Agents2Go: An Infrastructure for Location-Dependent Service Discovery in The Mobile Electronic Commerce Environment

Olga Ratsimor
Department of Computer Science and Electrical Engineering University of Maryland Baltimore County
1000 Hilltop Circle, Baltimore, MD 21250
+1 410-455-3971
oratsi2@cs.umbc.edu

Vladimir Korolev
Department of Computer Science and Electrical Engineering University of Maryland Baltimore County
1000 Hilltop Circle, Baltimore, MD 21250
+1 410-455-3971
vkorol1@cs.umbc.edu

Anupam Joshi
Department of Computer Science and Electrical Engineering University of Maryland Baltimore County
1000 Hilltop Circle, Baltimore, MD 21250
+1 410-455-2590
joshi@cs.umbc.edu

Timothy Finin
Department of Computer Science and Electrical Engineering University of Maryland Baltimore County
1000 Hilltop Circle, Baltimore, MD 21250
+1 410-455-3522
finin@cs.umbc.edu

ABSTRACT

In recent years, the growth of Electronic Commerce and Mobile Computing has created a new concept of Mobile Electronic Commerce. In this paper we describe the Agents2Go System that attempts to solve problems related to location dependence that arise in a Mobile Electronic Commerce environment. Agents2Go is a distributed system that provides mobile users with the ability to obtain location dependent services and information. Our system also automatically obtains a user's current geographical location in CDPD (Cellular Digital Packet Data) based systems without relying on external aids such as GPS (Global Positioning System).

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**],
H.3.4 [**Communications Applications**].

General Terms

Algorithms, Design, Experimentation.

Keywords

Mobile information access, context dependent, location dependent, service discovery, agents.

1. INTRODUCTION

With the proliferation of mobile computing, more and more people use a variety of mobile devices in their dally lives. Recent years have also seen a remarkable growth in Electronic Commerce. The merger of these concepts has resulted in the emergence of Mobile Electronic Commerce (M-Commerce).

One of the most critical requirements for M-Commerce is the ability to discover services in a given context. An important component of a user's context is their current location. For example, a user's on arriving at a location that he/she has never visited before should be able to find a local cab service. Current mobile devices have well known inherent limitations [1] like limited power supply, smaller user interface, limited computing power, limited bandwidth and storage space. These limitations necessitate the development of systems that provide mobile users with high quality, precise and context relevant information. It is important that these systems be highly scalable since the demand for service searches will increase in the future.

A location dependent search utilizes a user's current geographical location to refine their search and provide access to locally available services. One of the challenges of location-based searches is determining the user's current location. Users are often uncertain, or even completely unaware, of their current geographical location making location based searching more difficult. An automated detection of the user's current location would be very helpful in eliminating this problem.

Location dependent systems are naturally described and implemented as distributed systems. This also improves their fault tolerance and scalability. For instance service information can be grouped by location and managed by a server responsible for the specified geographical region. In such a decentralized scheme user requests are processed at the local server and do not burden the rest of the system. This makes the system more efficient, responsive and scalable.

In this paper we introduce the Agents2Go System. It is an agent based distributed system that allows the creation of location dependent service/information system. We present the use of Agents2Go to create a location dependent restaurant recommender system. We use this application to drive the description of our system.

2. RELATED WORK

There are a number of platforms that provide multi-agent infrastructures to allow inter-agent communication and collaboration. These platforms can be used to create collaborative intelligent agent environments that could provide location based information services. One such infrastructure is the Lightweight Extensible Agent Platform (LEAP)[3]. LEAP provides a lightweight platform that is executable on small devices such as PDAs and phones. It is FIPA[15] compliant and also supports WAP[4] and TCP/IP.

The YellowStone Project[6] from Reticular Systems, Inc. also deals with location dependent services. Essentially, there are communities of software agents called agencies, which provide information services and e-commerce support for a particular geographical area. However, a participating user has to specify his/her current geographical location.

