Technical Report TR-CS-01-18

Query Routing and Processing in Mobile Ad-hoc Environments

Filip Perich, Sasikanth Avancha, Anupam Joshi,
Yelena Yesha, Karuna Joshi

Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County
1000 Hilltop Circle
Baltimore, MD 21250

5 November 2001

Abstract

In existing mobile information access systems, mobile devices are typically viewed as consumers of information, with information providers resident on the wired network. With the advent of short-range ad-hoc connectivity based on Bluetooth like systems, an alternative scenario arises where mobile devices gather and exchange information from not just wired sources, but also their environment and one another. Each device is both a source and a consumer of information/data. In this paper we describe new challenges that this scenario presents to the distributed database framework, and present the design of a framework for serendipitous querying and query response in an ad-hoc mobile environment. We also describe an implementation of this system on mobile devices connected over Bluetooth networks, and present experimental results.

I. INTRODUCTION

The constant enhancements in capabilities of palmtop, embedded and wearable devices, together with the advent of pervasive connectivity and nanosensors represent a new paradigm for the way we, as software developers and builders, think about interaction among devices. Some of the pervasive connectivity is due to the infrastructure based on 2.5/3G cellular networks. In such environments, mobile devices are typically viewed as consumers of information, with information providers resident on the wired network. In such systems the traditional client–server interaction is an appropriate model perhaps with the "client" database being extremely lightweight [4] or having a (partial) replicate of the main database on the wired side [34]. An alternative approach will be necessary with the spread of short–range narrow-band systems such as Bluetooth [33], which allow for devices in a "vicinity" to spontaneously network with one another. Mobile devices will become more autonomous, dynamic and adaptive with respect to their environment. They will become both sources and consumers of information, and cooperate with other devices in their vicinity in order to pursue their individual and collective tasks. For instance, a car can ask those traveling in the opposite direction if they know of a gas station within ten miles. Of course, this could be done by the car relaying its GPS coordinates to some centralized server over an infrastructure-supported wireless network. However, this approach may be expensive, both in terms of monetary cost and the accessibility of the infrastructure. In such an *ad-hoc* environment, the neighborhood of any entity can be extremely volatile. For instance, when a person walking in a mall queries other shoppers for the best deals in jewelry, she may obtain different results based on the time she places the query. As every entity is potentially changing its location with respect to others, it may not be possible for it to pre-determine the set of information sources (catalogs) in its vicinity at any given point of time. For querying in such a scenario, we need to create a robust infrastructure in which independent devices existing in a particular location can and will discover, inter-operate, and cooperate with other devices in their vicinity in an as-needed and as-desired basis.

Current research in mobile computing has generally been limited to allowing applications built for the wired world (e.g., WWW, databases etc.) to run in wireless domains using proxy based approaches ([3], [12], [25], [24]). However, the enabling of peer–peer interaction between entities in *ad-hoc* environments poses additional challenges that must also be addressed.

To further motivate the need for this framework in an *ad-hoc* environment, let us consider the following scenario. It is 5:40 in the afternoon, and Bob's working day at his new job is just ending. As he is getting ready to leave the office, his phone rings. It is his new friend Jane asking him to meet her at the local shopping mall. Bob agrees to meet her, and notifies his palmtop about the decision. In addition, Bob asks the palmtop to find directions to the mall, as he has never been there before. While he is walking through the building toward the parking lot and ultimately toward his car, the palmtop is able to connect to the office network infrastructure and fetch the appropriate directions through a *DReggie*-like service/information broker [8]. Once in the car, Bob reads and follows the instructions. However, he feels that the traffic is not moving fast enough; he would like to get to his destination quicker. He instructs his palmtop, which can now connect to the cars around him or passing him, to ask them whether they know about a faster route to

the mall. The palmtop contacts its neighbors and returns with an alternative map, which is longer but which avoids the afternoon traffic jam that is building up on the current route. Bob therefore takes the different roads and arrives at the mall's entrance twenty minutes before the expected time. He decides to use the extra time by checking out the local stores to see if he can get a good deal on some small gift for Jane. It is forty minutes later and Jane finally arrives. After exchanging greetings, they decide to walk to a quiet place for dinner. Bob asks his palmtop to suggest available restaurants and lets Jane pick one. She chooses the closest Italian restaurant, which indicates it has an available table with no waiting period. Thus, they get seated immediately and spend several hours while eating and chatting. In the meantime, Bob's palmtop learns that it will stay in the given location for a while and it starts to autonomously interact with other devices in its vicinity. For instance, it could share traffic condition related information it might have cached while Bob was driving over too the mall. It could also obtain and cache business cards of other people in the mall by matching Bob's profile with theirs. After the meeting with Jane, Bob walks down to the garage and drives back to his apartment.

In this example, we can see that Bob's palmtop is heavily utilizing the locally available resources as well as changing its interaction mode based on the particular context to satisfy any implicit or explicit query that Bob may pose. The palmtop utilizes the knowledge of Bob's interests and preferences (i.e., facts and rules) that were either explicitly provided by Bob or that it has learned during its life span. It uses this knowledge to determine the appropriate actions it may have to execute to satisfy a given request. For example, the palmtop is capable of deducing that when its current location is Bob's office, it is more efficient to contact the local *DReggie*-like broker. On the other hand, when it's current location is Bob's car and moreover the car is moving on a highway, it deduces that a presence of any "local" information provider encompasses only a very short duration, and thus does not waste its resources by caching unnecessary information. Finally, while Bob is walking through the mall, the palmtop is able to cache the local advertisements and combines them with Bob's preferences in anticipation of his future movements.

In the following sections, we will illustrate the challenges posed by such *ad-hoc* environments, our proposed solution and results from experiments. In section II we discuss other existing work in the area of distributed data management in mobile networks. Section III presents the challenges and problems of distributed data management in *ad-hoc* networks. We present details of the framework that addresses these problems in section IV. In section V, we describe systems level details of the implementation of this framework. We present experimental results in section VI and conclude the paper in section VII.

II. Background and Related Work

The problem of management of data in a distributed environment has been well researched, both in terms of fixed infrastructure (e.g., the WWW) and infrastructure-based wireless networks (e.g., Mobile IP and PCS). The work on distributed and federated databases is well-known in the community, so in this section we present previous work related to data management in wireless networks.

