**Abstract**— M-Commerce is an emerging discipline involving mobile applications and devices, middleware and wireless networks. Though it is an emerging business application, much of the issues such as location management and application development have not been addressed. Mobile devices are bound to temporary and impromptu loss of network connectivity while they are moved around and have scarce resources. With a vast number of emerging mobile applications, it becomes necessary to provide the users with a common interface- the mobile middleware. The mobile middleware can be defined as an enabling layer of software that can be used by applications developers to connect their m-commerce applications with different systems and operating systems without introducing mobility awareness in the applications. We propose a proxy-based, context-aware, peer-to-peer service-oriented middleware that allows the applications to run with much greater reliability without being too concerned about their location dependency. The middleware provides application developers with primitives to describe how device-specific context changes and the degree of location dependence vary. A proxy-based peer-to-peer architecture is designed to provide a means for small devices to send lightweight requests that are in turn converted into SOAP requests thus relieving the mobile devices of time and processor consuming tasks. This framework simplifies the application development process while identifying the constrained interfaces available at the mobile devices.

**Keywords**— m-commerce, middleware, context-awareness, location management, web services, peer-to-peer.

1. Introduction

M-Commerce is an emerging discipline involving applications, mobile devices, middleware and wireless networks. The estimates made by Gartner Group and the Wireless Data and Computing Service pronounce that at least 40% of B2C e-commerce will be initiated from smart phones supported by WAP and that the mobile commerce market will rise to $200 billion by 2004 [1]. Though it is a fast-approaching business frontier, much of the issues are yet open and have not been addressed. The increasing popularity of mobile devices will require the applications to adapt to changes in context, such as variations in network bandwidth, battery power and so on. Moreover different applications have varying degree of location dependence. To ease the development of context-aware, location dependent applications, the mobile middleware has to provide the application developers with primitives to describe how context changes and location dependence should be handled thus relieving them from dealing with low-level details.

Business Management is like trying to play a game of chess in which the rules change suddenly and frequently. Modern time’s business is based on instant communication and quick information which is made possible by technology. The ‘wired’ Internet drastically changed the way business was done and the mobile revolution will just have as overpowering an effect. The Mobile Commerce opens up whole new avenues of advertising, productivity and workplace efficiency.

Mobile Middleware in particular has to be light-weight in order to run on the resource-constrained devices and should also support context-awareness. Unfortunately the current generation of middleware is, to a large extent, heavyweight and inflexible, and thus, fails to properly address the requirements. Web Services paradigm is a promising technology for developing applications in a heterogeneous environment. When considering the world of mobile devices, the technical challenges associated with the development of the Web Services framework such as low CPU, memory and bandwidth capabilities need to be taken into account. Taking this into account a proxy-based architecture will alleviate the mobile device of the time and processor consuming Web
services related tasks. The mobile devices can send the lightweight requests to the proxy that can be converted into SOAP requests thus providing significant performance improvements [2]. The use of web services in mobile devices opens the possibility to provide services on the spot, no matter where the service requestor or provider is located. Further to overcome the challenges of a single proxy-based architecture, which constitutes to a single-point of failure, a network of trusted proxies in which each proxy can act as a web service requestor for certain services is being suggested.

The motivation for providing infrastructural support is to alleviate the need for mobile applications to handle issues of context awareness and location dependency in an ad-hoc manner thus simplifying application development. Through this, the applications will be able to specify, in a uniform way, which behaviors to adopt in particular contexts and what degree of location requirements are needed. LBS applications address common customer needs to be location-aware:

- Where am I? (location refinement)
- What is around me? (search for points of interest)
- How do I get there? (help in reaching these locations)

Our middleware framework will enable mobile applications to acquire location information while honoring the context information for their individual needs.

The rest of the paper is organized as follows. In Section 2, we present related work and background study. Section 3 presents middleware issues related to location management and context awareness, followed by Section 4 defining the Middleware core and architecture we propose. Section 5 discusses a case study using the proposed framework. Section 6 discusses Future work and we end with conclusion in Section 7.

2. Related Work and Background Study

In this section we give a broad overview of the new classes of middleware built specifically for mobility. However an exhaustive review of all existing mobile middleware is beyond the scope of this paper and we refer the interested reader to [15]. We also discuss the technology we plan to exploit and apply in our proposal.

