broker

The buyer gets on the Auctioneer application to buy a ticket. The buyer send the following information to the broker - the city(base geographical unit),the Street (or *All* or *Zone* depending on Naming Scheme used), the distance(if contained within naming scheme is used), the game and the participating teams.

5.5.3 Broker - Broker

When the buyer queries the broker for a seller in a particular location, the broker parses the XML data and searches the data structures for the target broker addresses. If it needs to contact other brokers, then it queries the target brokers for the required ticket. The broker also does some match making to see how close the stored ticket is to the requested ticket.

2.6 Ticket representation

We have used XML to store tickets. A typical ticket looks like this:

```
<title> Ticket </title>
<ticket>
<type> BasketBall </type>
<teams>Gator vs Seminoles</teams>
<date> 31st Aug,2001 </date>
<place>Gainesville, Fl </place>
<price>$50</price>
</ticket>
```

We have surveyed through some of the storage structures for XML [12] data like the EDGE based approach, the Object based approach and the B-tree based approach. B-tree based approach is found to be the most efficient one and hence we have used it to store the tickets.

The first level of the tree designates the game. The next level stores all the tickets.

2.7 My-Ticket

This is a ticket based on the push-based approach using the micro HTTP Server depending on the buyer's choice.

A buyer makes the choice of the game details using the web-publishing client and then uploads the ticket to the local broker. The broker has an entry for every buyer. The buyer will then switch on his micro HTTP Server and wait for any tickets.

When the buyer starts his micro HTTP Server the broker will make the logical status of the buyer to 'ON' signifying that the buyer can accept messages. This will be turned 'OFF' when the broker stops the micro HTTP Server. When a ticket is sold to the broker will check if there is a matching buyer for the ticket and will notify the buyer on his micro HTTP Server using the INFORM message (opcode 40) format.

6 Performance

This section covers the various performance issues [13] [14] related to the micro HTTP Server.

A server's performance is evaluated on the basis of reliability and scalability. We have evaluated the performance simulating an environment consisting of multiple clients on different machines. There were many difficulties in simulating clients on different machines. Moreover there are differences in delay between the delay and loss characteristics of the LANs and WANs used in test beds.

A strategy using minimal number of client machines to simulate actual web traffic is proposed, by having C clients on N machines. In this schema the client repeatedly sends requests to the server and then waits for a certain amount of time and then sends another. The factor N is kept as high as possible and C as small as possible. While generating these requests from the client's end care must be taken such that the resource limitations on the client side do not affect the performance of the server. As the number of requests from a single machine increase, memory and CPU might be a bottleneck. There could be a stage in the performance evaluation where the client and not the server could be a bottleneck. These client bottlenecks could be avoided by reducing the number of client processes per machine.

3.1 Server Design I

A typical web server listens for requests on a particular port (normally port 80 for HTTP protocol) and opens an active connection to communicate with every client. The Web Server has no limitations on the number of connections open, unless we run out of memory to support each connection. The design for our micro HTTP Server has to be slightly changed since it supports only two active connections at a time.

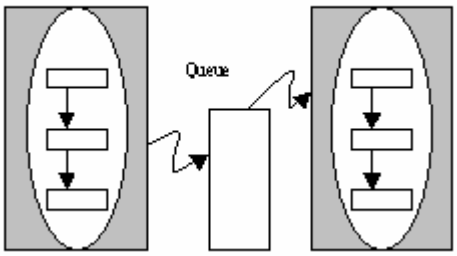


Figure 7: Server Design I

The micro HTTP Server has a Main Thread, which continuously listens for requests and puts them in a queue. The length of the queue is fixed. The service handler thread waits for messages to be put in the queue and then handles the handshake protocol with the client. If the size of the queue is full, then a negative acknowledgement is sent back to the client. This helps the client in knowing that its request cannot be served. The object Queue is shared between the Main Thread and the Service Handler Thread and the methods are synchronized to have serial update of the shared data.

3.2 Server Design II

With more number of simultaneous requests we observed some packets being dropped and the service handler thread being unnecessarily waiting for packets from the client. This lead to a deadlock state where the service handler thread could not serve any more requests. We thus had a modified Service Handler Thread as shown in design II below.

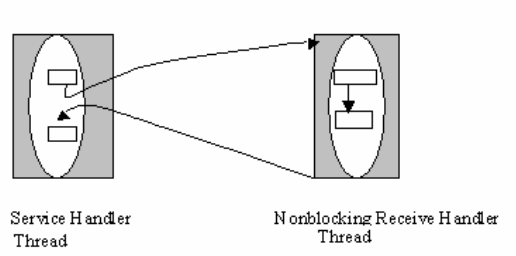


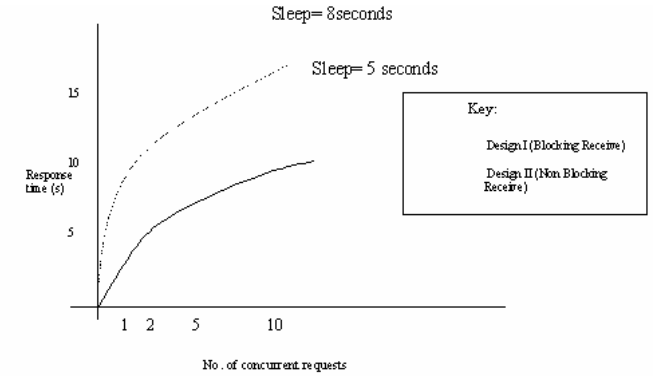
Figure: 8 Server Design II. The Main Thread is same as in Design I

We designed the non-blocking receive handler which was spawned after the server had sent its acceptance to the request from the client and was now expecting to receive a query from the client (phase 2 of handshake protocol) from the client. The service handler thread would wait for a specified amount of time to wait for data to arrive. The above design was basically an implementation of a non-blocking receive. The communication was much more reliable but at the cost of higher communication time.

