Mobile Spatial Subscriptions for Location-Aware Services

by

Kah-Kuen Fu

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2010

AUTHOR'S DECLARATION

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Spatial subscriptions have been used to specify locations of interest in Distributed Event-based Systems (DEBSs). However, current DEBSs representations to support spatial subscriptions are not expressive enough to describe some forms of subscriptions in mobile settings. For instance, users are not allowed to specify a spatial subscription that refers to other more well-known locations, in case they are not familiar with the names of their current locations. In addition, the middleware in existing DEBSs does not support changes at runtime, and modification to these middleware systems to support spatial subscriptions are highly coupled with specific DEBS infrastructures.

In this thesis, I argue that by enhancing the expressiveness of spatial subscriptions, a new model of mobile spatial subscriptions for location-aware services can be defined and a reusable plug-in implementation approach that supports existing DEBSs can be developed. This thesis first summarizes the essential abstractions to specify mobile spatial subscriptions, and analyze the expressiveness of existing DEBSs to support these abstractions. Second, it proposes a three-level mobile spatial subscription model, which supports the essential abstractions used to specify spatial subscriptions. The first level of the model handles subscriptions consisting of geometric coordinates; the second level supports subscriptions with location labels; the third level interprets subscriptions which specify locations by stating their dynamic properties. Next, a plug-in implementation approach is introduced, and hence, the three-level model can be integrated with different DEBSs with minimal modification to the middleware. The subscription model is implemented as a subscriber/publisher component, instead of directly modifying the existing DEBS. Finally, I develop a prototype system, Dynamic Mobile Subscription System (DMSS), and illustrate the usefulness and applicability of the three-level model and the plug-in implementation approach.

Acknowledgements

I would like to express my sincere gratitude to my advisor Professor Paulo Alencar for his inspiration of my Master's studies and research. Throughout my thesis-writing period, he also provided encouragement, advice, and lots of good ideas.

I would also like to thank Professor Ladan Tahvildari, Professor Donald Cowan and Professor Daniel Berry for reading my thesis and providing helpful comments.

I thank my fellow labmates: Rolando Blanco and Eduardo Barrenechea provided guidance and help for my research, as well as reviewing my work. Yu-Ling Chang was particularly helpful guiding me through the difficult times, and for all the emotional support, camaraderie, and caring she provided.

Lastly, I wish to thank my family in Hong Kong. Without their support, I would not be able to study in a foreign country, Canada.

Dedication

To my parents.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

1.1.1 Motivation

With the advancement of technology, lightweight mobile devices are becoming prevalent and affordable for ordinary people. These devices, such as smart phones, PDAs and other portable handheld devices, in conjunction with wireless network access and global positioning system (GPS) support can serve as platforms for mobile-networked applications.

These mobile devices usually have a physical keyboard, touch screen or microphone as human input, or have sensors to gather information automatically. In addition, they usually consist of a screen to display updated information received. Therefore, by carrying such devices, users can form network groups, and can share and receive information in real-time from the other users in the network.

A current popular application of these devices is Location based Services (LBS) [2-9]. Having GPS or another positioning sensor integrated with these devices, users no longer need to update their current locations manually. Instead, these devices are able to acquire their geographic location in regular time intervals. Hence, a device's location can be embedded into messages to be delivered, and users can choose the message they want to receive according to the locations where messages are issued. An example is to acquire real-time traffic information by using a GPS-assisted mobile device during driving. On one hand, the device receives, either directly or indirectly, traffic information from other devices. For instance, the device might need to acquire constantly those traffic information based on its current location. On the other hand, the device also disseminates the traffic of its location, such as travel speed, to the other devices, so that the others can analyze and estimate the traffic by using the most updated information.

1.1.2 Distributed Event-based Systems (DEBSs)

Traditionally, information sharing among devices adopts the client-server approach [10]. As a result, each device establishes connections to other devices to check whether information was updated. As a result, the number of connections increases factorially with the number of devices in a network. Moreover, each of the devices does not know in advance whether other devices have updated information, or such information is relevant or not. Therefore, a large number of unnecessary connections are established, and hence resources, including limited computation power, battery, and network bandwidth, are wasted.

To solve these problems, researchers have investigated a new paradigm, Distributed Event-based Systems (DEBS), to connect the devices in a network. DEBS, also known as Publish/Subscribe systems, are push-based middleware that support the exchange of information [10].