2. A. Related Work

Licia Capra et al. [3] talk of the traditional object-oriented middleware that have been ported to mobile environment. These object-oriented middleware systems have been adapted to mobile scenarios with little or no success targeting nomadic settings where mobile devices roam while being connected. To support the characteristics of wireless communication media such as frequent disconnections, a middleware that supports asynchronous communication has been proposed as being more viable for developing mobile applications. The paper also discusses the publish-subscribe paradigm which requires the client to connect to the same proxy each time which cannot be guaranteed in a mobile scenario. Licia Capra et al propose the idea of reflection as a principle to accommodate the variety of requirements imposed by mobility.

Licia Capra, Wolfgang Emmerich et al. [4] describes a context-aware reflective middleware system for mobile applications. It argues that the principle of transparency in traditional middleware systems may not be optimal for mobile scenarios. By providing a reflective API, the applications can inspect and alter the middleware behavior in order to achieve better quality of service. However this approach suffers from a drawback that is very crucial to wireless domain – performance.
Chintal Patel et al. [5] offers an attractive way to make SOAP understand application requirements and adapt them accordingly. The paper proposes a Self-Organizing SOAP layer atop the SOAP stack which maps the applications requirements in terms of Quality of Service or domain needs to a set of SOAP features this making the SOAP protocol adaptive and resilient enough to take care of the underlying network fluctuations. This architecture provides a robust and powerful interface to the applications. However it does not handle issues of performance and security.

Kimmo Raatikainen [6] discusses problems related to different protocols such as XML in wireless environment and talks about a binary representation of XML for mobile domain. The eXtensible Markup Language (XML) provides a representation of information that is both easily compiled by machines and readily comprehensible to humans.

T. Pilioura et al. [7] show a strong interest in making mobile devices capable of providing and consuming web services over wireless networks. The paper discusses several scenarios of using web services in mobile devices and spots their advantages and challenges. The paper concludes the use of web services on mobile devices will make it possible to access any data and service, on any device, at any time.

Vinay Bansal et al. [2] analyzes and quantizes the performance of web services on PDAs. It talks of the gargantuan hype and push from major companies such as Sun, Microsoft, and IBM for web services implementations for wireless devices. The paper concludes that proxy server architecture performs the best providing significant performance improvement and should be a starting point in building applications that access web services.

In [8], [9], [11] Upkar Varshney et al. have clearly discussed location management requirements of several m-commerce applications. They divided location requirements into precision, response time and coverage. Table 1 indicates the same.

<table>
<thead>
<tr>
<th>Applications</th>
<th>Precision</th>
<th>Response Time</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobile Finance</td>
<td>Meters (sub-cell)</td>
<td>Seconds</td>
<td>Local</td>
</tr>
<tr>
<td>Mobile Advertising</td>
<td>Hundreds of meters (cell)</td>
<td>Minutes</td>
<td>Local</td>
</tr>
<tr>
<td>Mobile Inventory Management/Product Location</td>
<td>Meters (sub-cell)</td>
<td>Seconds</td>
<td>Local, Possibly nationwide</td>
</tr>
<tr>
<td>Proactive Service Management</td>
<td>Kilometers (multiple cell)</td>
<td>Hours</td>
<td>Local</td>
</tr>
<tr>
<td>Wireless Re-engineering</td>
<td>Hundreds of meters (cell)</td>
<td>Minutes</td>
<td>Nationwide</td>
</tr>
<tr>
<td>Mobile Auction</td>
<td>Hundreds of meters (cell)</td>
<td>Seconds</td>
<td>Local, Possibly nationwide</td>
</tr>
<tr>
<td>Mobile Entertainment</td>
<td>Hundreds of meters (cell)</td>
<td>Minutes</td>
<td>Local</td>
</tr>
<tr>
<td>Mobile Distance Education</td>
<td>Hundreds of meters (cell)</td>
<td>Minutes</td>
<td>Local, possibly nationwide</td>
</tr>
<tr>
<td>Mobile Office</td>
<td>Hundreds of meters (cell)</td>
<td>Minutes</td>
<td>Nationwide</td>
</tr>
<tr>
<td>Wireless Data Center</td>
<td>Hundreds of meters (cell)</td>
<td>Minutes</td>
<td>Local</td>
</tr>
</tbody>
</table>

Table 1. M-Commerce Applications Location Requirements
2. B. Background Study

Web services is an emerging technology that intends at integrating heterogeneous applications. It constitutes a new model for using the web that allows the web publishing of commerce functions and their complete access. Due to the great benefits of this paradigm, such as interoperability and reusability, there has been a strong interest in making mobile devices capable of consuming web services over wireless networks. It opens up the possibility to provide services on the spot, no matter where the requestor or provider is located.