Within a mobile database environment, cached data on mobile clients can take the form of materialized views. In order to efficiently maintain such materialized views while taking into consideration disconnected operations, Lauzac and Chrysanthis [28] present a mechanism within the fixed network they call "view holder", that maintains versions of views required by a particular mobile host. They also propose an extension to SQL that enables the programming of the view holders by the mobile clients based on their preferences and capabilities and discuss it's implementation. Kottkamp and Zukunft [26] have presented optimization techniques of query processing in mobile database systems of queries that include location information. They present a cost model for query optimization incorporating mobility specific factors like energy and connectivity. They have also examined different localization strategies for mobile users. Using a newly developed simulation model, they show that no single localization strategy performs acceptably under all conditions and identify the critical factors for adapting a query processing subsystem to the employed

location management strategy. Bukhres *et al* [6] propose an enhancement to the infrastructure-based mobile network model of Mobile Hosts (MHs) connected over a wireless virtual subnet and Mobile Support Stations (MSSs) connected to a wired static network. They recommend the addition of a mailbox, which serves as a central repository for the MHs. This mailbox will be maintained by the cellular provider and will be duplicated in all the MSSs so that the MH can access it both locally and remotely. Pitoura [31] presents a replication schema based on augmenting the mobile database interface with operations with weaker consistency guarantees. A implementation of the schema is presented by distinguishing copies into quasi and core; protocols for enforcing the schema are introduced. The paper also evaluates the performance of the weak consistency schema for various networking conditions. Demers *et al* [13] present the system architecture of the Bayou System which is a platform of replicated, highly available, variable-consistency, mobile databases on which to build collaborative applications. The emphasis is on supporting application -specific conflict detection and resolution and on providing application-controlled inconsistency.

We note that in most of the papers discussed above, the wireless networks are supported by the fixed, wireline infrastructure. Most of the query optimization techniques proposed in this body of work require the support of wireline networks. Both [26] and [6] use the concept of a MH being in one of two modes, local mode (within the home network) or nomadic mode (outside the home network).

Our work, on the other hand, assumes no support for MHs from the fixed infrastructure. In fact, our focus is on networks formed spontaneously by MHs as and when required. These networks may be formed either with other MHs or with fixed nodes via a MSS. Querying and processing data in such *ad-hoc* networks is not only useful but also required. When the MH requires instantaneous information (e.g. traffic updates or bad weather warnings), it may be more easily accessible from other "local" MHs than a fixed node. In our work, a mobile device is always is nomadic mode, as defined by [26] and [6]. This implies that the necessity of maintaining duplicate location information no longer exists.

We now present other relevant work in data access in mobile systems not directly related to our work. Mazumdar and Chrysanthis [30] present a system called PRO-MOTION that helps achieve consistency in mobile databases through the use of localization. Guy *et al* [19] discuss an optimistically replicated file system designed for use in mobile computers. The file system, called Rumor, uses a peer model that allows opportunistic update propagation among any sites replicating files. This work describes the design and implementation of the Rumor file system, and feasibility of using peer optimistic replication to support mobile computing. Holliday *et al* [22] present the notion of a distributed database made up entirely of mobile components. To handle frequent disconnections, authors have developed a disconnection and reconnection procedure to allow normal processing on the connected components. This paper discusses a protocol based on epidemic communication to support such a system while ensuring one-copy serializability. Holliday *et al* [21] have also investigated an epidemic update protocol that guarantees consistency and serializability in spite of a write-anywhere capability and conduct simulation experiments to evaluate this protocol. They present experimental results supporting this approach as an alternative to eager update protocols for a distributed database environment where serializability is needed. Loke and Zaslavsky [29] have describe two ideas for mobile agent based distributed workflow enactment: an algebra of agent itineraries and its correspondence to workflow specifications, and a mobile agent control center for managing agents enacting workflows. Acharya *et al* [1] present a broadcast-based mechanism for disseminating information in a wireless environment. To improve performance for non-uniformly accessed data, and to efficiently utilize the available bandwidth, the central idea is that servers are repeatedly broadcasting data to multiple clients at various frequencies. The authors superimpose multiple disks of different sizes and speeds to create an arbitrarily fine-grained memory hierarchy, and study client cache management policies to maximize performance. Infostations [16] is a system concept proposed to support "many time, many where" wireless data services including voice mail. It allows mobile terminals to communicate to Infostations with variable data transmission rate to obtain the optimized throughput. The main idea is to use efficient caching techniques

to hoard as much data as possible when connected to services within an island of high bandwidth coverage, and use the cached information when unable to contact the services directly.

III. Challenges of Data Management in Ad-hoc Environments

If the entities in the *ad-hoc* environment are treated as information repositories, we can describe this model as a type of mobile distributed databases, albeit a far more complex one than the conventional client-proxy-server model. We can illustrate this by classifying our environment in terms of four orthogonal axes, i.e., autonomy, distribution, heterogeneity, and mobility ([32], [14]). The system is highly autonomous since there is no centralized control on the individual databases that clients maintain. It is also heterogeneous; we only assume that entities can "speak" to each other in some neutral format. The system is clearly distributed – parts of data reside on different computers, and there is some replication as entities cache data/metadata. However, this is not just a situation with replicated data; a large degree of mobility is *ipso facto* inherent to the system. The important difference between our scenario and the ones in traditional mobile distributed databases therefore lies in the fact that our model further relaxes the requirement that some nodes must be fixed in a network. Instead, in the *ad-hoc* environment every entity can change its location and no fixed set of entities is "always" accessible. Note that this is distinct from disconnection management that infrastructure based systems deal with. In those systems, disconnections of mobile devices from the network are viewed as temporary events and when reconnected, any ongoing transactions between the mobile and the server will simply continue from where they left off before the disconnection or be rolled back. Thus, in addition to various issues of traditional distributed databases, the *ad-hoc* environment imposes the following new challenges.

A. Data sources available vary with location and time.

As entities move, their neighborhood changes dynamically. Hence, depending on the specific location and time a particular query is given, the originator may obtain different answers or none at all. For example, when Bob's palmtop in the car asks its neighborhood for an alternative route, it obtains most likely a different answer from a passing school bus than from a traffic light. Moreover, the originating entity cannot depend on a global catalog that would be able to route its query to the proper location. The only information always guaranteed to an entity is that residing on it. There is no guarantee that the device will be able to access information that resides on neighboring devices under high mobility conditions. This is due to the fact that current wireless networking technologies cannot support stable connections under such conditions. However, each entity can at least describe and advertise its capabilities (i.e., data sources and particular data instances it knows about) to its neighbors [9], [10], [18]. Therefore, to allow each entity to continue to interact with others in a dynamically changing environment, each device should have the option not only to store the necessary metadata information about its own capabilities, but also cache the metadata (and perhaps data) obtained from neighbors in its current vicinity. Moreover, some entities may decide to open their local catalog to the public to allow other entities in their neighborhood to utilize the cataloged information.

B. The query may be explicit or implicit.

In our framework, all entities should be capable of posing both explicit and implicit queries whenever desired. However, for pervasive systems to succeed in general, much of the interaction between the devices need to happen in the background. This implies that some interaction occurs without an explicit human intervention [7]. The *ad-hoc* environment, therefore, requires that some entities are able to accept queries from humans and propagate them in the *ad-hoc* network. Other entities in the environment are required to have access to individual rule-based profiles, which determine the future actions of these entities. This, for example, allows a user to ask her handheld device for the closest Indian restaurant and get answers which the device had obtained and cached, when it passed by other devices, because the user's profile indicated that

she prefers Indian food. In addition, this allows the handheld device to inform its user about the presence of a police officer it has learned of, anytime the user is speeding on a highway.