DEBSs are composed of three major components: publishers, subscribers and event brokers. Publishers are information providers that issue events, and subscribers consume these issued events. Events are the data, or information, transmitted from one agent to other interested agents. Subscriptions, issued by subscribers, specify the types of events in which subscribers are interested. Middleware, also known as an event broker, is responsible for providing a bridge between publishers and subscribers. When a publisher issues an event, the event broker notifies the corresponding subscribers who have specified their interest in the published event. Figure 1 shows a simplified architectural view of DEBS, where the arrows represent data flows.

Existing DEBSs can be categorized into different types depending on their filter models, which support different data types in subscriptions and employ different matching mechanisms. Three common types of filter models will be analyzed in this thesis, including subject-based, type-based and content-based. The filter model affects the capability of adopting different location models and hence leads to different location-based abstractions that can be supported. A more detailed description of existing filter models will be provided in Section 2.

In DEBSs, each device only establishes one single connection to the middleware, regardless of the number of devices in the network. The middleware acts as a hub connecting all the devices together. Thus, the whole network becomes more scalable, as long as the middleware is able to handle the total number of connections established by the devices.

Moreover, the middleware aims at disseminating only updated relevant information to the devices, according to their subscribed interests. As a result, connections for checking updates are no longer needed. Therefore, the overhead of the system is reduced, and both computational power and network bandwidth are saved.

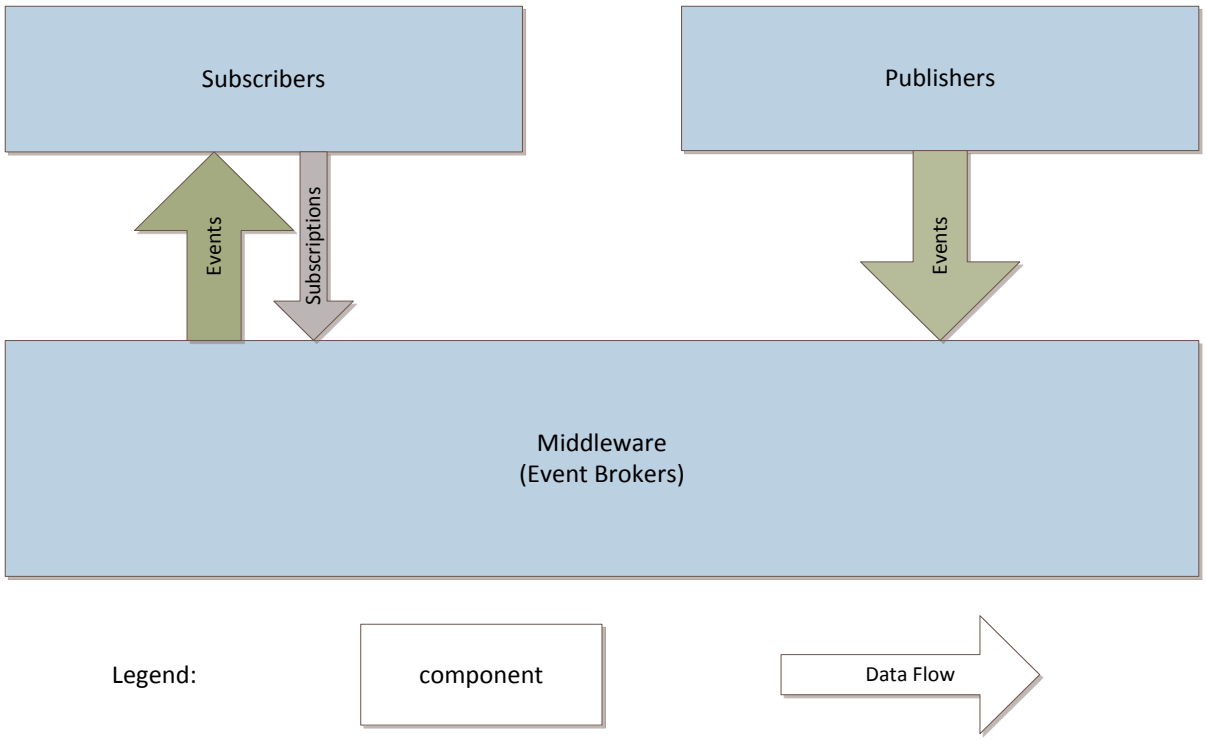

Figure 1 Architectural view of a simplified DEBS

1.1.3 Spatial Subscriptions

As mentioned in the last sub-section, subscriptions describe events of interest, which are related to at least one knowledge domain. For example, expressing a subscription *bus arrives at the central terminal at 6PM* involves the transportation, spatial and time domains. This thesis focuses on Location-based Services (LBS), and hence the scope of subscriptions is limited to the spatial domain, also known as spatial subscriptions.