Web services are a form of Web applications that are self-contained, self-describing and modular which can be published, located and invoked across the Web. The Web service model necessitates at least the following basic activities: describe, publish/unpublish/update, discover and invoke. This paradigm revolves around three main roles: Web service provider, Web service requestor and Web service discovery agents. The Web service provider is an entity that provides functions or services, which can be anything from a simple request to complicated business processes. A Web service requestor is the entity that requires a service which is available on the Internet. The requestor could be an application program, another Web service or a human accessing the service through a wired or wireless device. A Web service discovery agent is an entity that provides a repository of service descriptions in which each service provider publishes its services and each service requestor can discover services and obtain binding information for the same. Figure 1 shows a scenario where the entity i.e. mobile device plays the role of the Web service requestor.

![Fig. 1. Web service paradigm with mobile phone as service requestor](image)

The W3C Web Service Architecture Working Group necessitates the use of standard technology for service description, communication and data formats for interaction between the above three entities. The most widely used standards in a service oriented environment are WSDL (Web Service Description Language), UDDI (Universal Description, Discovery, Integration) and SOAP (Simple Object Access Protocol). However, the essential part of Web services is the Interact relationship between a Service provider and Service requestor. Discovery agencies need not be used - they will in some cases but not in others.

In [18] Judith Myerson, Systems architect and engineer IBM, talks about guaranteeing Web services with a service-level agreement (SLA). An SLA is a formal contract between a service provider and a client guaranteeing quantifiable network performance at defined levels. SLAs can be either very general or extremely detailed and generally include the steps that should be taken by the service provider and the client in the event of a failure. The basic SLA Web service architecture is as follows:
The service provider publishes a SLA-covered Web service by creating a Web service and generating a basic SLA for the service. The client finds the appropriate Web service (Service broker may or may not be present) and negotiates with the provider to formalize and finalize the SLA and bind to its Web service.

However while SLA focusses on guaranteed bandwidth, it still needs to specify the average round-trip latency and packet loss over a given period of time since it cannot guarantee consistent response times. Moreover off-the-shelf Web services testing tools are available to check for latency and throughput.

A Peer-to-peer is a communication model in which each entity has the same capabilities and either can initiate a communication session. In general, each communication node, also known as a peer has both server and client capabilities. The main advantage of this paradigm is to do away with the expense of maintaining a centralized server. It also provides a way for businesses to exchange information with each other directly. Figure 2 shows an example peer-to-peer network.

In a peer-to-peer network, all entities are considered equal and have the same abilities to use the resources available on the network. Such a network demands a communication protocol to let the peers speak with each other. Gnutella is a worldwide distributed sharing system which provides an example of software that supports a peer-to-peer communication model. The communication model is based on a constrained broadcast mechanism where each peer forwards its request to all of its neighbors and the response messages are routed along the same path along which the original request arrived by means of dynamically updated routing tables maintained by each peer.

Peer-to-peer networks have the advantage of less initial setup and operating expense and eliminate the single point of failure as in a centralized server, which exposes the whole network to attacks. WEchose P2P over traditional Client-Server Networks for the following reasons:

- Better Scalability without decreasing search time and without need of costly centralized resources
- All nodes are equal in their capacity for sharing information with other members.
- P2P networks use protocols that make it easier for individual nodes to participate and share information.
- In contrast to client-server architecture which relies on a single server storing information and distributing it to clients in response to their requests.

Though peer-to-peer model is quickly emerging as a computer paradigm of the future, there are a number of security issues associated with it that still remain open and are being researched.
3. Middleware Issues for M-Commerce

Conventionally, middleware unites different networks, technologies; allowing a uniform interface. Upkar Varshney defines mobile middleware as “an enabling layer of software that is used by applications developers to connect their m-commerce applications with different networks and operating systems without introducing mobility awareness in the applications” [1].

In a mobile environment, a middleware becomes crucial due to the limited capabilities of mobile devices, the potential value of many applications and the varied requirements needed by these applications. [3] talks of the existing middleware, such as CORBA and Java RMI. Though there have been attempts to adapt these existing technologies to mobile settings, these paradigms assume a permanent connectivity which cannot be assured in a mobile environment. A mobile middleware should be decoupled and opportunistic. In a sense, the computation should proceed even when the mobile device is temporarily disconnected and the middleware should exploit connectivity whenever available. All this favors the use of asynchronous communication for mobile middleware.