C. Since information sources are not cataloged a priori, schema translations cannot be done beforehand.

Some entities in the *ad-hoc* environment may have a limited capability, which prohibit them from executing a resource-expensive schema translation. Moreover, some entities may not even desire to perform reasoning or otherwise interact with information they do not understand in the first place. They may be satisfied with the results of interaction with only those information providers whose schema they completely understand. In some other cases, an entity may store the information that it may not understand at a given point in time in the hope that a translator becomes available in the future, and the information then becomes useful. Finally, it may also be the case that an entity attempts to immediately discover a translator and execute the translation as soon as possible in order to complete the ongoing task. In our implementation we consider three cases. First, an entity utilizes only such information and interacts with only such information providers that it completely understands. Second, the entity stores additional information that it may not understand at a given point in time in the hope that a translator will become available in the future, and the information will then become useful. Lastly, the entity tries to find a translator, and executes the process of translation as soon as possible, to accomplish the current task, during which the unrecognized information was obtained.

D. Cooperation amongst information sources cannot be guaranteed.

Clearly the issues of privacy and trust will be very important for an *ad-hoc* environment, where random entities interact in random transactions. In particular, there are three main issues that must be considered. First, there may be an entity that has reliable information but refuses to make it available to others. Second, there may exist an entity in the *ad-hoc* environment that is willing to share information; however that information may be unreliable. Lastly, when an entity makes information available to another entity, questions regarding protection of future changes and sharing of that information arise. To answer the need for privacy and security of data and information providers, one may suggest that a solution to this problem is the introduction of a Public Key Infrastructure (PKI), wherein each entity is assigned an x509.3 certificate and trust relationships [7] are established. This could allow the entity to sign all of its messages. If the receiving entity has the public key associated with the sender, then it will also allow the receiver to verify the validity of the message. However, in an *ad-hoc* environment, the Certificate Authority may be unreachable to verify that a certificate is correct. This brings into question the validity of the certificate itself. Of course, to solve this problem we may choose to impose a limitation on the framework that only one Certificate Authority is required for the entire environment. Even so, the usage of the Certificate Revocation List makes the current PKI insufficient for *ad-hoc* environments. Hence, any variations of PKI, such as XMLSig [15], will also be inapplicable for the similar reasons. Thus, the *ad-hoc* environment requires a more flexible security infrastructure that does not necessarily rely on a third party to verify both authentication and authorization of each entity in the environment. For our initial framework design we have, thus, not addressed the issue.

IV. Design of the Proposed Framework

Our framework is designed to handle serendipitous querying and data management efficiently and scalably in mobile *ad-hoc* environments. The framework consists of multiple instances of two main components, the Information Manager that we call *InforMa* and the information provider. Figure 1 shows an overview of the framework and the interaction between various entities in the framework. We describe *InforMa* in greater detail in section IV-A and the information provider ontology in section IV-B.

We believe that a framework such as ours should be able to satisfy any query, regardless of its complexity. However, we also argue that in a highly *ad-hoc* environment, irrespective of device capabilities (e.g.,

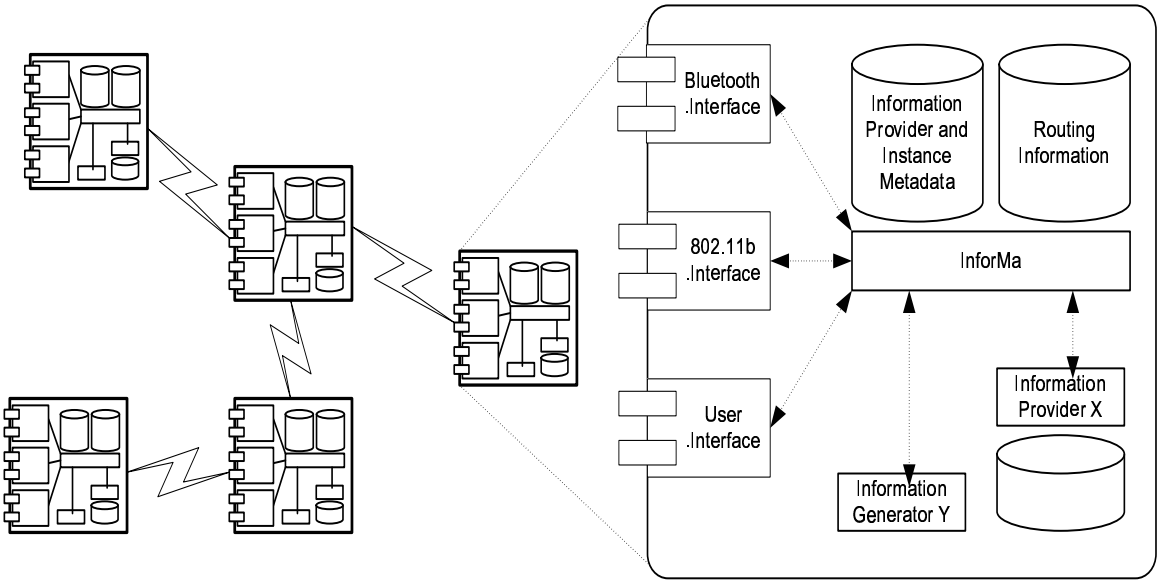

Fig. 1. Entity Details and Interaction in the Proposed Framework

memory size, computing power, battery life), it may be inefficient for devices to pose complex queries since the environment changes rapidly. Instead, the devices in our framework are expected to pose *simple* queries only. By *simple* queries, we mean that these queries can be answered either by searching through a set of facts or by making simple decisions. For example, a query presented to a logical database could be answered by evaluating ground truths and their derivatives only. Similarly, in a RDBMS, queries could be answered by performing simple operations on tables containing the information as facts. On the other hand *complex* queries are those that can only be answered by using, for instance, a reasoning engine based on a logic programming language such as Prolog or a RDBMS that performs complex join operations on multiple tables in order to answer the query. Answers to *complex* queries can only be obtained by analyzing relationships, like inheritance and transitivity, between many facts.

A. InforMa Design

Every entity in our framework is autonomous, i.e., its actions can be independent of those of its human user/owner, as far as interacting with other devices is concerned. Every entity is able to initiate an action solely based on the current contextual information and some rules. These rules, we assume, were either specified a priori or were learned during the entity's *execution life*. Finally, since all entities in *ad-hoc* environments are treated equally from the networking perspective, we carry this notion of equality to the level of entity functionality. Therefore, in the framework each entity possesses the following characteristics:

- Every entity manages a subset of the world knowledge repository that it can provide to itself and possibly to others. Naturally, this subset may be inconsistent with the knowledge of other entities and may even be empty.
- Every entity implements *InforMa*, which is a local metadata repository that includes schema definitions for locally available information providers and particular facts such as queries and answers for local and non-local information providers. Therefore, *InforMa* stores advertised schema for local information providers and also for those that it believes the entity can reach by communicating with other entities in its vicinity. In addition, *InforMa* stores facts that were produced locally or that were obtained from others. For example, when the entity has a local weather information provider and it furthermore knows that it is raining, *InforMa* includes metadata to reflect that knowledge. The schema for information providers is represented in the