LBS provide users with information based on locations [5, 9]. As a result, each event published has to be associated with a location, and hence subscribers then specify their locations of interest by issuing spatial subscriptions.

Spatial subscriptions play an important role for subscribers to express precisely their interests. In fact, spatial subscriptions are more than specifying locations by their names, such as *central terminal* in the previous example. The spatial subscriptions used in DEBSs are not only governed by the chosen filter model, but also the location model. The existing filter models will be illustrated in Section 2.4, and the location models are discussed in Section 2.3 and the following sub-section.

1.1.4 Location models

There are two common types of location models, namely the symbolic location model and the geometric location model [11]. A geometric model defines locations by using a coordinate system. Locations are represented as a point, a set of points or even an equation based on a coordinate system. In principle, every location can be expressed, as long as the granularity of the coordinate system is fine enough to represent such coordinates. A symbolic model defines locations by using pre-assigned labels, which resemble the human perception. For instance, people use *Toronto* and *Canada* to represent a city and a country. As these names are mostly assigned based on historical reasons, there is no systematic way to label these locations, and hence only pre-labeled locations can be referenced.

The actual expressiveness of these location models may be degraded when combining with different filter models in DEBSs, as some of the filter models have some restrictions when specifying spatial subscriptions. I will further analyze different combinations of these models and provide a detailed explanation in Section 2.4.

1.2 Current Problems

Although the DEBS paradigm solves some of existing problems mentioned in the last section, this paradigm raises other problems when used to support mobile spatial subscriptions, which are summarized in the following paragraphs:

Lack of expressive notations

Ordinary users do not have expertise about geometric locations. By deploying a geometric location model, users are required to remember the physical coordinates of locations. Obviously, it is not a feasible solution. For those devices with touch-screens, users may select locations of interests on a map. However, difficulties may arise when specifying a location in 3-dimensional space, such as a room inside a building, by using a 2-dimensional screen. Moreover, it is not easy to select the same area on a screen every time, and would lead to generating different spatial subscriptions even

selecting the same location. The filtering and optimization processes would become more complex for the middleware.

By deploying a symbolic location model, users can specify only pre-labeled locations. Sometimes, however, users may forget the names of locations in which they are interested. In real-life situations, users may refer to other more well-known locations. For example, they could specify *the country north of the US* to refer to *Canada*. Unfortunately, symbolic location models do not support these types of expressions. The subscriptions refer to other locations are particular useful in mobile environment, as users may move to a unfamiliar environment and do not know the exact name of some locations, as most are not well-known.

Lack of run-time modification approaches

Another problem is that the whole system has to be halted while adding spatial subscription support to existing running DEBS, as modifications have to be made to the middleware. All mobile devices and servers are required to be updated at the same time and, as a result, the whole system needs to be halted until the update procedures are done. Hence, it may not be feasible for large scale systems which are already deployed on numerous mobile devices.

Lack of re-usable solutions

Modifications to DEBSs to support spatial subscriptions are system-specific. Therefore, different strategies are needed in order to enable different DEBSs to support spatial subscriptions. Consequently, code reusability is not easily achieved.

1.3 Proposed Solution

In this thesis, I will solve the problems mentioned in the last section in two stages. First, a three-level spatial subscription model will be introduced. Second, this model will be integrated into existing DEBSs based on a plug-in approach.

The subscription model is designed to facilitate most types of spatial subscriptions. In the model, spatial subscriptions are divided into three categories, and each level of the model handles its corresponding category of subscriptions. The first level handles spatial subscriptions based on geometric coordinates; the second level handles the subscriptions based on the labels of locations and spatial relations; the third level handles the subscriptions based on location labels and their dynamic properties.