Very recently researchers at AT&T Research Labs have described a project with similar goals[5], which also locates the user in a CDPD [2] environment using cell tower ids.

There are also a number of services on the Internet that provide location dependent information. Digitalcity.com[7] provides users with local information such as news, events, weather forecasts, restaurant information, local attractions etc. Several other sites like "Yahoo! Get Local"[8] from Yahoo! and citysearch.com[9] from MSN provide similar services. All these sites are mainly geared towards users that have wired desktop computers. In contrast, BeyondGuide[10], Inc. specifically focuses on providing location related information services (tourism information) for mobile users. A user of BeyondGuide[10] can use a mobile phone to gain access to information about some specified geographic location. The information is delivered to the user through his/her cell phone in voice format. MyAladin.com[11] also seeks to provide location-based services for mobile devices. Unfortunately, as of this writing, we were unable to find more detailed information about their services.

Our Agents2Go system has several distinguishing features that provide advantages over the existing location dependent service search systems. First, it is a platform to deploy any location dependent service, not just to provide location dependent information. Second, our system automatically discovers a user's location and uses this information to refine its searches to provide the most relevant information to the user. Also the Agents2Go system allows service providers to actively participate in the system through dynamic information updates. This improves the quality of information or services presented to the user and ensures that the service providers are able to send their latest promotions/updates to their users.

3. DESCRIPTION OF THE COMPONENTS OF THE AGENTS2GO SYSTEM

The Agents2Go System, illustrated in Figure.1, is composed of several components: the PalmApp, the Agents2Go Server, the Locator, the Agents2Go Information Repository, the restaurant Brokers and participating Restaurant Agents.

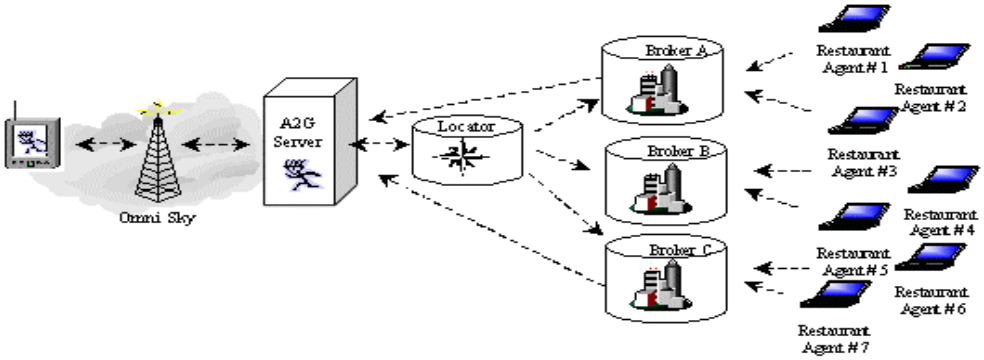


Figure 1 Full view of the Agents2Go System

3.1 PalmApp

The PalmApp is the end user interface to the Agents2Go System. This component runs on the user's PDA equipped with a CDPD modem. Essentially, it is a generic "form visualizer" that is independent of the system functionality. To reduce complexity, in-house markup tags are used to specify the layout and components of the form.

In our system, the PalmApp captures a user request, converts it to an appropriate format, and then forwards that request to the Agents2Go Server. The PalmApp also handles the responses from The Agents2Go Server and presents them to the user.