DARPA Agent Markup Language (DAML) [20]. In addition, instances of information such as queries and answers, is also described in DAML. The DAML project is an effort spearheaded by DARPA and the W3C focused on standardizing DAML as the language to use to describe information available on any data source, in order that the information may be understood and used by any class of computers, without human intervention. DAML is being developed as an extension to the Resource Description Framework (RDF) [27] and the Extensible Markup Language (XML) [5]. XML is a structured language that allows information to be more accurately described using tags, thus removing ambiguities in the information. However, XML has a limited capability to describe the relationships (schemas or ontologies) with respect to objects. The use of ontologies provides a very powerful way to describe objects and their relationships to other objects. RDF is a data model that is capable of describing relationships between objects. Every RDF object is either a class or a property of a class. This objected-oriented model together with the descriptions of relationships allows for logical inferring of other relationships that may not be explicitly described. DAML is designed to improve the descriptive capabilities of RDF. For example, DAML introduces the concept of *Restriction* on a property. This allows specification of constraints on cardinalities of the values of an object. DAML also uses strong data typing, described by XML-Schema [17], to allow object values to be restricted to a specific data type.

Based on the model in which *InforMa* interacts with other entities in its vicinity, we can differentiate among the following four categories of *InforMa* instances:

1. In the most simple form, *InforMa* maintains required information only about information providers present locally on the device. Every entity in the *ad-hoc* environment is required to implement this version of *InforMa*. We believe that this version would be most suitable for resource-limited devices. In addition, this particular form of *InforMa* is most suitable for entities whose environment changes rapidly. This is the case of Bob's palmtop when Bob is driving and asks it for information about a quicker route to the mall. There is no need for the palmtop to remember information disseminated by other entities in its vicinity, as most of it becomes invalid within a short period of time. Instead, the palmtop tries to contact its current neighbors only whenever necessary.

In this case, *InforMa* stores the advertisements for its local information providers only. Thus, any time a query is posed, *InforMa* is contacted to provide an answer. *InforMa* first attempts to determine whether any local information provider is capable of answering the query and contact it. Otherwise, *InforMa* tries to locate some other public *InforMa* and requests its assistance. Finally, when all previous attempts fail, *InforMa* attempts to contact all of its neighbors to ask them for their help. However, once the query is satisfied, *InforMa* may choose to forget any information obtained from the other entities, in order to save memory.

2. As an extension to the first version, *InforMa* may decide to temporarily store the foreign information in the hope that a future query may be answered by reusing it. In this category, knowledge available to *InforMa* still remains restricted to the entity. For example, when Bob asks his palmtop for local restaurant information, the palmtop can assume that a related query may be posed in near future and thus decide to store the list of available restaurants, their menus and waiting time information.

3. Typically, for more resource-rich devices, *InforMa* may decide not only to store information related to local information providers and the ones obtained while answering local queries, but also accept information that was disseminated by other entities in its vicinity. Therefore, *InforMa* is now more capable and efficient in satisfying queries that originate from its home entity. So, for example, while Bob is walking in the mall, his palmtop may receive various advertisements from the stores Bob passes by and store them to answer future queries.

4. Finally, the most capable *InforMa* instance makes its knowledge available to all entities in its vicinity by accepting their query requests and by actively advertising its knowledge. This is the case when an entity concludes that it will be present at some location for a relatively long duration of time, as is the case of

Bob's palmtop while he and Jane are having a dinner. While in the restaurant, the palmtop may decide to share its metadata knowledge about available information repositories with other entities in its vicinity.

The above design of *InforMa* allows our framework to provide peer-peer interaction among various types of devices regardless of their information and resource limitations.

B. Information Providers

In this section we describe the other component of our framework – the information provider. We discuss what it means for an entity to be an information provider and describe the interaction of an information with *InforMa*.

An entity in our framework is an information provider when it possesses the capability to accept a query and generate an appropriate response based on the body of knowledge under its control. The body of knowledge under the information provider's control consists mainly of facts. These facts could be associated with practically anything in the world, for example the location of gasoline service stations in a certain area and price of gasoline at each station. We can now see that any entity in our framework can provide information about more than one class of knowledge. Thus, for example, an entity might be both a "gas station information provider" and a "weather information provider". We note that it is not required for every entity in our framework to be an information provider. Some devices may be too resource-limited or otherwise restricted to be able to store or share any information at all. These devices simply use information advertised by peers in their environment.

Information providers register themselves with the local instance of *InforMa* by default. Thus, every *InforMa* can now share metadata about information providers with *InforMa*'s. Information providers may also register themselves with remote *InforMa*'s or the latter can learn and cache metadata about these providers from peer *InforMa*'s. In this manner, both information about information providers and the information under the control of every information provider is disseminated to all parts of the ad-hoc environment. We believe that this dissemination mechanism allows for the greatest possible flexibility in managing and sharing the information as fast as possible.

The schema for every information provider must be understood by other entities in the environment. If this is not true, then the information is useless. It is theoretically possible that the schema of all information providers is described in a different language. In this worst-case scenario, the existence of a schema translator becomes paramount. We can easily see that this is not a scalable solution and that the translator quickly becomes a bottleneck, preventing smooth exchange of information. We have, therefore, decided to use a common language to describe the schema for any information provider and chosen DAML for this purpose. We have described the schema for information providers and its management in section V-B.

V. Implementation Details

We have implemented a prototype of the framework on a Bluetooth ad-hoc network, using the Bluetooth protocol stack developed by Axis Communications Inc. [23] and Bluetooth modules developed by Ericsson. We envision that future prototypes of our framework will be implemented on other ad-hoc network technologies as well. The current versions of the Bluetooth specifications and hardware impose some restrictions that smooth prevent transmission of large amounts data under certain conditions. These restrictions may be eliminated in future versions of the hardware and the specifications. In this section we briefly discuss these restrictions and our solutions to them. We also discuss the application level details including the information provider ontology and interaction between various components of the framework.

A. Network Level Details

The design of *InforMa*, as described above, requires that every device be capable of broadcasting messages to other devices in the network. One of the primary limitations of Bluetooth is the lack of a broadcast

mechanism to transmit arbitrary messages. Broadcasting in Bluetooth is restricted to messages used for device discovery. In order to exchange application level messages, a device must first establish a link level connection with its peers. In addition, every communicating device must either be a *master* or a *slave*. Thus, simultaneous link level connections cannot be established between a pair of devices.

To solve the problem described above, we have used the connect-transmit-disconnect procedure. Thus, for example, if *InforMa* on device A needs to query its peer on device B, then device A establishes a link layer connection, transmits the required data for which it receives an acknowledgment and immediately disconnects. Once *InforMa* on device B is ready with the response, then device B waits for a very small period of time (to avoid race conditions) before following the same procedure to transmit the response. Although this solution might be inefficient if the devices are relatively immobile, it is very efficient under conditions of high mobility.