Following a plug-in approach, the model is then separately implemented as a normal subscriber/publisher component. The component subscribes to all spatial events. When spatial events are received from the middleware, the component converts the events into new types of spatial events and publishes them. Therefore, the corresponding subscribers will eventually receive those events. The component acts as a spatial subscription conversion unit. Although this solution creates some communication overhead, no modification of the existing middleware is needed. In addition, the component can be plugged into different DEBSs with only a few modifications.

1.4 Contribution

In this thesis, I argue that by enhancing the expressiveness of spatial subscriptions, a new model of location-based subscriptions can be defined and a reusable plug-in implementation approach that supports existing DEBSs can be developed. The contribution of this thesis includes:

Analysis of the expressiveness of spatial subscriptions of existing DEBSs

First, I summarize the essential abstractions to specify mobile spatial subscriptions, and analyze the expressiveness of existing DEBDs to support these abstractions.

A three-level model for spatial subscriptions

Second, I propose a three-level mobile spatial subscription model, which is to support the essential abstractions used to specify spatial subscriptions.

A plug-in implementation approach

Then, a plug-in implementation approach is introduced, and hence, the three-level model can be integrated with different DEBSs with minimal modification to the middlewares.

A case study: illustrates the applicability of the models proposed

At last, I develop a prototype system and illustrate the usefulness and applicability of the three-level model and the plug-in implementation approach.

1.5 Thesis Outline

This thesis is organized as follows. Chapter 2 reviews the existing DEBSs and analyzes their expressiveness to support mobile spatial subscriptions. Chapter 3 proposes a three-level spatial subscription model which is able to handle most types of spatial subscriptions. Chapter 4 illustrates a plug-in approach to implement the subscription model. Chapter 5 shows a simple prototype system,

Dynamic Mobile Subscription System (DMSS), which demonstrates the procedures needed in order to support mobile spatial subscription in existing DEBS. Finally, Chapter 6 concludes the thesis and articulates the directions of future work.

Chapter 2

Related Work

2.1 Overview

In this section, I will discuss how different types of existing DEBSs can be extended to support spatial subscriptions. I first summarize the abstractions used to describe a location. Then, I compare different location models and discuss their capabilities to handle these abstractions. Finally, I illustrate how different location models are adopted by different DEBSs, and hence analyze the expressiveness of DEBSs to support mobile spatial subscriptions.

2.2 Location-based Abstractions

To handle spatial subscriptions, a DEBS should be able to interpret the expressed locations. The interpretation processes rely on various location-based abstractions to resolve the spatial expressions. In fact, location-based abstractions can be classified into primary identities and secondary identities. In the following sub-sections, the definitions of primary and secondary identities are discussed. Then, various abstractions for spatial relations and the use of these abstractions to express location is illustrated.

2.2.1 Primary identities

Primary identities are location coordinates that are interpretable without referring to other locations. They can be expressed in geometric or symbolic formats [11]. Geometric coordinates are expressed as points or geometric figures in a multi-dimensional Euclidean space. However, they are often not practical for ordinary users, who do not have expert knowledge to specify complicated-expressed location, do not use or remember physical coordinates. Symbolic coordinates, on the other hand, are a form of abstract labels, such as city names and street names. In contrast to the geometric coordinate approach, whose spatial relations can be calculated mathematically based on geometry, the symbolic coordinate approach requires their spatial relations to be modeled explicitly, since these relations cannot be deduced merely from the labels.

2.2.2 Secondary identities

A secondary identity defines a location by stating its relative position to another location and consists of two parts. The first part is the primary identity of a reference location and the second part is its spatial relation to the actual describing location. For example, *Canada* can also be referred as *the country to the north of the USA*, where *the USA* is (the primary identity of) the reference location and *the country to the north of* specifies the spatial relation. As DEBSs are designed for ordinary users who normally rely on common sense to specify locations, expressing the spatial relations should not require expert knowledge.

2.2.3 Abstractions For Spatial Relations

Location-based abstractions are the concepts used to describe the spatial relations between locations. Event brokers rely on these abstractions to interpret the location expressions or the spatial relations between locations. In this sub-section, the essential abstractions are summarized [7, 12-15] and each abstraction is briefly introduced and their roles in expressions are discussed.