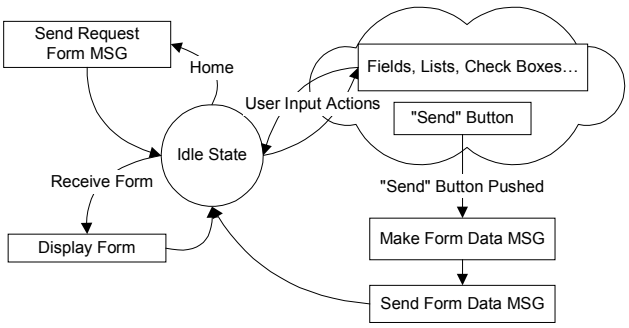


Figure 2 State Diagram of PalmApp Component

3.1.1 Communication and Location Detection

All the messages that are exchanged between the Agents2Go Server and the PalmApp are sent using the Centaurus Communication Protocol (CComm)[12]. CComm is a layered protocol that provides communication between mobile clients and a server. *Level2* is the layer that is closest to the application layer. It handles transmission of data messages and control messages between a server and a client. This layer handles multiple clients, is insensitive to disconnections and is easily portable. *Level1* is the layer that serves as glue between *Level2* and existing stacks like IrDA[13], BlueTooth[14], CDPD[2], or TCP/IP. It handles all interactions with these stacks including connection/disconnection issues, user identification and authentication.

Our current Agents2Go System uses CDPD *Level1* Module, which is an extension of the UDP *Level1* module. CDPD provides an infrastructure that allows the transmission of data over idle capacity of already existing cellular voice networks. Cellular networks consist of cell towers. Each cell tower has a unique id and defines a geographical boundary called *cell*, the area that is serviced by that tower. Our system employs these tower ids to identify a user location. We have developed a library that allows us to control and interact with Novatel wireless modems through MSCP (Minstrel Status and Configuration Protocol). The PDA's CDPD module employs this library to obtain periodic status reports. These reports contain information like cell tower signal strength (which can be used to minimize packet loss), the tower id (which we use for inferring coarse information about current location), etc. The PalmApp has access to this periodically updated report through a *Status Data Structure*.

3.1.2 Messages

All the messages that are exchanged between the PalmApp and the Agents2Go Server are encapsulated in a generic message format. The generic message format specifies a *Sender ID*, a *Message Type* and *Message Content*. Messages sent from a PDA to a Server use that PDA's ID as the *Sender ID*. This PDA ID represents a unique id for a user's PDA in a particular application. Messages sent from a Server to a PDA use the Server's ID as the *Sender ID*. A Server ID represents a unique id of a particular Agents2Go Server. The *Message Type* field can contain one of three message types: "*response form message*", "*form request message*", or "*form data message*". The *Message Content* field contains the message itself.

The PalmApp uses a generic "*form request message*" to request forms from the Agents2Go Server that are displayed to the user. The "*form request message*" specifies the form name that the PalmApp is requesting and the cell tower, with which it is currently communicating. This tower id is obtained from the *Status Data Structure*.

On startup, the PalmApp requests an "*initial query form*" from the Agents2Go Server. Through this form, the user can specify a desired request. Upon submission of the request (through a send button for example), the input of the form is converted into a "*form data message*" and sent to the Agents2Go Server. The "*form data message*" includes only the values of active components in the form like check boxes, fields, lists, etc. Once the PalmApp sends the "*form data message*" it waits for a response from the Agents2Go Server. The format of a response is a form that the PalmApp displays to the user. This form may contain a "*home*" button that causes the PalmApp to generate the "*initial query form*" message to allow the user to enter a new query.

Unlike a "*form request message*" or a "*form data message*", the "*response form message*" is initiated at the Agents2Go Server and destined for the PalmApp. This message contains a form that the PalmApp is required to display to the user. This message may contain a response to the user's query, an error message, etc.

3.2 Agents2Go Server

The Agents2Go Server is the component that handles messages to and from a PalmApp. User queries are forwarded to the Locator, and the corresponding responses are forwarded back to the PalmApp.