Another issue relevant to our design is the link level Maximum Transmission Unit (MTU) size specified by Bluetooth. This value is set to 672 bytes by default due to a limited buffer size of 800 bytes on the Bluetooth modules. Per Bluetooth specifications, this size allows two Baseband packets of the largest possible size (341 bytes) along with headers to be sent to the module simultaneously. However, this restriction on the MTU implies that packets larger than 672 bytes must be segmented before they can be sent to the module. The Link Layer Control and Adaptation Protocol (L2CAP) layer of the Bluetooth stack is responsible for segmentation and reassembly of packets. However, per the specifications, it only ensures that packets between 341 and 672 bytes large are segmented into two Baseband packets. Packets larger than 672 bytes are thrown away with an error message. The relevance of this issue to our design will become clearer when we describe the application level details in the next section.

The obvious solution to this problem is to use a standard transport protocol like UDP to perform the appropriate segmentation. However, the use of such standard transport protocols presupposes the existence of a specific network protocol like IP. Now, in order to transmit application data using UDP/IP or TCP/IP over Bluetooth is to establish a PPP connection between the devices. The PPP can only be used after a link layer connection has been established between the two devices; in Bluetooth this can be done using the RFCOMM protocol. Thus, the use of a standard transport protocol has introduced four extra layers between the application and the Bluetooth stack. This leads to an obvious degradation in performance of the application.

We solved the problem by using the Service Discovery Protocol (SDP) layer of the Bluetooth stack as the transport protocol. The SDP layer is placed above the L2CAP layer in the stack. Within the SDP layer, we have introduced a segmentation and reassembly procedure to ensure that large application-level data packets are transmitted successfully across the Bluetooth connection [2]. The SDP in Bluetooth is a simple protocol designed to enable devices in a Bluetooth network to query each other for services. Every service and all attributes of the service are assigned a Universally Unique Identifier (UUID). The main advantage of this granularity of description of services is that queries and responses are very small, thus saving valuable bandwidth. The main disadvantage of SDP, from the point of view of *InforMa*, is that no *semantic information* about services can be provided in queries and responses. Reasoning over service related information is not possible. Thus, we have used SDP only for *simple* queries and responses, and to act as the transport protocol for *InforMa*. Of course, SDP can be enhanced to perform matching based on semantic information as described in [2]; however, we have opted to keep the service discovery lightweight.

B. Application Level Details

The most important component of the framework at the application level is the set of ontologies that describe the various types of information providers. We focused our efforts on developing ontologies for information providers that would be most useful for devices in moving vehicles (e.g., Bob's palmtop in our initial scenario). Common, well-known information providers useful to such devices are emergency

related (e.g., police, medical, and fire department vehicles/buildings), traffic and road condition related, weather related and maintenance related (e.g., gas station, towing service etc.). For purposes of proving our system, we have designed and implemented the ontology for gas station information providers. This ontology is based on and very similar to the DAML-S ontology [11], which attempts to comprehensively describe services for the WWW. DAML-S is an extension of the DARPA Agent Markup Language (DAML).

In order to allow our framework to satisfy any generic query, we have decided to adopt a simplified version of DAML-S representation of processes, service models and particular information. Using the DAML-S like description, we are able to match queries with information provider registration information as well as with particular answer instances. A device can, therefore, describe itself by defining the appropriate service models it implements, the process models that provide the information, and the required inputs to be provided.

All messages that are exchanged among *InforMa* instances and among information providers/consumers are based on the Service ontology. This ontology defines the core of all messages by defining the relationships among the inputs, outputs and process model the service is supporting. The advertisement and registration ontologies then further extend the core by adding the necessary descriptions of entities that provide the described information. Finally, the query and response ontologies extend the core to identify requesters as well as the responders. For example, the Gas Station ontology, which is a particular instance of process model, defines that its input is a location, and output is the location of closest gas station and the particular prices for each gas grade.

InforMa uses all the information encoded in the DAML metadata to find the appropriate answer or at least a information provider that could potentially answer the query. In our framework, each *InforMa* first tries to find a valid non-expired answer. If it fails to find some answer, it tries to match for local information provider, a remote information provider, or at least some other *InforMa* that could have a richer cache. The matching is then done by finding the appropriate process model, and validating all inputs and outputs when necessary. We have implemented the current framework using graph and search techniques; however, it is possible more capable *InforMa* entities may also utilize more powerful reasoning techniques using Prolog engines.

In figure 2 below, we show an example of the Gas Station ontology and in figure 3, an example of the registration message sent by an information provider of gas station information. Figure 4 shows an example both a query and the corresponding response.

In our implementation, the gas station information provider, on start up, first sends a registration request to the local *InforMa*, thus registering itself. Every information provider is identified by a locally unique identifier. *InforMa* adds this information provider to its list of local and remote information providers. *InforMa* now knows to route any query related to gas station information to this provider, using the identifier information. On receiving a query, the provider attempts to answer it and sends back its response to the local *InforMa* which routes it back to the source. Renewal of registration information at a remote *InforMa* is the sole responsibility of the information provider. *InforMa* will simply remove the information related to the provider from its table, once the provider's lifetime has expired.

Every *InforMa* has a limited cache in which it stores registration information, queries and answers for a short period of time. Now, depending on its mode of operation, the cache size, the arrival rates of registration information and queries, and the lifetime of the registration information, *InforMa* may or may not be able to answer a certain query. In order to increase the chances of responding positively to a certain query and also to decrease response time, *InforMa* caches responses to previous queries. In addition, the frequency of queries directed to particular information providers determines whether or not information about them will be retained in *InforMa*'s cache or not. Thus, for example, if the gas station information provider registers itself with some remote *InforMa*, but receives no queries, then that *InforMa* may replace it with a more heavily used information provider, even though the former's lifetime may not have expired. We have established

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<rdf:RDF
 xmlns:rdf=     "&rdf;#"
 xmlns:rdfs=    "&rdfs;#"
 xmlns:daml=    "&daml;#"
 xmlns:service= "&service;#"
 xmlns:process= "&process;#"
 xmlns:place=   "&place;#"
 xmlns:mogatu=  "&m;#"
 xmlns=     "&gas;#">

 <rdfs:Class rdf:ID="Gas">
  <rdfs:subClassOf rdf:resource="&m;#ProcessModel" />
  <rdfs:subClassOf>
 <daml:Restriction>
  <daml:onProperty rdf:resource="#sourceLocation"/>
  <daml:cardinality>1</daml:cardinality>
 </daml:Restriction>
  </rdfs:subClassOf>
 </rdfs:Class>

 <daml:Property rdf:ID="sourceLocation">
  <rdfs:subPropertyOf rdf:resource="&m;#inputs"/>
  <daml:domain rdf:resource="Gas" />
  <daml:range rdf:resource="&place;#Place"/>
 </daml:Property>

 <daml:Property rdf:ID="closestLocation">
  <rdfs:subPropertyOf rdf:resource="&m;#outputs"/>
  <daml:domain rdf:resource="Gas" />
  <daml:range rdf:resource="&place;#Place"/>
 </daml:Property>

 <daml:Property rdf:ID="gasInformation">
  <rdfs:subPropertyOf rdf:resource="&m;#outputs"/>
  <daml:domain rdf:resource="Gas" />
  <daml:range rdf:resource="GasInformation" />
 </daml:Property>