Narrow bandwidth, limited computing power, small memory and battery capacity is another concern for mobile middleware. This constitutes the context of the mobile device and by making the middleware context aware; we can enhance the development of context-aware services and applications. There are several scenarios of using web services in M-commerce. Though this service based middleware aims at integrating applications distributed over heterogeneous environments, the use of a mobile device as a web service requester directly makes it too severe due to the mobile device limitations leading to low performance with regards to latency, throughput, etc. With a proxy-based architecture, the mobile device does not need to support web services functionality since the workload is passed over to the proxy thus relieving the mobile device of the time and processor consuming web services related tasks. This opens up the issue of a single point of failure since the proxy controls all user data and decides what services are accessible to the device.

Location dependency is another subject that the mobile middleware should deal with. Different applications have varying location requirements ranging from precision to response times which is a considerable overhead. Though several applications share many common attributes, there are other differences that make them fall in another possible class of requirements. Existing location management schemes used in existing wireless networks provide location information at an area level i.e. to the order of several cells. Most m-commerce applications will require much higher precision possibly in the order of a few meters. Analogous to context information, the applications can pass its location requirements to the middleware and the middleware can map these requirements to appropriate location schemes that are accessible in the current location.

Addressing all the above issues, the middleware we propose provides application developers with primitives to express how context changes and location dependency should be handled using profiles. We propose a peer-to-peer proxy-based web services oriented middleware architecture that provides an asynchronous communication paradigm in which the mobile device can send lightweight requests to the proxy which in turn converted into SOAP requests thus relieving the mobile devices of time and processor consuming tasks.
4. Middleware Architecture

Our Middleware will provide application developers with primitives to express how context changes and location dependency should be handled. This customization will take place by means of application profiles. An application profile provides a way to the applications to specify the location requirements and the context configurations that must hold while requesting a particular service. [4] talks of a context-aware reflective middleware which uses application profiles that defines associations between services that the middleware delivers, the policies that can be applied to deliver the services and the context in which these policies be applied. It provides a reflective API that allows applications to inspect and alter the middleware behavior in order to achieve better quality of service. In this section, we discuss the high level architecture for our proxy-based peer-to-peer service oriented middleware. The middleware components fall in two categories: the mobile device-based component and the proxy-based component. The mobile device-based component provides an API to the application developers to spell out the location and context requirements for the required location-based service. It also interacts with the mobile operating system to extract context information such as memory, battery etc. The proxy-based component uses Web services aware protocols to invoke the required service on behalf of the mobile client and returns the response to the device depending on its context and location requirements. Figure 3 depicts the same pictorially. It just shows one peer.

![Fig. 3. Proxy Server Architecture](image)

We propose the use of eXtensible Markup Language (XML) [3], [6] to encode application profiles. The application profile encompasses all the context and location requirements needed by the application. It also includes the necessary information whether to cache the request and result in case the mobile gets disconnected along with the necessary caching information. Each application profile defines associations between the services that the middleware delivers, the policies that can be applied to deliver the services and the context and location configurations that must hold in order for the policy to be applied. We propose to use the same location primitives as indicated in Table 1.

The following shows an example application profile.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Service xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation=".\App_Prof_xsd.xsd">
  <Device_Identification_Number>1234567890</Device_Identification_Number>
  <Service_Name>UMB</Service_Name>
  <Category>Finance</Category>
  <Cache>
    <Cache_Time/>
    <Unit/>
  </Cache>
  <Description>Financial Transaction</Description>
  <Location>
    <Precision>
      ...
    </Precision>
  </Location>
</Service>
```
The above profile is created for a Finance application to connect to a Bank in the area with Zip 64112. It requires a location precision of less than or equal to 200 meters and a response time of 350 seconds. The application also requires that the result be returned in a compressed Base-64 format if the battery is less than or equal to 45%. A typical response to this request will be as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<Service_Response xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation=".\Response.xsd">
    <Device_Identification_Number>12345 67890</Device_Identification_Number>
    <Response_Success>Yes</Response_Success>
    <Response>
        <Type>text</Type>
        <Name>www.umb.com</Name>
        <Height/>
        <Width/>
        <Length>50</Length>
        <Data_Encoding>Ascii</Data_Encoding>
        <Data>www.umb.com/us/kc/pda/home.htm</Data>
    </Response>
</Service_Response>
```

The response is returned with the information of the closest UMB bank in that area. The result was returned in ASCII since the battery power was above 45% and the suggested policy was not applied. The application can then go ahead and connects to the website designed for mobile devices.