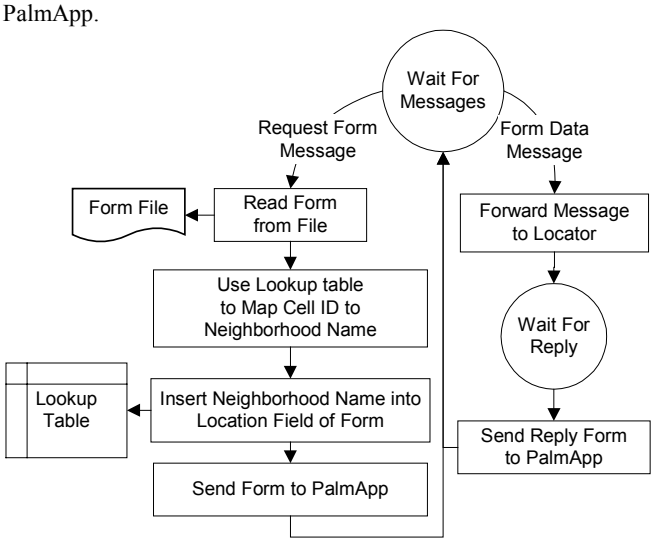


Figure 3 A State Diagram for The Agents2Go Server Component

Upon receiving a "*form request message*", the Agents2Go Server reads the requested form from a file. If the desired form cannot be located, a suitable error form is sent back to the PalmApp. Once the desired form is located, the Agents2Go Server uses a lookup table to map the specified cell tower id (obtained from the request message) to its neighborhood name. This neighborhood name is inserted into the location field of the form that needs to be displayed to the user. This neighborhood name, which can be changed by the user, is inserted into location field of the form displayed to the user. Thus a user, regardless of his/her current location, can find information about any participating region. This is encapsulated in a "*response form message*" and sent back to the PalmApp.

When the Agents2Go Server receives a "*form data message*" from a PalmApp, it forwards it to the Locator. Other alternative designs could be used, a "*form data message*" could be forwarded to the corresponding Broker. Once the Agents2Go Server receives a response from either the Locator or a Broker, it generates the

corresponding "*response form message*" and sends it to back the PalmApp.

3.3 Locator

The Locator is the component that receives requests from the Agents2Go Server, determines which Broker is responsible for the area from which the request originated and then forwards the request to that Broker. The Locator maintains a table that maps geographical areas to Brokers. This table is dynamically built and maintained. The Locator listens on a well defined port for registration messages from Brokers. This registration consists of a port on which the Broker will accept requests forwarded by the Locator, and the geographical area for which this Broker is responsible. Upon receiving a request form the Agents2Go Server, the Locator looks inside the request string and extracts the point of origin information. This information is used to determine the designated Broker. The Locator then forwards the request to that Broker. If the Locator is unable to locate a suitable Broker for the given request, the Locator sends a "*broker not found*" message back to the Agents2Go Server. A reliable communication channel is maintained between the Agents2Go Server and the Locator for all message transfers.

3.4 Broker

The Broker is the component of the system that maintains information about restaurants in its designated geographical region. The Broker processes requests from a user and generates suitable responses. These requests are forwarded to the Broker from the Locator and the generated responses are sent to the Agents2Go Server for forwarding to the requesting PalmApp.

The Agents2Go System partitions participating restaurants into sets based on the geographical region in which these restaurants are located. These sets are called *coverage regions*. Each *coverage region* is assigned a unique name and is serviced by a designated Broker. This Broker is responsible for generation of replies for requests pertaining to its *coverage region*. Our current Broker implementation allows grouping of several geographical regions or partitioning a single geographical region to construct a *coverage region.*

Every Broker in the Agents2Go System is also associated with a specific Agents2Go Information Repository. An Agents2Go Information Repository is a set of databases that contain information about participating restaurants in a Broker's *coverage region*. Restaurant information like name, address, cuisine etc. of all participating restaurants in that *coverage region* is distributed among these databases. This information can be classified as *static*, since it rarely changes. The Broker is also responsible for frequently changing restaurant information like waiting times and promotions. This kind of information can be classified as *dynamic.* Figure 4 Restaurant A, Restaurant B and PDA i are able to participate in the Agents2Go System. However, PDA ii is not.

This *dynamic* information is maintained within the Broker itself. This separation of *dynamic* and *static* information reduces the number of messages that is exchanged between the Agents2Go System components.