2.2.4 Equivalence

Two locations are equivalent if they cover the same area. However, evaluating equivalence relations is not a trivial process. Determining whether two location expressions are the same is not simply matching their names word-by-word, but involves matching physical areas, as areas might have more than one unique name.

Equivalence is a basic and essential abstraction to express locations. Expressing a location by its primary identity relies on the equivalence concept; expressing secondary identities also relies on the equivalence concept, as they contain the primary identities of other locations.

2.2.5 Containment

A containment abstraction defines a location *cover* or *is covered by* other locations. The essence of containment in expressing primary identity lies in the possibility that location can be expressed in various degrees of granularity. Consider a subscriber showing an interest in *Toronto* and a publisher issuing an event related to *Ontario*. In human perception, they should be matched. However, by simply deploying the equivalence concept, the event does not match with the subscription because of the two areas are not exactly the same. On the other hand, expressiveness of secondary identities can

be enhanced by using the containment concept. For example, the province *Ontario* can be expressed as *the province contains Toronto*.

2.2.6 Intersection

An intersection abstraction is used in secondary identities to describe the regions which *cover or are covered by* some, but not all, of the area of a reference location. For example, *the streets intersecting with King Street*. In addition, by using intersection, the locations that overlap with a set of disjoint regions can be expressed more easily. An example is an expression *the regions intersecting with the bus stops*. Without the intersection relation, each region overlapping with a bus stop has to be specified individually.

2.2.7 Adjacency

An adjacency abstraction is essential in expressing secondary identities. In some cases, end users may not know the primary identities of certain locations, but they do know these positions relative to a more well-known nearby location. Hence, the adjacency concept helps to express the locations in relation to a reference location.

2.2.8 Distance

A distance abstraction can be used in secondary identities to describe numerically the relative distance to a reference location. There are two major types of distance abstractions. The first type is physical distance, which can be easily obtained using the geometric coordinates of two objects. However, the physical distance may not be representative of the actual distance or effort needed to traverse different locations, as there is no direct straight path between two locations in most situations. Hence, the physical straight distance is not enough to express actual distance.

The second type is logical distance, which represents the actual paths or the actual distances required to travel between different locations. In some cases, the logical distance between two locations is long even though the corresponding physical distance is short.

2.2.9 Direction

The direction abstraction, such as North/South/East/West and Left/Right, is used to refine the other location concepts, which can only specify non-directional relations. For instance, directions combined with distance functions are useful for drivers, such as subscribing to an event *2km down a road*. The concept is used to express secondary identities.

The essential location-based abstractions used to express locations are summarized in Table 1.

	Equivalence	Containment	Intersection	Adjacency	Physical Distance	Logical Distance	Direction
Expressing primary identities	●	●					
Expressing secondary identities	●	●	●	●	●	●	●

● = the location-based abstracts required

Table 1: Location-based abstractions used to express primary and secondary identities

2.3 Location models

Location models define the data structures representing locations and also the spatial relationships between locations. There has been substantial effort devoted to proposing location models aimed at enhancing the expressiveness of spatial relations [11, 12]. In terms of expressiveness and modeling effort, no model appears to be superior. This section briefly reviews the existing location models that can support symbolic coordinates. Geometric location models, which are usually not feasible for use by ordinary users, are not included in the discussion. The reviewed models include the simplest model, in terms of implementation effort, being adopted in most of the existing DEBSs, and those sophisticated symbolic models in the GIS literature. In addition, mapping layers used to enhance the capabilities of supporting location-based abstractions are introduced.

2.3.1 String-based models

String-based models are the simplest type of location models in terms of the required modeling effort. The primary identity of each location is represented by a unique abstract label. Two locations are equivalent only if the labels are literally the same. Containment relations can be achieved by suffix match. For instance, the province *Ontario* and the city *Toronto* are expressed as *Canada-Ontario* and *Canada-Ontario-Toronto*. The former contains the latter as the former is the prefix of the latter. However, by merely comparing two strings, two locations with different primary identities are presumed to be different even if they cover the same area. Hence, equivalence and containment are

supported but have limitations. Another disadvantage of the approach is that application developers have to know the *contained by* relations of a location before specifying it.