 <rdfs:Class rdf:ID="GasInformation">
  <rdfs:subClassOf>
 <daml:Restriction>
  <daml:onProperty rdf:resource="#price"/>
  <daml:cardinality>1</daml:cardinality>
 </daml:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
 <daml:Restriction>
  <daml:onProperty rdf:resource="#grade"/>
  <daml:cardinality>1</daml:cardinality>
 </daml:Restriction>
  </rdfs:subClassOf>
 </rdfs:Class>

 <daml:Property rdf:ID="price">
  <daml:domain rdf:resource="GasInformation" />
 </daml:Property>

 <daml:Property rdf:ID="grade">
  <daml:domain rdf:resource="GasInformation" />
  <daml:range rdf:resource="GasGrade" />
 </daml:Property>

 <daml:Class rdf:ID="GasGrade">
  <daml:oneOf rdf:parseType="daml:collection">
 <GasGrade rdf:ID="LowGrain" />
 <GasGrade rdf:ID="MediumGrain" />
 <GasGrade rdf:ID="PremiumGrain" />
 <GasGrade rdf:ID="87oct" />
 <GasGrade rdf:ID="89oct" />
 <GasGrade rdf:ID="91oct" />
 <GasGrade rdf:ID="93oct" />
 <GasGrade rdf:ID="95oct" />
 <GasGrade rdf:ID="Diesel" />
  </daml:oneOf>
 </daml:Class>

</rdf:RDF>
```

Fig. 2. Gas Station Ontology

a priority-based scheme to allow *InforMa* to determine what information to retain and what to discard. In this scheme, local information providers have the highest priority, followed by remote information providers and finally answers to previous queries. This forms the basis for the cache-replacement policy in *InforMa*.

In addition to supporting registration of information providers, our framework also supports the concept of solicitation of information about information providers. Thus, *InforMa* on a device sends out solicitation requests to its peers, periodically. If a new information provider is discovered in this process, *InforMa* caches the information if it can. The Bluetooth device addresses of the 1-hop neighbors are available to *InforMa* due to the neighbor discovery procedure executed by every device on start up. One important point to emphasize here is that solicitation of information providers from remote *InforMa*'s is restricted to 1-hop neighbors. This prevents unnecessary flooding of this information across the network.

It is possible for a query to travel multiple hops in our framework. Every *InforMa* knows either the final destination of a particular message or a route to it. It obtains this information as follows: if a query or

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<rdf:RDF
  xmlns:rdf=     "&rdf;#"
  xmlns:rdfs=    "&rdfs;#"
  xmlns:daml=    "&daml;#"
  xmlns:service= "&service;#"
  xmlns:process= "&process;#"
  xmlns:place=   "&place;#"
  xmlns:mogatu=  "&m;#"
  xmlns:gas=     "&gas;#">

    <m:ServiceRegistration>
      <m:presents>
      <m:Process>
        <m:processID>4</m:processID>
        <m:InforMaID>0:0 </m:InforMaID>
        <m:processName>
                Gas Station Locator
        </m:processName>
        <m:lifetime>-1</m:lifetime>
        <m:type>provider</m:type>
      </m:Process>
      </m:presents>
      <m:implements>
      <m:ProcessModel rdf:about="gas#Gas" />
      </m:implements>
      <m:inputs>
      <daml:Bag>
        <place:Place />
      </daml:Bag>
      </m:inputs>
    </m:ServiceRegistration>
</rdf:RDF>
```

Fig. 3. Gas Station Location Registration Message

registration arrives on the Bluetooth interface on the local device from a remote device, *InforMa* stores the address of the remote device in its routing table. If, on the other hand, it receives a forwarded query or registration request from a remote device, it notes in its routing table that the source of the message can be reached through the forwarder, unless it already knows how to reach the source. To facilitate this routing mechanism, we ensure that every message contains the Bluetooth device address of the source.

VI. EXPERIMENTS

In this section we describe the experiments conducted to verify the correct functioning of the framework and also to quantify the performance of *InforMa* under certain conditions. The variables in the performance related experiments include *InforMa* cache size, arrival rate of queries, arrival rate of registration information and the number of hops that a query must traverse before it can be answered.

A. Experiment 1

In this experiment, we have attempted to measure both the performance of *InforMa* in terms of processing time and the overall Round Trip Time (RTT) for the query/match/response procedures to execute. This experiment was conducted on two Bluetooth-enabled laptops. The Bluetooth stack is set up on systems and

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<rdf:RDF
 xmlns:rdf=     "&rdf;#"
 xmlns:rdfs=    "&rdfs;#"
 xmlns:daml=    "&daml;#"
 xmlns:service= "&service;#"
 xmlns:process= "&process;#"
 xmlns:place=   "&place;#"
 xmlns:mogatu=  "&m;#"
 xmlns:gas=     "&gas;#">

  <m:ServiceRequest rdf:ID="55">
  <m:requestedBy>
  <m:Entity>
  <m:processID>3</m:processID>
  <m:InforMaID>0:.0.2</m:InforMaID>
  </m:Entity>
  </m:requestedBy>
  <m:implements>
  <m:ProcessModel rdf:about="&gas;#Gas" />
  </m:implements>
  <m:inputs>
  <place:Place>
  <place:ZIP>21250</place:ZIP>
  </place:Place>
  </m:inputs>
  </m:ServiceRequest>
</rdf:RDF>
```

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<rdf:RDF
 xmlns:rdf=     "&rdf;#"
 xmlns:rdfs=    "&rdfs;#"
 xmlns:daml=    "&daml;#"
 xmlns:service= "&service;#"
 xmlns:process= "&process;#"
 xmlns:place=   "&place;#"
 xmlns:mogatu=  "&m;#"
 xmlns:gas=     "&gas;#">

  <m:ServiceResponse>
  <m:implements>
  <m:ProcessModel rdf:about="&gas;#Gas" />
  </m:implements>
  <m:inResponseTo>
  <m:ServiceRequest rdf:ID="55">
  <m:requestedBy>
     <m:Entity>
     <m:processID>3</m:processID>
     <m:InforMaID>0:0.2</m:InforMaID>
     </m:Entity>
  </m:requestedBy>
  </m:ServiceRequest>
  </m:inResponseTo>
  <m:inputs>
  <place:Place>
  <place:ZIP>21250</place:ZIP>
  </place:Place>
  </m:inputs>
  <m:outputs>
  <daml:Bag>
  <place:Place rdf:ID="UMBC">
     <place:Street1>1000 Hilltop Circle</place:Street1>
     <place:Longitude>1</place:Longitude>
     <place:Latitude>1</place:Latitude>
     <place:ZIP>21250</place:ZIP>
  </place:Place>
  <gas:Price>1.29</gas:Price>
  </daml:Bag>
  </m:outputs>
  </m:ServiceResponse>
</rdf:RDF>
```

Fig. 4. Gas Station Location Query and Response Messages

the *InforMa* process is also started. We assume that 10 information providers have already registered with *InforMa*, one of which is the “gas station information provider”. The cache size in *InforMa* is set to 64 entries. A client program one of the devices queries the other for the gas station information provider via the Bluetooth module. The *InforMa* on the “server” receives the query, parses it, determines if it has the answer and if not passes on the query to the “gas station information provider”. The provider parses the query, matches it with its own knowledge and returns the response to *InforMa*, which returns this via its Bluetooth module to the client. We repeated the query/match/response sequence 1000 times and obtained average values for the RTT as 4.56 s and 0.003 s for *InforMa* processing time.