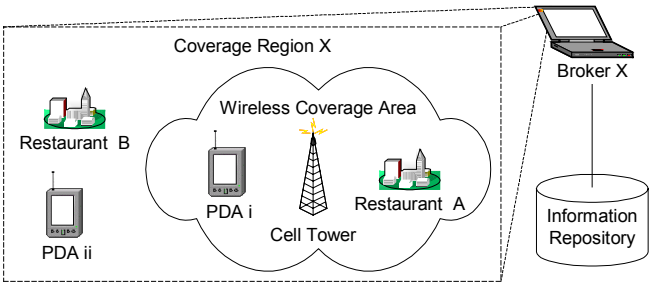
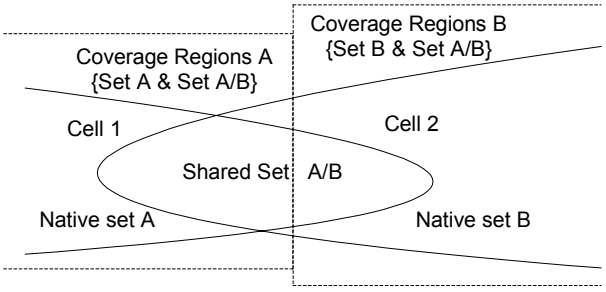


Figure 5 Restaurant A, Restaurant B and PDA i are able to participate in the Agents2Go System. However, PDA ii is not.

It is common for wireless cells to overlap. If a user is in a cell overlap region, then that user's PDA connection can hop from one overlapping cell to another. So, the cell tower id that the user's PDA picks up can change quite frequently. If the cell overlap is contained within a single *coverage region*, then cell hopping is not an issue because any cell id that is picked up in that cell overlap will map to the same *coverage region* name and is managed by the same Broker. However, if the cell overlap is on the border of two or more *coverage regions*, the user's PDA may pick up ids that belong to cell towers that service neighboring *coverage regions*. This could create a scenario where a user's request query could be routed to a neighboring Broker that has absolutely no information about the current location of that user.

The Agents2Go System solves this issue by imposing a policy that prohibits *coverage regions* from overlapping unless there are some cell overlaps falling on their borders. Further, the Agents2Go System requires a special configuration for the Information Repositories that are associated with these overlapping *coverage regions*. The Agents2Go System partitions restaurants of overlapping regions into two types of disjoint sets: *shared* and *native* type. A *Shared* set contains restaurants that are located in the areas of cell overlap (that fall on the borders of *coverage regions*). A *Native* set contains the remaining restaurants that are in the Broker's *coverage region* but not in the cell overlap. Each set is stored in a separate database.



It is also possible that a user, after requesting info, moves to a different coverage region, while the request is being processed. In this scenario, the reply to the request contains the information relevant to the region from where the request originated. This information could still be of relevance to the user since he/she is not far form the initial coverage region.

On initialization, a Broker establishes connection with its Agents2Go Information Repository. The Broker queries its repository to obtain ids of restaurants for which it will broker information. If the Broker is unable to establish required connection(s) with its repository, the initialization fails and the Broker exits gracefully. Upon successful connection establishment, the Broker builds a "*waiting time*" table. The "*waiting time*" table is a data structure that the Broker uses to maintain the dynamically changing restaurant information. The restaurant ids and the location identifiers for the restaurants are used as keys of the table. The values of the table are the waiting time information, promotion information and time stamps of updates. Once the table is built the Broker registers itself with the Locator component. The registration contains geographical regions that this Broker administers and the port on which the Broker will listen for forwarded requests from the Locator. If the registration with the Locator fails, the Broker exits gracefully.

Once initialized, the Broker starts to listen for updates sent by the local Restaurant Agents and requests forwarded by the Locator. When a Broker receives a wait time update and some promotion information from a Restaurant Agent, it timestamps it and then caches this information into the "*waiting time*" table. When the Broker receives a request from the Locator, it first checks for the validity of the request. If the request string does not match the expected format, further processing of that request is terminated and an error message is sent back to the user. For valid requests, corresponding database queries are dynamically generated. Request parameters are dynamically incorporated into a database query.