An extra mapping layer can be overlaid on the model to support the other location-based abstractions. The layer consists of a mapping table, translating the secondary identities to their primary identities if their covered areas are physically equivalent. For instance, *USA-adjacency* is mapped by the layer to *Canada* and *Mexico*. As the model is string-based and does not support numerical relations, both the physical and logical distances between locations have to be expressed in terms of discrete ranges. For example, *Toronto_short* is mapped to *NewYork* and *Toronto_far* is mapped to *Vancouver*. More than one set of relations can be defined for the same abstraction to facilitate different travelling agents, as the definitions of short/far vary from agent to agent. Although the exact numerical information is lost, the expressiveness is enough for ordinary end users, who do not usually care about the exact values.

However, this mapping process requires manual effort, which can increase factorially with the number of locations in the model. Consider that there are L locations. Theoretically, each location can have maximum R types of relations to the other L-1 locations. Thus, the possible number of mapping relations is $RL!$, which increases dramatically as L increases. Moreover, if the position of an object is changed, $RL(L-1)$ relations in the mapping table have to be updated.

Significant numbers of DEBSs adopt string-based models to evaluate the relations between spatial subscriptions/events, as string pattern matching is supported by all types of DEBSs. However, the mapping layer is not deployed in the existing models. As a result, only limited equivalence and containment spatial relations can be evaluated.

2.3.2 Set-based models

Set-based location models are formed based on a predefined set of elementary location objects. The primary identities of location can be created by grouping one or more objects as members, and they can be further derived into different subsets. In contrast to the string-based model, the set-based model evaluates the spatial relations not by comparing the labels of the sets, but the objects contained in the sets. Thus, the equivalence, containment and intersection abstractions can be supported. Consider there are two non-empty sets, X and Y:

- Equivalent: $X = Y$
- Containment: X contains Y if $X \cup Y = Y$; Y contains X if $X \cup Y = X$
- Intersection: X and Y intersect if $X \cup Y \neq \varnothing$ AND $X \cap Y \neq X$ AND $X \cap Y \neq Y$

By comparing the elements contained in the sets, locations with different names can also be matched. As a result, equivalence and containment abstractions can be supported more comprehensively compared with those of string-based location models.

The model is also able to express adjacency abstraction. Each adjacent pair is denoted by a set, such as *{Canada, USA}, {USA, Mexico},* and a set L_{adj} contains the entire pairs record the adjacency relations. Although the *set* notations have no way to express distance and direction abstractions numerically, as mentioned, the numerical relations can be expressed in the notation of discrete ranges. However, set is merely capable of representing binary relations, as an object is either contained or not contained in a set; therefore, the discrete ranges with more than two values cannot be expressed.

A possible solution is to create new sets manually which are named by their relations to the other primary identities, such as *Toronto-far = {Vancouver, …}* and *Toronto-short ={ NewYork, …}*. The approach only requires *RL* mapping definitions to express all the relations between locations, instead of *RL!* in the mapping layers of string-based models, where *R* is the number of the types of relations and *L* is the total number of locations. When the position of an object is changed, *R* numbers of relations have to be updated.

2.3.3 Graph-based models

Graph-based location models represent locations and their relations in terms of vertices and edges. Vertices, in form of labels, are the symbolic coordinates of locations. Two vertices are linked by edges if the two locations they represent are directly connected. The edges can be weighted with numerical values to represent the distance or the time needed to pass through two vertices.

Similar to string-based models, graph-based models regard two vertices as equivalent only if the names of the vertices are exactly the same. Moreover, only the vertices expressed in the same degree of granularity are allowed to connect to each other and locations with two degrees of granularity will be grouped into two disconnected graphs. For example, the graph with vertex *Canada* cannot be connected to the graph with vertex *Toronto*. Thus, graph-based models are not able to handle the containment abstraction.

Among the four models introduced in Section 3, the graph-based model is the only one that is capable of directly supporting the logical distance between locations. The degree of precision depends on the number of vertices modeled along the paths between locations. Consider person *X* is traveling from *Toronto* to *New York City* by car. *X* has to pass through different locations via various highways,

while each segment of highway may have a different speed limit. The more vertices added to an edge, the more precise the model. The precision can be adjusted depending on the actual requirements of applications. In addition, the model can provide different sets of information through weighted edges. For example, a person can walk through a forest, while a car has to follow the road outside the forest. Hence, the weights between locations may not be the same for different agents. Physical distance can also be supported by defining another set of weights.

The adjacency abstraction is also naturally supported as graph-based models are constructed by connecting vertices.

