Middleware for Location-based Services

Ricardo Couto Antunes da Rocha
Laboratory for Advanced Collaboration
Pontifícia Universidade Católica do Rio de Janeiro
rcarocha@inf.puc-rio.br

16 de novembro de 2004

1 Introduction

The popularization of mobile devices and positioning technologies like GPS has the increased interest for location-based applications. Such applications use device location to identify in which information or service a user could be interested in. A location-aware tourist guide is an example of that kind of application.

Location-based services and applications are especially useful in mobile computing environments. However, mobile computing environments are intrinsically heterogeneous, since different network technologies could be used and integrated, like GPRS and 802.11. The challenge for researches and industry is to build a common infra-structure that enable the development of mobile applications despite such technological heterogeneity.

In a recent NSF workshop, mobile computing researchers concluded [18] that most of mobile environment infra-structures will be provided by middlewares, allowing traditional interaction paradigms like client-server, and new paradigms, like peer-to-peer and agent-based. Varshney and Vetter [29] argue that middleware could enable the development of mobile applications, independently of the communication media.

Several middlewares [5, 9, 21, 10, 20, 24, 11] have been developed for mobile computing. Although some of those middlewares are general purpose, a few of them [19, 4] have adressed the development of location based applications.

This article discusses middleware requirements and models for location-based services. The remainder of this article is organized as follows. Section 2 describes location-based services concepts. Section 3 describes the main components of software architecture in a location-based service environment. Section 4 discusses middleware requirements to enable the development of location-based services and applications. Section 5 discusses the suitability of middleware models for location-based services and shows which models meet the requirements discussed in section 4. Section 6 presents some conclusions of this article.
2 Location-based Services and Applications

Location-based services (LBS) are services that integrate a mobile devices location or position with other information so as to provide added value to a user [27]. For example, an application that shows tourist sites near from the user according to his current location characterizes a location-based service. Figure 1 shows DoCoMo’s i-area service which allows the user to select services according to his current location area.

![Figura 1: DoCoMo’s i-Area Example](image)

The main aspect that enables LBS is the capability to evaluate device’s location. Technically, since GPS (Global Positioning System) in the 1970s, there has been technology to determine physical location of devices and integrate them with other services. However, integration of such technology with commercial applications has gained attention only after the popularization of mobile phones.

Actually, location-based services are more than a commercial opportunity. Some governments (Europe and USA) have forced mobile telephony industry to implement location services capable to locate users in emergency situations. Due to this obligation, telecommunication companies have implemented additional location technologies which could be used in LBS.

Additionally, several works have developed new location technologies, both for indoor and outdoor positioning. For instance, MoCA architecture [26] uses signal strength of access points to evaluate the position of a device.

There are several uses for location-based services. For example:

- **News**: information dissemination based on the location of a user. Example: weather information.

- **Point of Interest**: shows points of interest near the user. Example: GUIDE project [8].

- **Directions**: shows directions from the current location of a user. Figure 2(b) shows a DoCoMo’s service that shows the map for the current user area.

- **Yellow Pages**: find services near the user. Figure 2(a) shows a DoCoMo’s service that allows the user to select the nearest restaurants.
• **Fleet Management**: tracks positions of a transportation fleet

• **Local Advertisement**: user receives advertisements according to his position

• **Emergency**: tracks current position of a user in an emergency.

• **Location-based games**: player interacts with another player according to his position. Example: DoCoMo’s Mogi game.

Consider the following scenarios:

**Scenario 1**: A graduate student wants to study algorithmics in the university library. So, he accesses an LBS application in his PDA, registers his interest (to study algorithmics) and requests the application to match another student who has the same interest and who is near the library. The application shows three students who match those requirements and the user sends a message to each one to make an appointment.

**Scenario 2**: A coffee shop sends advertisements to mobile users within the neighborhood of 1 km. The advertisement is a publication in a publish/subscribe system, informing that the coffee shop is selling cappuccino. A consumer wants to receive notifications about coffee, but he does not like to drink more than 2 cups of coffee in 5 hours. The consumer registers his interest in the system by a subscription and eventually receives notifications with the coffee shops advertisement.