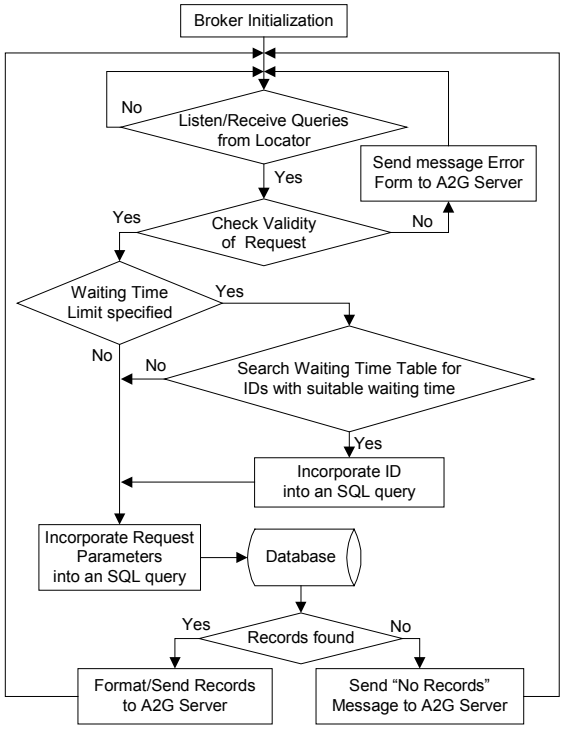
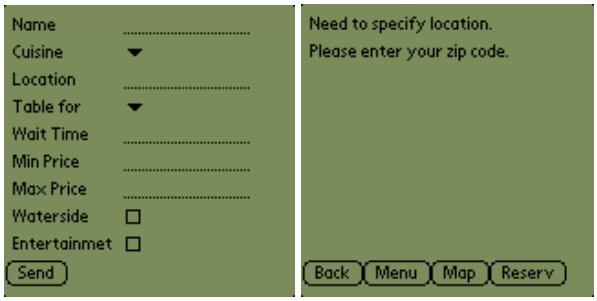


Figure 6 State Diagram for Broker Sub Part

If the request contains a waiting time limit, then the Broker searches its "*waiting time*" table for the restaurants that have their waiting time below the requested time limit. This search returns ids of the restaurants that have suitable waiting time. Returned ids are also incorporated into the database search query. Once the query is constructed, it is executed on the Broker's Agents2Go Information Repository. If no records are found, then a "*No record found*" message is returned to the user.



If matching records are found, a timestamp for each result record is evaluated. This timestamp can belong to one of three age groups: "*fresh*" age group, "*aged*" age group, or "*trashed*" age group. A record will be treated differently depending on its age group, and on whether the user is interested in dynamic information.

To identify the age group of a timestamp, the difference between the value of that timestamp and current system time is calculated. This difference is the age of the timestamp. The timestamp age is compared against two threshold values. The first, lower, threshold denotes the limit between the "*fresh* "age group and the "*aged*" age group. The second, higher, threshold denotes the limit between the "*aged*" age group and the "*trashed*" age group.

Hence, if a timestamp is "*fresh*", the record that has been selected is sent to the user, and the dynamic information is displayed in its regular format. Else, if a timestamp is "*aged*", the record is sent to the user along with a warning that the record is not up to date. And finally, if a timestamp is "*trashed*", the user request is analyzed to determine if the user is interested in dynamic information. If the user's request specifies a waiting time limit, the record is dropped from the result set. On the other hand, if there is no time limit specified, the record is sent to the user, but the dynamic, outdated portion of that record is replaced with an "*Information is unavailable*" message. This classification of the record's timestamps gives users some flexibility. Users themselves can determine if the dynamic information is useful.