B. Experiment 2

In this experiment, we further attempt to quantify the performance of *InforMa* by varying the cache size from 2 entries to 64 entries. The cache is always assumed to be full. This implies that every time new

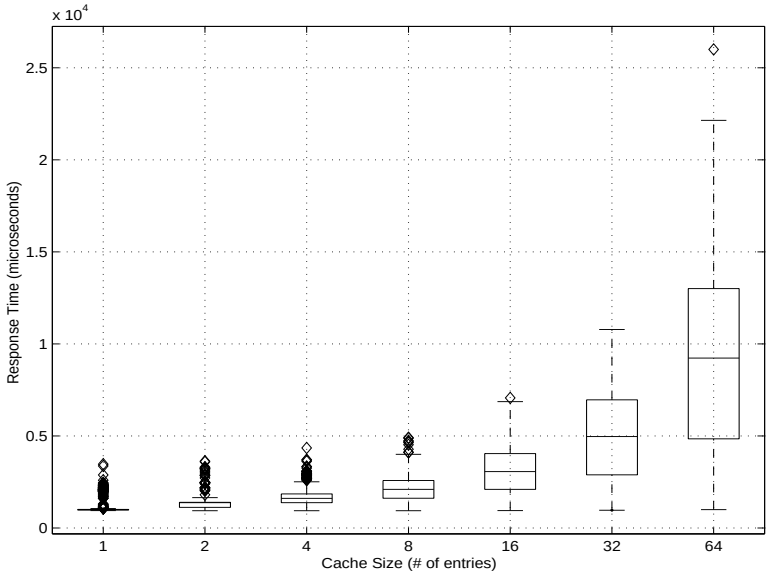

Fig. 5. Response Time of *InforMa* with varying Cache Size (boxplot)

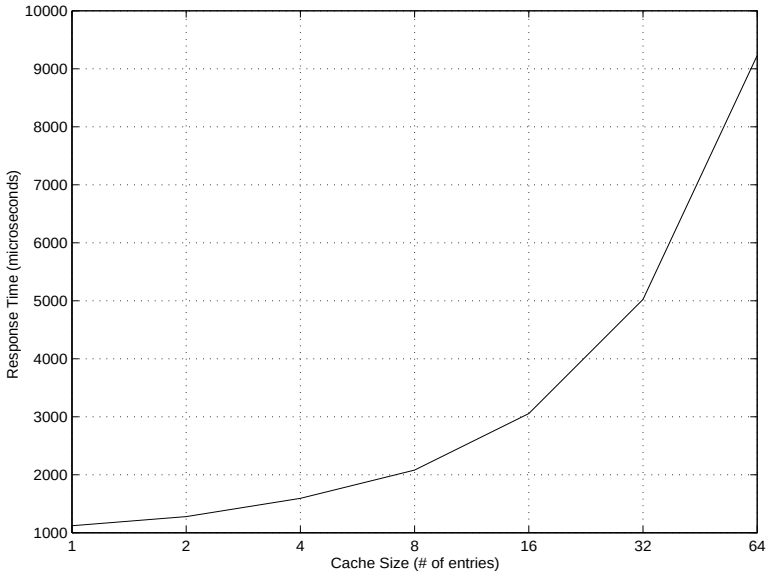

Fig. 6. Response Time of *InforMa* with varying Cache Size (Average Time)

registration information arrives at *InforMa* it must choose a victim provider to replace in the cache. Coupled with periodic or random arrivals of queries, this scenario would lead to a increase in *InforMa*'s response time despite the increase in cache size. As in the previous experiment, the query from the client is for the "gas station information provider". The rest of the query/response procedure is the same as that described above. The main idea behind this experiment is to determine the effect of cache size on *InforMa*'s processing time under high load conditions – periodic registrations coupled with queries for information. We repeated the query/match/response sequence 1000 times. Figure 5 shows a boxplot of the cache size versus response time. Figure 6 shows a plot of the average response time of *InforMa*. As expected, response time of *InforMa* increases with increase in cache size. However, it is clear from both graphs that this increase is sub-linear. Although the cache size has increased 64-fold, the response time has only increased about 9 times. We

should also note that the unit of time in this case is μs. Even for the largest chosen cache size (64 entries), we have observed that the response time is negligible compared to the overall RTT.

C. Experiment 3

In addition to the above experiments, we have also evaluated an existential case when multiple devices need to cooperate in order to satisfy a particular query. In this experiment, the owner of one device in the ad-hoc environment instructs it to find the closest gas station based on the current location. We have simulated this request by initiating the client program, which acts as a user interface, to send a DAML encoded query to its local *InforMa* (located on-device). Once *InforMa* receives the request, it parses the DAML message to convert it into an internally understandable object. It matches the information against the DAML metadata for all information providers and answers it has cached. For this particular experiment, the cache is filled with 10 registered information providers, and one remote provider matches the query. *InforMa* determines the best route to the remote provider's home device, and forwards the request over Bluetooth to it. Once the remote *InforMa* receives the query, it repeats the matching process to determine that it has a local provider that can satisfy the request. It contacts the GasLocator provider and sends it the request. The GasLocator generates an appropriate answer, and sends it back to its local *InforMa* after reversing the sender/location addresses. Its local *InforMa* then forwards the answer to the final destination, which in turn forwards the answer to the client program to display it.

VII. Conclusions and Future Work

In this work we have presented and discussed the need for a robust framework that enables data exchange and querying in mobile *ad-hoc* environments. Existing mobile information access systems require the support of wired infrastructure, thus restricting the flexibility of information exchange among peer mobile devices. We have described in detail the additional challenges to the distributed database framework that arise in the *ad-hoc* scenario. First, the availability of data sources varies with location and time. Second, queries may be both user-explicit or implicit. Next, as information sources are not cataloged a priori, schema translations cannot be done beforehand. Lastly, cooperation amongst information sources cannot be guaranteed due to various security and privacy reasons. We have designed and implemented a framework prototype that addresses these challenges and issues through the use of a standardized semantic language to that helps mobile devices to share information. The primary component of our framework is *InforMa* – a powerful information manager that allows applications to query and obtain responses from their dynamically changing vicinity. We have conducted experiments to validate the correctness of our design and also to evaluate the system, and particularly *InforMa*, in terms of response time.