3.3 Results

The response time increases proportionally to the number of the requests. The response time for the Nth request is more compared to that of the (N-1)th request.

The graph is for 5 clients simulated on the same machine with 5 requests per client and an interval of 1second between them



As seen from the above graph the response time increases with the increase in the request number. The first request is

immediately served since the queue is empty. Thereafter as the queue fills up the response time also increases.

7 Conclusion and Future

This paper defined an infrastructure and middleware for enabling Mobile Commerce applications for smart phones.

We have built the micro HTTP Server along with a new U-H protocol and analyzed its performance with two different designs. The Web Publishing Client was used for creating web pages and tickets on the fly and the users used the Auctioneer application to sell as well as buy the tickets. We have designed and implemented the Auctioneer application enabling location dependent queries for buying using the phone.

The brokers could be expanded to serve not only requests for tickets but could act like a directory server giving public information to users. The brokers can serve requests for any MC items. The Web Publishing client can be expanded to smartly remember tags based on user use.

The micro HTTP Server could be used in ad-hoc information retrieval and could be seen a tool for collaborative environments thereby allowing buyers to communicate each other. The brokers can be seen as agents used to initiate communication or collaboration between different users of the infrastructure.

Peer-to-Peer computing [15] has emerged and analysts prove its advantages over client-server computing. Sun Microsystems project JXTA [16] deals with peer-to- peer computing. The micro HTTP Server could be used to exploit this area of computing.

Security could be added to the above framework to facilitate transactions.

The above features if incorporated will certainly make a mark in tomorrow's world and will truly justify the evolution of mobile computing.

References

[1] F. Müller-Veerse, "Mobile Commerce Report, (pdf)," Durlacher Corp., London Appeared in October 2000 issue of IEEE Computer

[2] Harry Chen, Anupam Joshi, Timothy Finin: "Dynamic Service discovery with mobile computing: Intelligent Agents meet Jini in the Aether ", Department of Computer Science and Electrical Engineering, University of Maryland at Baltimore County, February 2000.

[3] Reticular Systems, Incorporate: "Using Intelligent Agents for Wireless E-commerce Applications: The Yellow Stone project": Reticular Systems Incorporate, May 2001.

[4] Motorola Developer Community Web Site: http://www.idendev.com.

[5] T.Bray, J. Paoli and C.M. Sperberg – Mc Queen "Extensible Markup language (XML) 1.0", W3 recommendation: http://www.w3.org/TR/REC-xml, February, 1998.

[6] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee, "Hyper Text Transfer Protocol – HTTP/1.1 ". RFC 2068, January, 1997.

[7] B. Brown, M.S. Computer Science, U.C. San Diego, Professor J. Pasquale, Ph.D. U.C. San Diego: "U-HTTP: A high performance UDP- based HTTP".

[8] M. Rabinovich, AT&T Labs –Research, Hua Wang, New York University: "DHTTP: An Efficient and Cache Friendly Transfer Protocol for Web Traffic."

[9] Counterpoint Systems Foundry Inc, Microsoft Corporation "IrDA Object Exchange Protocol" IrOBEX Version 1.2, March 1999.

[10] Ulf Leonhardt, "Supporting location awareness in Open Distributed systems" Department of Computing, Imperial College of Science, Technology and Medicine, University of London, 1998.

[11] Y. Charlie Hu, Daniel A. Rodney and Peter Druschel "Design and scalability of NLS, a scalable Naming and Location Service", Computer Science Department, Rice University, TX, USA, Submitted for publication, November 2000.

[12] Feng Tian, David J. DeWitt, Jianjun Chen, Chun Zhang: "The design and performance evolution of alternative XML Storage Strategies": Department of Computer Science, University of Wisconsin-Madison, WI, June 2001.

[13] Gaurav Bagga and Peter Druschel: "Measuring the capacity of a Web Server". Department of Computer Science, Rice University, Houston, Texas. Accessed June 2001.

[14] Arun Iyengar, Ed Mac Nair and Thao Nyugen. "An Analysis of Web Server Performance", IBM Research Division, T. J. Watson Research Center,NY. Accessed June 2001.

[15] Gregory Alan Bolcer, Michael Gorlick, Arthur S. Hitomi, Peter Kammer, Brian Morrow, Peyman Oreizy, Richard N. Taylor. "Peer-to-Peer Architectures and the Magi ™ Open-Source Infrastructure". Endeavors Technology, Inc. http://www.endeavors.com, 2001.

[16] Sun Microsystems: Project JXTA http://www.jxta.org. Accessed July 2001.