The use of such application profiles eventually opens doors to conflicts. A conflict arises when different policies can be used in the same context to deliver a service. When such a case arises, the middleware does not know which policy to apply in order to deliver the service. Consider an example. James is accessing his PDA at Starbucks Coffee. He needs to access some file which is on his home machine. The constraints he sets up for the file retrieval are as follows:

1) Send file in Compressed format if Bandwidth < 50%.
2) Send file in Encrypted format if Battery > 40%

When James clicks this request, the mobile device-based component of our proposed middleware extracts the necessary context information, creates an application profile and sends it to the nearest proxy (could be the Starbucks Coffee® Server). The Proxy-based component then analyzes the application profile and satisfies the request by obtaining the necessary file in the required format. It is assumed that there is a web service that provides the service of file handling. Everything goes okay and James can access the required file on his PDA in his preferred format. Now let us assume that the same service is requested when the PDA battery is 45% and the bandwidth is 30%. In this case both policies hold true to satisfy the required service and the middleware does not know which policy to apply. This is an example of conflict. There are several conflict resolution strategies available such as First in First Serve or Prioritization. The use of a particular strategy is left to the individual implementation.

Figure 4 shows the end-to-end architecture of the proxy-based peer-to-peer service oriented middleware.

![End-to-End System Architecture](image)

Fig. 4. End-to-End System Architecture

In the above figure, (1) indicates the lightweight protocol between the mobile device and the proxy, while (2) indicates the HTTP/SOAP request/response between the proxy and the Web service providers. The mobile device establishes a TCP connection to the proxy and forwards the application profile along with other context information in a light weight protocol format. The design of this protocol is beyond the scope of this paper.

In the following subsections, we discuss the Context Service and the Service Discovery Peer components of the middleware.
4. A. Context Service

The context service is the mobile-based component of the middleware. It simplifies the development of context-aware applications by placing the context functionality in the middleware. Applications can now just state what context information needs to be monitored by specifying the context parameters and leaving the details of context management to the middleware. The overall Context Service architecture is shown in Figure 5.

![Fig. 5. Context Service Architecture](image)

The Context Service component comprises of the following modules:

- **Context Service API**: This interface allows the applications to specify all their context and location needs while accessing certain services to our middleware.
- **Arbitrator**: Regardless of the type of application requests, they all arrive at the arbitrator through the context service API. The arbitrator then interacts with the other modules as needed.
- **Application Profile Creator**: This module creates an application profile [refer Section 4] based on the primitives [refer Table 1] specified by the application.
- **Context Drivers**: There is one context driver for each type of context information viz. Battery or memory. The Context drivers pull out information from the context sources at the request of the arbitrator.
- **Context Cache**: Context drivers make use of the context cache that retains recently accessed context information in main memory for performance reasons.
- **Context Driver Interface**: This interface is used by the arbitrator to request context information from appropriate context drivers.
- **Dispatcher**: This module encapsulates the application profile and context information in a simple request and sends it to the proxy using the lightweight protocol.
- **Policy Libraries**: These are a set of libraries that are tightly coupled to the operating system to set the appropriate device profiles such as ‘phone alert’ or ‘vibrate mode’ while returning the response to the application as requested. E.g. Jane needs the response in a silent mode since she is in a conference.
4. B. Service Discovery Peer

The Service Discovery Peer is the peer-based component of the middleware architecture. It supports the Web Service functionality thus relieving the mobile device of the time and processor consuming Web service related tasks. The overall architecture is shown in Figure 6.

Each Service Discovery Peer includes the following modules:

*Service Discovery Interface*: The Peer accepts the encapsulated request from the context service component through this interface.

*Web Service Interface*: This provides an interface to access the web services.

*Service Requestor*: The encapsulated request arrives at this module. It interacts with the other modules as needed.

*Service Cache*: To make the middleware asynchronous, this cache holds the application request and the associated response, in case the client gets disconnected. To cache or not to cache is specified by the application through the application profile.

*Service Discovery Component (SDC)*: This module implements the functionality of geographic service discovery. Each Service provider within the geographic area must register itself with the SDC by providing its service details and a basic SLA for the service. Figure 7 show the interface specifications of the service discovery component.

*Service Registry*: Each peer has a local service registry where the parameters of various services are stored once a service is registered. For example, a web service that provides location tracking using GIS/GPS will have a location precision in the range of few to several hundred meters (cell) and will have a response time in the order of few minutes [9]. The register message from the service provider also includes a cache time after which the binding in the local registry must be updated for it to hold valid. This requires an update message to be sent by the service provider to maintain the binding in the service registry.

*SLA Tester*: This module is a COTS tool for testing SLA-covered Web services. It tests the response time and other components of the concerned web service.

*P2P Interface*: This provides the peer-to-peer interface for the middleware. If a peer is unable to satisfy the application’s request, it forwards the request over to the other peers through this interface.