Once the response to the query is formed, it is converted into an appropriate format and sent to the Agents2Go server. So there are two types of Restaurant Information messages that could be sent to the Agents2Go Server.

3.5 The Restaurant Agent

The Restaurant Agent is the component of the system that resides and runs at the location of the participating restaurant. This component allows a restaurant host to update dynamic information such as waiting times, promotion information, etc.

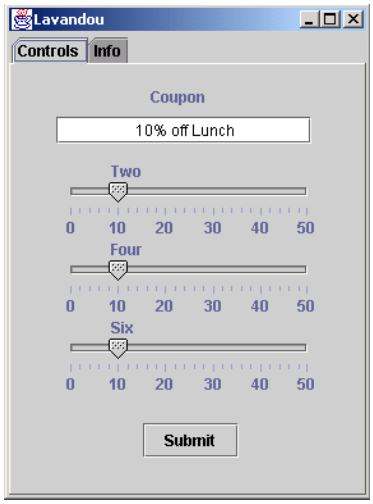


Figure 7 Graphical interface for the Restaurant Agent.

Once the updates are made and the "*Submit*" button pushed, the update message is sent to the Broker that is responsible for the geographical area in which the restaurant resides. If the restaurant is located in a cell overlap region, which is managed by several Brokers, then the update message is sent to every Broker that manages the overlap region. The update message contains the restaurant id, and other relevant information like the value of the wait time for table for two, the wait time for table for four, the wait time for table for six, etc. Each update message also contains a timestamp specifying the creation time. The Broker, upon receiving an update message, extracts the relevant values from the message and inserts these values into the appropriate row of its "*waiting time*" table.

4. EXPERIMENTAL SETUP

Let's consider an example illustrating the Agents2Go System at work. Consider a user ***A'*** currently located in the University of Maryland Baltimore County (UMBC) campus. User ***A'*** would like to use the Agents2Go system to find information about a local French restaurant with a waiting time less than 20 minutes for a party of four. The UMBC *coverage region* is comprised of several geographical regions; Arbutus, Catonsville and Ellicott City. There are 10 restaurants in this *coverage region*. There is also a set of cell towers that provider wireless connectivity for this region. The Agents2Go Server is configured to map ids of these cell towers viz. 25, 30, 18, etc., to the UMBC *coverage region*. The Locator is configured to route requests originating in the UMBC *coverage region* to the UMBC Broker responsible for this *coverage region*. The UMBC Broker is configured to manage *dynamic* information for the 10 restaurants in its *coverage region*. The UMBC Information Repository associated with the UMBC Broker contains *static* restaurant information for these 10 restaurants. The Restaurant Agents for these restaurants should be configured to send updates to this UMBC Broker. Currently, we employ a simulator to simulate updates from these Restaurant agents. The Update Simulator generates a set of update messages containing random waiting times and promotions for these restaurants.

The user ***A'*** starts up the PalmApp on his/her Palm V connected to an Omni Sky modem. The user is presented with an initial splash screen from which the user can navigate to the *request form*. The location field of the request form is automatically filled-in with the name of the *coverage region* in which user ***A'*** is currently located (UMBC). User ***A'*** enters a request for an French restaurant with waiting time limit less than 20 minuets for a party of four. The user ***A'*** submits the request to the Agents2Go system. The Agents2Go Server forwards this request to the Locator, which in turn forwards the request to the Broker in-charge of the UMBC *coverage region* (UMBC Broker). The UMBC Broker generates a response to this request and through the Agents2Go server, delivers it to the user. In this case, the Agents2Go System finds a restaurant named "Tersiguel's" that matches all of the criteria that user ***A'*** specified. The Agents2Go System sends user ***A'*** the *dynamic* and *static* information about this restaurant. Once the user is done viewing this information, he/she can use the *Home* button to navigate back to the *request form* screen from which he/she can specify a new request.