2.3.4 Refined geometric-based models

Refined geometric models, also known as hybrid models, are formed by overlaying a symbolic layer on top of geometric location models [11]. The overlaid symbolic layer can be seen as a mapping layer. The representations of locations are in the form of literal labels, but they are translated, by the mapping layer, into geometric figures for the spatial relation evaluation processes. The geometric figures are represented by a group of coordinates in the Euclidean space and all the relation evaluations between locations are based on coordinate geometry. Hence, except for logical distance abstractions, the secondary identities of locations are well supported. To support logical distance abstractions, as mentioned in section 2.2.3, paths between locations have to be modeled explicitly. By using this mapping mechanism, only a specific coordinate in the mapping table has to be updated when its geometric coordinate is changed.

One of the challenges is to ensure the consistency of the translation of the representations from symbolic to geometric coordinates, since there may exist more than one way to express a location, especially when the shape of the location is complex. Moreover, different degrees of approximation may be adopted, which makes the situation even worse. Another challenge is the computational complexity, which is high for evaluating the relations between irregular-shaped locations.

2.3.5 Expressiveness of the locations models

The capabilities of the four location models to handle the location-based abstractions are summarized in Table 2, which indicates clearly to what extent the location models support the relevant location abstractions. For example, the refined geometric-based model does not support *logical distances*. Other hybrid models can be deployed to solve the problem, but the discussion of

hybrid model formation is beyond the scope of this paper. Interested readers can refer to [11, 12] for a more detailed discussion.

	Equivalence	Containment	Intersection	Adjacency	Physical Distance	Logical Distance	Direction
String-based	◒	◒	○	○	○	○	○
Set-based	●	●	●	●	○	○	○
Graph-based	◒			●	●	●	
Refined Geometric-based	○	○	○	○	○		○

● = completely supported ○ = Supported with limitations (extra mapping layer is required)

◒ = supported with limitations ○ = Completely supported (extra mapping layer is required)

Table 2: The capabilities of various location models to support the location-based abstractions

2.4 Different Filter Models of DEBSs

Existing DEBSs can be classified into different types by their filter models, which define how subscriptions can be specified and how events of interest are specified. In this section, three common types of filter model DEBSs, which include subject-based, typed-based and content-based, are briefly reviewed. The ways to deploy the existing location models to different types of DEBSs are discussed. Then, each capability that supports location-based abstractions is examined. The detailed formation of each type of DEBSs can be found in [16-18] .

2.4.1 Subject-based

By using subject-based DEBSs, end user applications subscribe/publish to predefined topics, which can be seen as channels exchanging information [16, 18, 19]. The topics are often categorized into a hierarchical tree/lattice-like structure. Thus, subscribing to a topic T means becoming a member of T and T's ancestors; publishing an event to T means forwarding the event to all the members of T and T's descendants. The topic names are in the form of strings and are organized in directory-structure-like notations to form a hierarchical structure. During the matching process, pattern matching is performed on the addresses expressed in the events and subscriptions.

2.4.1.1 Adopting a string-based location model

Subject-based DEBSs are able to support the string-based location model, as the model also relies on pattern matching on the location labels to evaluate spatial relations. The location labels can be organized into directory-structure notations based on containment relations to form a hierarchical structure. In this circumstance, spatial events, or abstract location labels, are grouped into tree/lattice-like structure based on spatial containment relations. For instance, events about *Toronto* can be grouped into the topic of */Canada/Ontario/Toronto*. Subscribing to the topic */Canada/Ontario* means receiving all the events grouped under the topics of */Canada/Ontario* and also its lower level ones including */Canada/Ontario/Toronto*. The filter model depends on string operators, such as prefix match or suffix match, to determine whether spatial events match subscriptions. As hierarchical trees are formed by containment relations, equivalence and containment relations can be expressed. For example, equivalence relations can be evaluated by whole-string match and containment relations can be evaluated by prefix match. However, other spatial relations such as directions cannot be evaluated based on the containment-based trees. Nevertheless, because of the relatively simple implementation, traditional DEBSs rely on this mechanism to handle spatial events/subscriptions [5].
As discussed, a mapping layer can be used to support other spatial relations. The mapping layer of the model can be implemented as a symbolic-link handler mapping the secondary identities' address to the primary identities' address before comparing the topics names. For example, a label */USA/North* can be mapped to */Canada* so that direction relations can be used in subscriptions. However, modeling the mapping layer requires lots of effort, which increases factorially with the number of locations in the model.