*Peer Manager*: This module handles the peer-to-peer communication of the middleware. There are many existing P2P systems (e.g. SETI@Home, Gnutella) that support collaborative environments.
and we refer the interested reader to [13] and [14]. This module makes the middleware opportunistic. In a sense, it tries to exploit connectivity whenever possible. Whenever a response is cached since the mobile device is temporarily disconnected, the peer periodically broadcasts the device identification number through its P2P interface to the other peers. So when ever the device reconnects possibly to another peer, the response is forwarded to that peer and then to the mobile device.

When a request from the context service component arrives at the Service Requestor, it extracts the service URI and the location primitives from the application profile and encapsulates this in a service request message that is sent over to the SDC. The SDC, in turn is connected to a local service registry and a SLA Tester module. When it gets a request for a service, the SDC runs a matching algorithm to relate the location primitives specified in the application profile with the ones stored in the local service registry. If it finds a match, it uses the SLA Tester module which is a COTS tool for testing SLA-covered Web services to test the response time and other components of that Web service. An acknowledgement or negative acknowledgement is then returned to the Service Requestor by the SDC. If a match is found, the Service Requestor contacts the appropriate web service through the WS Interface using SOAP. Otherwise the request is passed over to the Peer Manager which sends it across to all its peers through the P2P Interface. While contacting a web service, the Service Requester uses the coverage information (if any) from the application profile to get the specified information. The web service may return just the required result (i.e. local domain) or all sets of results (i.e. nationwide domain). The Service Requestor then depending on the coverage domain specified by the application extracts only the necessary information and sends it to the application.

![Diagram](https://via.placeholder.com/150)

Fig. 7. (a) Registering/Updating Service. (b) Initiating service discovery procedure (c) Receiving results

Figure 8 indicates the protocol and the steps used by the middleware.

![Diagram](https://via.placeholder.com/150)

Fig. 8. Protocol used by the middleware
Matching Algorithm used by Service Discovery Component

Location Precision can be divided into 4 classes:

<table>
<thead>
<tr>
<th>Class</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network (4)</td>
<td>&gt; 10,000 meters</td>
</tr>
<tr>
<td>Cluster (3)</td>
<td>Between 1,000 and 9,999 meters</td>
</tr>
<tr>
<td>Cell (2)</td>
<td>Between 100 and 999 meters</td>
</tr>
<tr>
<td>Sub-cell (1)</td>
<td>≤ 99 meter</td>
</tr>
</tbody>
</table>

Table 2. Classes of Location Precision for M-Commerce Applications (a smaller number in parenthesis indicate higher precision)

Most of the M-commerce applications require a precision of a sub-cell or a cell [Table 1]. The matching algorithm can convert the numeric precision value specified by the application into one of the above four classes. For example a precision of several hundred meters would fall into a ‘cell’ precision class. Whenever a service provider registers itself with a peer, and provides its primitives, the SDC converts them into one of the above classes and stores it in its local service registry along with the service name and other parameters.

The other location primitives such as response time of the web service will be specified by the SLA thus guaranteeing quantifiable network performance at various levels. These values can be taken from the SLA and stored in the registry when the service is registered.

Based on this premise, the matching algorithm is as follows:

Algorithm Matching_Service(Service Category, Service Name, Precision, Response Time, Coverage Domain)

1. Check if Service Category is one of the acceptable categories. If not, return failure.
2. Check if Service Name is present in the service registry. If not, return failure.
3. Convert the Precision value into one of the Precision Classes.
   a. If Value <= 99 then Class = Sub-cell.
   b. If Value >= 100 and Value <=999 then Class = Cell.
   c. If Value >=1000 and Value <=9999 then Class = Cluster.
   d. Otherwise Class = Network.
4. Determine if there is an entry in the service registry with category=Service Category, name=Service Name, precision <= Class, response_time <= Response Time and domain >= Coverage Domain.
5. If present, contact SLA Tester with Service IP/Port and required timeout, to test the latency and other components of the Web service.
6. If success, create a Service Reply containing the Service IP/Port and other additional information and return to Service Requestor. Else return failure.

In the above algorithm, a failure indicates that the current peer cannot service the client. In this case, the Service Requestor forwards the request to the peer manager module, which broadcasts the request to all its peers.
5. Case Study

In this section we will use the proposed framework for m-commerce applications. Let’s consider the mobile inventory management class of applications. This class involves tracking of goods, services and possibly even people. The tracking of goods may help service providers in determining the time of delivery to its clients, thus improving relations and obtaining competitive edge over other businesses. Moving inventory from dock to stock has always been a time-consuming and costly process that needs to be improved. With the help of mobile technology, just-in-time delivery of the goods can be performed. We however do not consider certain problems such as determining an appropriate match for the amount of inventory carried by trucks in a geographical area with dynamic changing delivery needs, the traffic conditions, etc… while discussing this example.