In subsequent improved versions of our framework, we will incorporate other wireless communication technologies like IEEE 802.11b. In addition, we will enhance *InforMa* matching capabilities via the use of a Prolog-based reasoning engine. In this paper, the main focus has been on addressing the issues concerned with querying and processing, which are similar to read-only mode operations in information access systems. When devices other than the original providers are allowed to update the information, consistency and coherence issues arise, which we also plan to address in future version of the framework. We also plan to introduce modules to enable the design of transaction management systems in mobile *ad-hoc* environments. The developed framework will thus serve a base for transaction management systems that support multiple parties and involve parallel execution of multiple protocols, such as micropayments.

References

[1] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast Disks: Data Management for Asymmetric Communication Environments. In *Proc. of the ACM SIGMOD Conference*, May 1995.

[2] Sasikanth Avancha, Anupam Joshi, and Timothy Finin. Enhancing the Bluetooth Service Discovery Protocol. Technical report, August 2001. TR-CS-01-08.

[3] Harini Bharadvaj, A. Joshi, and Sansanee Auephanwiriyakyl. An Active Transcoding Proxy to Support Mobile Web Access. In *Proc. of the IEEE Symposium on Reliable Distributed Systems*, October 1998.

[4] C. Bobineau, L. Bouganim, P. Pucheral, and P. Valduriez. PicoDBMS: Scaling down Database Techniques for the Smartcard. In *Proc. of the 26th International Conference on Very Large Databases*, 2000.

[5] T. Bray, J. Paoli, and C. Sperberg-MacQueen. Extensible Markup Language. `http://www.w3.org/TR/1998/REC-xml19980210`, 1998.

[6] O. Bukhres, S. Morton, P. Zhang, E. Vanderdijs, C. Crawley, J. Platt, and M. Mossman. A Proposed Mobile Architecture for Distributed Database Environment. Technical report, Indiana University, Purdue University, 1997.

[7] Andrej Cedilnik, Lalana Kagal, Filip Perich, Jeff Undercoffer, and Anupam Joshi. A secure infrastructure for service discovery and access in pervasive computing. Technical report, August 2001. TR-CS-01-12.

[8] Dipanjan Chakraborty, Filip Perich, Sasikanth Avancha, and Anupam Joshi. DReggie: Semantic Service Discovery for M-Commerce Applications. In *Workshop on Reliable and Secure Applications in Mobile Environment, 20^{th} Symposium on Reliable Distributed Systems*, October 2001.

[9] Harry Chen, Dipanjan Chakraborty, Liang Xu, Anupam Joshi, and Tim Finin. Service Discovery in the Future Electronic Markets. In *Proc. of the AAAI 2000 Workshop on Knowledge Based Electronic Markets*, 2000.

[10] Harry Chen, Anupam Joshi, and Tim Finin. Dynamic Service Discovery for Mobile Computing: Intelligent Agents meet Jini in the Aether. *Baltzer Science Journal on Cluster Computing*, 4(4), October 2001.

[11] The DAML Services Coalition. DAML-S: Semantic markup for web services. http://www.daml.org/services/.

[12] S. Czerwinski, B. Zhao, T. Hodes, A. Joseph, and R. Katz. An Architecture for a Secure Service Discovery Service. In *Fifth Annual International Conference on Mobile Computing and Networks (MobiCom '99)*, pages 24 – 35, Seattle, 1999.

[13] A. Demers, K. Petersen, M. Spreitzer, D. Terry, M. Theimer, and B. Welch. The Bayou Architecture: Support for Data Sharing among Mobile Users. In *Proc. IEEE Workshop on Mobile Computing Systems & Applications*, 1994.

[14] M. Dunham and A. Helal. Mobile computing and databases: Anything new? *ACM SIGMOD Record*, 24(4), December 1995.

[15] D. Eastlake, J. Reagle, and ed. D. Solo. XML-Signature Syntax and Processing. `http://www.w3.org/TR/2001/PR-xmldsig-core-20010820/`, 2001.

[16] D. Goodman, J. Borras, N. Mandayam, and R.Yates. INFOSTATIONS : A New System Model for Data and Messaging Services. In *Proc. of IEEE VTC'97*, volume 2, pages 969–973, 1997.

[17] XML Schema Working Group. XML Schema. `http://www.w3c.org/XML/Schema`, 2000.

[18] E. Guttman, C. Perkins, J. Veizades, and M. Day. *RFC 2068: Service Location Protocol, version 2*, 1999. ftp://ftp.isi.edu/in-notes/rfc2608.txt.

[19] R. Guy, P. Reiher, D. Ratner, M. Gunter, W. Ma, and G. Popek. Rumor: Mobile Data Access Through Optimistic Peer-to-Peer Replication. In *ER Workshops*, pages 254–265, 1998.

[20] J. Hendler. DARPA Agent Markup Language. `http://www.daml.org`, 2000.

[21] J. Holliday, D. Agarwal, and A. Abbadi. Database Replication Using Epidemic Communication. In *Euro-Par*, 2000.

[22] J. Holliday, D. Agarwal, and A. Abbadi. Exploiting Planned Disconnections in Mobile Environments. In *RIDE*, 2000.

[23] Axis Communications Inc. *OpenBT: An open source bluetooth stack for Linux*, 2001. http://sourceforge.net/projects/openbt/.

[24] R. John. UPnP, Jini and Salutaion - A look at some popular coordination framework for future network devices. Technical report, California Software Labs, 1999.

[25] Anupam Joshi. On proxy agents, mobility and web access. *ACM/Baltzer Journal of Mobile Networks and Applications*, 2000.

[26] H. Kottkamp and O. Zukunft. Location-Aware Query Processing in Mobile Database Systems. In *Proc. of the ACM Symposium on Applied Computing*, Feb. 1998.

[27] O. Lassila and R. Swick. Resource Description Framework. `http://www.w3.org/TR/1999/REC/rdf-syntax-19990222`, 1999.

[28] S. Lauzac and P. Chrysanthis. Utilizing Versions of Views within a Mobile Environment. In *Proc. of the Ninth Int'l Workshop on Database and Expert Systems and Applications*, pages 408–413, Aug. 1998.

[29] S. Loke and A. Zaslavsky. Towards Distributed Workflow Enactment with Itineraries and Mobile Agent Management. In *E-Commerce Agents*, pages 283–294, 2001.

[30] S. Mazumdar and P. Chrysanthis. Achieving Consistency in Mobile Databases through Localization in PRO-MOTION. In *Proc. of the 2nd DEXA Int'l Workshop on Mobility in Databases and Distributed Systems*, pages 82–89, Florence, Italy, September 1999.

[31] E. Pitoura. A Replication Schema to Support Weak Connectivity in Mobile Information Systems. In *Proc. of the 7th International Conference on Database and Expert Systems Applications*, 1996.

[32] M. Satyaranayanan. Digest of proceedings: Workshop on mobile computing systems and applications - december 1994. *Bulletin of the Technical Committee on Operating Systems and Application Environments*, 7(1):5 – 12, December 1995.

[33] The Bluetooth SIG. The Bluetooth Specifications. http://www.bluetooth.com/.

[34] Carl Tait, Hui Lei, Swarup Acharya, and Henry Chang. Intelligent file hoarding for mobile computers. In *Proc. of the First ACM International Conference on Mobile Computing and Networking - MobiCom'95*, 1995.