We experimented with the Agents2Go System in several geographical regions in the state of Maryland and the state of Texas. The sizes of coverage regions that we formed varied from five to fifteen miles in diameter. From our experiments, we have

determined that cell overlaps range from any where between a quarter of a mile to a single street block. This allowed us to draw a conclusion that cell overlaps are not a big design issue. On the average it took us anywhere from a few seconds to one minute to get a reply to a request, with the time variance mostly depending on the quality of the reception in a particular location. This means that in the absence of pathological conditions (user at a cell boundary, user moving extremely fast), our response will return before a user has changed location significantly since issuing the query.

5. FUTURE WORK AND CONCLUSION

We have implemented a working prototype of the Agents2Go System. It is a location aware, distributed system that allows mobile users to request and receive various services information that is of most relevance to their current geographical location. Thus, the mobile users will not be burdened with extraneous information for services in remote locations. Also, the Agents2Go System allows service providers to supply dynamic service updates. This dramatically improves the value of the service for the providers and gives users more refined service information. Our implementation currently deals with restaurants, but it could be easily updated to work with other location specific services. All of the above mentioned features make our system well suited for various M-Commerce environments.

We are currently working on improving the Agents2Go System. We are expanding our knowledge base of restaurants located in the greater Baltimore and Washington region. New cell ids and coverage regions are being added to the Agents2Go System. We are also working on developing a more interactive environment between users and restaurants. In particular, we are starting to work on implementing auctions between users and services. We are looking at scenarios where a user is holding an auction for promotions from restaurants that he/she is interested in visiting. We are also working on expanding a Broker's functionality to include the capability for a Broker, to forward a request that yields no matches to neighboring Brokers that might be able to provide the user with a suitable response. Another extension to our work could be incorporation of disconnected operations. If we could anticipate the region in which the use will be next we could preload service information for that region and speed up the interaction process. Also if user is located in a region with poor wireless coverage, the information about this region could be uploaded onto the users PDA whenever there is a good reception. We are also considering logging service requests. The logs could be used for statistical analysis, which could be used to recommend improvements for service providers. We are also looking into making the Agents2Go System more distributed and fault tolerant. There is also a plan to improve the interfaces. Since our underlying communication platform [12] also works with Bluetooth, we are starting to experiment with discovery based on enhancing Bluetooth SDP.

6. ACKNOWLEDGMENTS

This work was supported in part by NFS awards IIS 9875433 and CCR 0070802.

7. REFERENCES

[1] A. Joshi, S. Weerawarana, and E.N Houstis, "On Disconnected Browsing of Distributed Information", in Proc. IEEE Research Issues in Data Engineering (RIDE '97), pp 101-107, 1997.

[2] Mark Taylor, William Waung, Mohsen Banan, "Internetwork Mobility: The CDPD Approach", Prentice Hall Professional Technical Reference, September 1996.

[3] The Lightweight Extensible Agent Platform http://leap.crm-paris.com

[4] The Wireless Application Protocol (WAP) http://www.wapforum.org/

[5] S. Muthukrishnan, Rittwik Jana, Theodore Johnson, Andrea Vitaletti, "Location Based Services in a Wireless WAN using Cellular Digital Packet Data (CDPD)", to appear in Proc. 2nd ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE01).

[6] The YellowStone Project http://www.agentbuilder.com/Documentation/Yellowstone/

[7] digitalcity.com http://www.digitalcity.com

[8] Yahoo! Get Local http://local.yahoo.com

[9] citysearch.com http://www.citysearch.com

[10] BeyondGuide http://www.beyondguide.com

[11] myAladdin.com http://www.myaladdin.com/www/index.jsp

[12] L. Kagal, V. Korolev, H. Chen, A. Joshi, T. Finin. "Centaurus: A Framework for Intelligent Services in a Mobile Environment", in Proc. IEEE The 21st International Conference on Distributed Computing Systems (ICDCS-21), pp 195-201, 2001.

[13] Infrared Data Association http://www.irda.org

[14] The Official Bluetooth Website http://www.bluetooth.com

[15] Foundation for Intelligent Physical Agents http://www.fipa.org/