Scenario 1 characterizes a **point of interest** LBS application because a user can register interest for an object with some attributes. Scenario 2 characterizes a **local advertisement** and **point of interest** application. Section 5 discusses the implementation of those scenarios in different LBS middleware models.

Considering the interaction aspect between the application and the middleware, location-based services can be classified [23] in **push** and **pull** services. In push services, user receives information without having to actively request it from the middleware. The user can **register** interest for information with some characteristics to filter information and receive a **notification** when such information arrives. This approach can be used in the implementation of scenario 2. In contrast, in pull services the user actively requests information from the middleware, so this approach is suitable for the implementation of scenario 1. In [27] there is a detailed discussion about taxonomy for LBS applications.
2.1 Location as a Context

Location is a context information. Context is a set of environmental states and settings that either determines an application’s behavior or in which an application event occurs and is interesting to the user [7]. For example, a context for a mobile user could be characterized by his current location, device type, connection state and his interests.

Mobile computing middleware uses context information to provide adaptability. This adaptation characterizes context-awareness computing, which is useful to provide proximity-based selection, automatic reconfiguration, contextual information and also to fire actions when there is a context change. For example, a tourist guide application could use location-awareness to provide updates in the map shown to the user. The middleware’s role is to allow applications to be aware of context changes and simplify the interaction between applications and the environment.

Mobile computing middlewares provide primitives to handle context information in general, including location information. However, the next sections will discuss cases in which location could be handled in a more specialized way. When compared with other context information, location has the following differences:

- Information can be generated either from devices or externally, from the infrastructure
- Can change in higher rates
- Can be interpreted in different levels (cell, longitude/latitude, zip code)

3 Software Architecture for LBS

A middleware is a software layer designed to facilitate the development and deployment of distributed systems. A middleware helps hiding details of technological infra-structure, typically network communication details, and it is especially useful in heterogeneous environments like LBS one.

For example, in an LBS environment the mobile infra-structure can be implemented either by 802.11 network or cellular network. Therefore, different location positioning technologies can be adopted, e.g. Cell ID and GPS. These different technological choices make hard the interoperability among mobile environments and, in order to achieve it, the Industry has made efforts to define several standards. The OpenGIS consortium [22] and Open Mobile Alliance [2] are examples of such efforts.

However, standards are not enough to support LBS since those applications demand new communication paradigms and high level services. The section 5 discusses middleware models that adequately achieve LBS requirements.

Figure 3, adapted from [17], shows the main components of an LBS architecture. Each component has the following roles:

- **Location position provider**: a component responsible for the generation of location position information.
• **LBS service**: the LBS middleware.

• **Application server**: an application provider where server applications run and use middleware services.

• **Application**: a client application running in the user device.

• **Network infra-structure**: all network components that support low-level communication between environment elements.

• **Subscriber database**: a database that maintains user information such as personal information, billing information and profiles.

• **Common services**: a middleware responsible for common services such as authentication.

4 Middleware for LBS

Several middlewares [5, 9, 21, 10, 20, 24, 11, 1, 3] have been developed to meet the requirements of a mobile computing environment. Some of these requirements are dynamic reconfiguration, adaptability, asynchronous interaction, context-awareness, collaboration and environmental heterogeneity. Most of middlewares can be classified in reflexive, tuple spaces-based, context-aware, data sharing-based and event-based (publish/subscribe). Each middleware proposes a different communication and adaptation paradigm or handle a specific problem of a mobile environment scenario.

This section and the remainder of this paper discuss important middleware aspects that should serve as a guideline to the development of LBS middleware. A few middlewares (e.g. [4, 16, 19]) have been developed specifically for LBS applications. However, since a middleware must be a general purpose solution, this paper focuses on middleware models that can meet the main LBS requirements.
LBS scenarios evolve collecting, analyzing and matching different types of information. Location is not the only information used in LBS scenarios. More complex LBS scenarios evolve other information like user profiles (e.g. personal information and interests) and information dissemination profiles. For each of these information, LBS systems have to handle different aspects [19]:

- **Spatial aspect**: an LBS middleware must be able to collect information about mobile position and fixed elements, associate them with physical/logical maps and match efficiently locations and regions.