An interesting application is “rolling inventory” – wherein in addition to storage at warehouse, products can be kept in the shipping trailers [17]. These trailers are periodically on the move from the warehouse to different destinations. Rolling inventory can reduce the amount of inventory space and cost for vendors and also reduce the time between when an order is placed and the goods are delivered. However these do incur trailer handling and other opportunity costs. Rolling inventory is a B2B m-commerce application.

Figure 9 indicates the location tracking of the trailers using the proposed framework.

![Fig. 9. Rolling Inventory (Mobile Inventory Management)](image)

Since different vendors may use different code names, a uniform product naming system will be required. A particular store requiring a product can connect to a local peer through a proprietary MIM application. The context service component in the store’s handheld/wireless device will create the application profile accordingly and send it over to the connected peer (Peer 2 in the figure above). The peer checks the profile to determine the category of m-commerce application requesting service. It then consults its database to ascertain whether it can satisfy the request. Each location-tracking system has its own limitations in terms of precision and response times. In the figure above, it is seen that Peer 4 has the service that can satisfy the request. Peer 2 forwards its
request to its connected peers and the request propagates from one peer to another till a peer who can satisfy the request is reached (Refer Fig. 8. for protocol). This peer contacts the appropriate web service which determines if there are any trucks carrying the required inventory in the region. The trailers are equipped with Tx/Rx for outdoor tracking. They can make use of Radio Frequency ID (RFID) or ad hoc Personal Area Networks (PANs) for indoor tracking of required inventory. Once the response is returned to the application, the application can take the necessary steps to contact that truck thus providing just-in-time delivery of goods.

There are several cases possible in this example. First, the application could request the middleware to cache the request and associated response to prevent resending the request in case of temporary disconnection. In this case, the application specifies the amount of time the information should be cached. The context service component accordingly creates the profile and sends it to Peer 2. So even if the mobile device gets disconnected, the middleware completes the request and caches the response for the amount of time specified by the application in the profile. Second, it is quite possible that no trailers may be present in the requested region at a particular time. The way around this is to increase the domain of search wherein the application can specify whether the search is local or nationwide. If the domain is expanded, then the middleware can return all the trailers in and around the required region and the application can then contact whichever trailer is closest.

Now we discuss, the step by step approach taken by our middleware to handle this application request.

1. A Store owner or a retailer needing certain merchandise opens a proprietary MIM application that uses our middleware.
2. The application obtains the necessary information from the retailer about the merchandise he is looking for along with the other information.
3. This information is passed over to the context service component of our middleware framework where it is received by the Arbitrator module through the Context Service API.
4. The arbitrator forwards the information to the Application Profile Creator module to create an application profile to handle this request. It may contact the Context Drivers through the Context Drivers Interface, if the application needs certain degree of context awareness.
5. The arbitrator then passes the application profile created to the dispatcher which encapsulates all this information in a simple request and establishes a connection with a nearby peer and dispatches the request to the proxy. The following shows a snippet of the application profile created by the context service component.

```xml
<service xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="/App_Prof_sch.xsd">
  <device_identification_number>134.193.2.1</device_identification_number>
  <service_name>Office Depot</service_name>
  <service_parameters>
    <parameter>ABC 1111</parameter>
  </service_parameters>
  <category>Inventory</category>
  <cache>
    <cache_time/>
    <unit/>
  </cache>
  <description>Determine location of a trailer carrying certain inventory</description>
  <location>
    <precision>
      <operator>&lt;=</operator>
      <value>600</value>
      <unit>meter</unit>
    </precision>
    <response_time>
      <operator>=</operator>
      <value>600</value>
      <unit>seconds</unit>
    </response_time>
  </location>
</service>
```
The above snippet indicates that the application is trying to locate trailers carrying inventory with code ABC-1111 (for Office Depot) in and around area with Zip Code 64110. It also requires a precision of \( \leq 600 \) meters and a response time of about 600 seconds with a nationwide coverage.

6. We assume that there is a web service hosted by Office Depot that provides tracking of its trailers carrying merchandise nationwide. This web service is registered with Peer 4 in our example.

7. The Service Requestor in Proxy (Peer 2 in this case) receives the encapsulated request from the mobile device.

8. The Service Requestor extracts the service URI and the location primitives from the application profile and encapsulates this in a service request message and sends it to the SDC.

9. The SDC runs the matching algorithm to determine if the request can be satisfied. It finds that the service name does not exist in its local service registry.