- **Temporality**: location information has a temporal dimension that must be included in query capability.

- **Inaccuracy, imprecision and uncertainty**: LBS must deal with inaccuracy and imprecision associated with location positioning technologies.

- **Large Volumes**: in real scenarios, LBS must handle large volumes of data. Therefore, scalability is a very important issue.

- **Continuous queries**: in an LBS scenario, queries executions are continuous. Due to this, a query engine of an LBS middleware must be efficient.

Moreover, there are some general mobile computing requirements that are stronger in LBS scenarios because of its widespread use in different industry segments. Those requirements are mainly related to the heterogeneity of technologies like channels of notifications (e.g. GPRS, 802.11, SMS), positioning technologies (e.g. cell id, GPS, indoor positioning) and positioning concepts (e.g. cell, latitude/longitude, street, zip code). Other important requirements include privacy and security.

The next section describes three middleware models that purposes suitable paradigms to LBS requirements. Some works [4] argue that these LBS application requirements are better achieved if location information is handled differently of other context information.

## 5 Middleware Models

This section describes three middleware models: publish/subscribe, tuple spaces and DBMS-based. The section discusses how these models can meet the requirements of a LBS application.

### 5.1 Publish/Subscribe

Publish/subscribe is one of the most prominent middleware models used in mobile computing. The importance of this model is due to the use of a communication paradigm suitable for disconnected operation and for its efficient information dissemination.

In the publish/subscribe model, communication is defined in terms of exchanging asynchronous messages, also called events. There are two communication elements: the publisher and the subscriber, or consumer. The publisher is an element that populates the system with events. A subscriber is an element that sends subscriptions that register
interest in order to be notified when a specific event is generated.

Communication is mediated by an event service, also called event broker, which is responsible for registering publications and subscriptions and also for sending notifications about events that match a subscription.

The main characteristics [17] of the publish/subscribe model are:

1. Asynchrony: a communication can succeed even if the destination is unavailable.
2. Multipoint Communication: information diffusion is intrinsic because more than one consumer can register interest for a same event.
3. Anonymous and decoupled interaction: a publisher and a subscriber do not need to reach each other.
4. Implicit: the physical destination of the events is implicit.
5. Stateless: events do not persist in the system after being sent to subscribers.

Most works in publish/subscribe systems are done in subscription language and data model. A consumer uses a subscription language to specify his interest in an event. There are four possible approaches [12, 17]: topic-based [28], content-based [6], type-based [13] and subject-spaces based [19].

Content-based and subject spaces models are the only sufficiently expressive approaches to the requirements of a location-based system [17]. The subject spaces model is discussed in the next section.

5.1.1 Content-based Model

In the content-based model, an event has a set of pairs (attributes, values). For instance, in an LBS application an event sent by a mobile user could be composed of the following pairs:

\[((\text{location}, (10, 500, 200)), (\text{battery}, 57\%), (\text{videostream}, \text{not-available}))\]

A subscription contains predicates that describe a value constraint for an attribute name. For instance, the following expression describes interest in events related to users within the region delimited by (10, 20) and (520, 79) coordinates:

\[(\text{location}, (x >= 10) \text{ and } (y >= 20) \text{ and } (x <= 520) \text{ and } (y <= 79))\]

This model is well suitable for information dissemination and for context-aware system architecture. For example, an application or another client can receive notifications about changes in the system state, like location of a mobile unit and devices resources, and execute the appropriate adaptations. In such architecture, location is handled as just another context information.

Location information has two important characteristics: it may change in a relativity high rate and the source of its information can be the infra-structure instead of the mobile device, depending on the positioning technology. Consequently, there are situations
in which it is useful to handle location as a separate aspect.

The L-ToPSS middleware [4] is an extension of the publish/subscribe model that handles the location information as a separately context information. In this approach, in addition to a publish/subscribe filter engine, there is a location constraint matching engine responsible for matching location conditions with the result of the filter engine. The result of this matching is a notification to the interested consumer. Figure 4 shows the architecture of this middleware.

In this approach, a publisher no more needs to send a publication when only the location information changes. It can decrease the overhead of message sending by a publisher.

This middleware maintains two data: subscriptions and location constraints. Instantaneous publications and location information are discarded when the period of location updates expires. Differently from the other approaches, the publisher of a location constraint can be either the publisher or the consumer of the publish/subscribe engine.

5.1.2 Subject Spaces-based Model

Consider the scenario 2, described in section 2. This sample scenario helps to identify a common limitation of the subscription language in a traditional publish/subscribe middleware. After receiving a notification with coffee advertisement, the mobile user should receive a new notification only 5 hours later. However, in a traditional publish/subscribe system either the subscriber receives only one notification or he receives several notifications that do not satisfy desired frequency. For instance, MoCA [26] subscription language allows the user to describe how many times he wants to receive notifications of a matching or mismatching publication but the user cannot describe how a notification can affect new notifications, as required in the coffee shop example.

Such a problem happens because the content-based publish/subscribe model is stateless, i.e., it does not maintain the state of publications and subscriptions. On the other hand, the subject spaces model [19] is a stateful model that can address more adequately
not only the subscription described in the coffee shop example but also other location-based application scenarios.

In a subject spaces model, a publish/subscribe system is describe as a set of subject spaces. Subject spaces are the metadata of a publish/subscribe system and they help to describe the values and relationships among publications and subscriptions.

Formally, a subject space is a tuple \( \{D_o, V_o\} \) where \( D_o = \{d_1, d_2, ..., d_n\} \) is the set of dimensions of the subject space and \( V_o \) is the set of values allowed in this space. Therefore, a subject space is a multidimensional space. Each dimension is defined as a tuple \( d = \{\text{name}, \text{type}\} \) where \text{name} is a unique identifier for the dimension and \text{type} is a data type. For example, a subject space \textit{location} could be defined as \( D_{\text{location}} = \{(x, \text{double}), (y, \text{double}), (z, \text{double})\} \).

The subject space model defines three relationships among subject spaces.

1. \( \sigma_1 \subseteq_1 \sigma_2 \): \( \sigma_1 \) fully contains \( \sigma_2 \) or \( D_{\sigma_2} \subseteq D_{\sigma_1} \)

2. \( \sigma_1 \subseteq_2 \sigma_2 \): \( \sigma_1 \) partially contains \( \sigma_2 \) or \( D_{\sigma_2} \cap D_{\sigma_1} \neq \emptyset \)

3. \( \sigma_1 \subseteq_0 \sigma_2 \): \( \sigma_1 \) and \( \sigma_2 \) are unrelated or \( D_{\sigma_2} \cap D_{\sigma_1} = \emptyset \)

Moreover, the subject space model also provides formalisms to describe:

- **Region**: a restriction in a domain of a subject space that specifies a sub-domain.
- **Object region**: an object state or object properties.
- **Interest region**: represents the set of values within a subject space in which a user is interested in.
- **Matching relation between regions**: an operator that describes the relationship between two regions. There are four operators: containment, enclosure, overlap and nearest neighbor.
- **Filter**: a function that maps a set of regions to a true or false value. It is used in a subscription to describe if a set of regions can be applied to the described interest.
- **Time**: is handled as a discrete domain. The subject space model defines two functions to handle time: a \( \pi \)-function which returns the most recent time instant when two regions have matched, and a \( \tau \)-function which returns the most recent time instant when two regions turned from a match or a mismatch.
- **Subscription**: specifies a set of region interest and a filter function which defines when object regions can satisfy the subscription criteria. Interest region can be in different subject spaces.
- **Publication**: specifies a set of object regions and a filter function that specify when an interest region can match this publication.
These definitions are enough to check if a publication matches a subscription, as shown in [19]. In a stateless publish/subscribe model, when a subscription matches a publication, a notification is sent and a subscription is removed by the event broker, unless the subscriber wants to be continuously notified at any applicable match.