10. The SDC reports a failure.

11. Upon getting a failure from the SDC, the Service Requestor forwards the request to the Peer Manager. The P2P network of this framework is based on any existing P2P architecture such as Gnutella.

12. The Peer Manager (Peer 2) forwards the request to each of its neighboring peers through the P2P interface.

13. From Fig. 9, both peers 3 and 4 will receive the request from peer 2. Peer 3 will report a failure and forward the request to its neighbors. Duplicate request processing can be avoided by having each P2P query message to include a peer generated unique id.

14. When the request reaches Peer 4, the Peer Manager propagates the request to the Service Requestor which extracts the Service URI and location primitives and sends it to the SDC.

15. The SDC in Peer 4 finds a match in its local registry and contacts the SLA Tester to test the latency and other components of the web service.

16. It returns the web service information to the Service Requestor.

17. The Service Requestor then contacts the web service using the WS Interface.

18. A response from the web service includes all the trailers with inventory code ABC-1111 carrying the required merchandise nationwide.

19. The Service Requestor then creates a response and sends it back. The response travels along the same path as the request flows through specially managed routing tables by the P2P network [13].

20. Once the response reaches Peer 2, it is sent over to the mobile device and back to the application. The retailer can then chose to contact any one to provide just-in-time delivery of goods.

6. Future Work

After proposing our framework for a service-oriented middleware for mobile domain, we now turn to several interesting research problems related to context awareness and location management in wireless world that would be a future step towards worldwide acceptance of m-commerce. [1] clearly indicates that most of the m-commerce applications are location sensitive. Location Management is evidently an obligatory requirement for m-commerce. The role of wireless carriers will be critical since the users will be using their network to perform all m-commerce transactions. Moreover differences in standards has stagnated the growth of mobile commerce applications. The
deployment of 3G networks would greatly assist the growth of m-commerce industry. Applications that could not be easily supported by a given location scheme could be modified to work with less stringent requirements such as lower precision. Embedded global positioning systems is another important area under development.

Security is an ongoing research area in peer-to-peer networks. The open nature of these networks offers an almost ideal environment for the spread of in authentic information. This can somewhat be reduced by assigning certain global trust values to each peer. Encrypted XML profiles can be forwarded among the peers.

Though our proposed middleware provides context awareness to extent where the application can specify the services it wants while holding certain context parameters, context awareness can be used in deciding if certain actions should be completed in certain time by using recent user actions in deriving a context. Currently our middleware provides only a local view of context for each host. An interesting extension would be to enable each device to have a broader view of its environment.

There is ongoing research in provision of Web service at a mobile terminal. With the advancement in the capabilities offered by portable devices, a mobile device hosted Web service provider seems highly foreseeable and technically feasible. This would greatly facilitate the development of context aware applications and location based services. Since these technologies are still in its nascent stages, much stays to be investigated.

M-commerce, in its promising stage, has a lot of barriers to overcome. Managing the complexity and expressing what we desire of mobile business systems is one of the biggest hurdles to overcome. There is always an uncertainty about new mobile business functions.

Our future work will involve in enabling a broader context view by using recent user actions. Moreover rather than indicating a failure back to the client, we propose on working on a negotiation protocol that would enable the client to restate its requirements if they cannot be satisfied by existing services.

Our idea of incorporating peer-to-peer networks in a service-oriented architecture and adopting the idea of using service-level agreement to guarantee web services will definitely prove a robust framework for the onset of m-commerce applications.

7. Conclusion

Wireless access allows the customer to access the Internet in just about any conceivable situation. There is definitely going to be an explosive growth of traffic in the Wireless backbone. We believe there is great potential for mobile devices in business environments as tools for commerce. Our middleware provides a robust distributed environment for mobile domain. It supports a decoupled communication paradigm favorable for developing mobile applications. It allows application developers to serialize their context and location requirements in a uniform way thus avoiding ad-hoc mobile application development. Though the middleware we propose argues against most of the issues in the mobile domain, devices are not able to know in advance which components they will need in which situation. We cannot predict all possible behaviors due to unforeseen context and location configurations and new application needs. Moreover since most of the technologies are still emerging, many challenges prevail, such as performance and security and need to be further investigated.
References

[5] Chintan Patel and Yuyung Lee, University of Missouri Kansas City, Chang Soo Kim and Jong-Woo Kim, Pukyong National University, SOS: Self-Organizing SOAP.
[17] Moving Inventory: Incoming Goods are on the move with Forklift Mounted Mobile Printers, Zebra Technologies International, LLC.