In a stateful publish/subscribe model like the subject space, no subscription is removed by the broker. Conversely, this model maintains subscriptions and sends notifications only when the system state changes. The system state is defined by global subject spaces and can change in three situations:

- Adding a publication or a subscription;
- Updating a publication or a subscription; and
- Removing a publication.

A subscription can be removed from the system, but no notification needs to be sent in this case.

A complete and detailed description of this model can be found in [19]. This description shows that the formal subscription language of this model is enough to accurately fit not only the coffee shop example but also other LBS scenarios where a notification sending depends on the former matching between publishing and subscriptions.

The subject space model has three main advantages:

- Avoiding redundant notification as seen in the coffee shop example;
- Implementing a symmetrical publish/subscribe system. In a symmetrical publish/subscribe system, a notification is only sent if the publication matches the subscription and vice versa. This characteristic is important for an LBS because it allows the user to filter out unwanted publications and avoid useless information dissemination;
- When the location of a user changes, the model updates this information in all subscriptions. In a traditional publish/subscribe system, when a location changes all subscriptions must be removed and reinserted in the system.

5.2 Tuple Spaces

The tuple space model was originally proposed to coordinate concurrent activities in parallel programming [15]. Several tuple space-based middlewares have been implemented as a proposed model to coordinate concurrent tasks between concurrent and decoupled processes. InfoSpaces [25], JavaSpaces [14] and TSpaces [30] are examples of such middleware. This model has also found applications to coordination in mobile computing environments [24].

In tuple space model, a process communicates with another process by using a tuple space. Tuple is a data element that contains values or specifies data type. A tuple space is a global collection of tuples used in the system. Several processes share a tuple space and communicate with each other by inserting, reading or changing tuples. Tuples are
persistent in a tuple space.

Communication based on tuple spaces is typically decoupled and anonymous. In order to achieve synchronous communication, elements must send information to the tuple spaces and wait for a response tuple generated by the addressee element. Some middleware, e.g. TSpaces, implement blocking tuple operation which facilitates the implementation of synchronous communication.

Implementations of a tuple space middleware must support three basic operations:

1. out(t): inserts a tuple t into the tuple space.
2. in(p): extracts a tuple t from the tuple space which matches a tuple template p.
3. rd(p): is similar to in(p) operation, except that a tuple t is not removed from the tuple space.

The tuple space model is sufficiently flexible to support the development of location-based applications. In this context, information producer and information consumer interact with each other by inserting and extracting tuples in the tuple space [17]. Therefore, a tuple space model comprises a pull-based interaction scheme.

Although tuple space model proposes very elementary operations to exchanging information, modern middleware [30] supports the creation of new operators (handlers in TSpaces terminology). This capability is essential to create really functional location-based systems.

5.3 DBMS-based Model

The DBMS-based model comprises the use of database interaction model to implement a communication and coordination model. This model supports a pull-based interaction scheme, similar to tuple space model, but offers well known and flexible relational operations to retrieve information.

This approach is widely used in industries and many geographical information systems operate according to this scheme. Moreover, other components of location-based service architecture naturally use a DBMS-model, e.g. user management systems and accounting information systems.

6 Conclusion

Middleware is an interesting approach for implementing interoperability between heterogeneous platforms and rapid application development. On the other hand, the increasing number of middleware proposals and implementations inhibits its adoption, mainly in commercial applications.

Location-based services introduce new requirements that must be handled efficiently by middleware architectures. Some of these requirements are handling efficiently large volumes of data and matching adequately information publication with subscriptions.
However, the development of a new middleware for LBS would be a mistake. Future research must be directed on adaptation of current middleware models and paradigms to LBS requirements.

Referências