2.4.1.2 Adopting a set-based location model

To the best of my knowledge, no existing research has been done on investigating the adoption of a set-based location model in subject-based DEBSs. An approach would be to organize the sets into containment structures and represent them by directory-structure-like addresses. An example is shown in Figure 2. Subscriptions to the events issued in USA have to specify the topic */North_America/USA*; subscriptions to */North_America/USA* means subscribing to *USA* and also its descendants. However, application users have to know the physical containment relations before they can specify the containment based addresses. In addition, this approach violates some of the fundamental design purposes of subject-based models - to minimize the matching effort for the events and subscriptions in the event broker. The location relations between locations are evaluated by

comparing the location objects contained in the sets instead of performing pattern matching on the addresses. As a result, this approach requires extra effort to evaluate the desired channel stated in the events/subscriptions. Thus, this approach appears to be impractical.

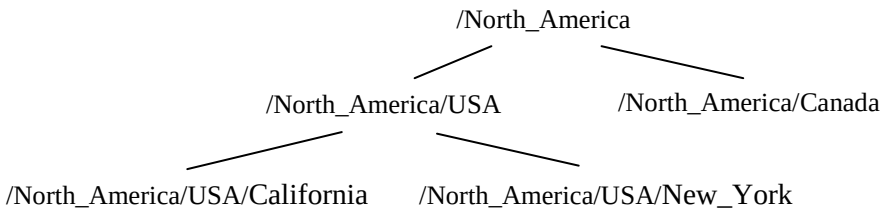

Figure 2: Set-based representations in subject-based systems

2.4.1.3 Adopting graph-based/refined geometric based model

The expressions should contain two parts to describe locations in graph-based models. The first part describes the reference position and the second describes the range. As each of the vertices has a unique label, equivalence relations can be evaluated by performing pattern matching on the labels of the vertices. However, subject-based DEBSs are not able to handle comparable types such as real numbers, as there are only limited numbers of predefined topics but an unlimited possible range of numbers. Thus, the weighted edge of graph-based models and the refined geometric location models that depend on numerical coordinates are not able to be expressed.

2.4.2 Type-based

Type-based DEBSs, a variation of the subject-based ones, classifies events based on their types instead of their topics [17, 20, 21]. Different types of events have different schemas, which are instantiated from the predefined objects in an object oriented language (OOL). Comparison of types is naturally supported by an OOL such as Java. Similar to subject-based approach, events can also be organized into hierarchical structures, which are facilitated by schema inheritance.

In contrast to the other three types of filter models, type-based models do not rely on the event content to propagate events, but are determined merely by the structures representing them, as subscribers issued their events of interest by specifying the types of interested schemas. For instance, two locations shown in Figure 3 are regarded as the same because their data structures are the same. Thus, the events disseminated to a subscriber may not be the contents the subscriber was intended to receive. In order to distinguish the two locations, different data structures are required for different locations. Under the circumstances, each location needs to have its own object schema, which violates

the design purpose of the type-based DEBSs. Obviously, it is not a practical solution in terms of both implementation and system maintenance. As a result, DEBSs with type-based filters usually cooperate with content-based filters to enhance expressiveness, by imposing constraints on the contents of the events [16, 17, 20].

<schema>
<Country>US</Country>
<City>New York</ City >
</schema>

<schema>
<Country>Canada</Country>
<City>Toronto</ City >
</schema>

Figure 3: Two locations expressed in the same schema

2.4.3 Content-based

In content-based DEBSs, subscribers do not issue their request by specifying a particular predefined topic, but specify the content of interest that may possibly appear in events. Compared with subject-based DEBSs, content-based ones offer more flexible subscription expressions, although extra processing power is needed for the real-time evaluation in event brokers. Usually, subscriptions impose a range of values on attributes of events while events only define a certain value for each attribute.

A content-based filter is a Boolean expression that consisted of predicates which are combined by Boolean operators, such as AND and OR. A subscription is considered to be satisfied if all the conditions stated in its expression are true. Evaluations, which take place in event brokers, examine whether the constraints stated in subscriptions match with events. The matched events are then propagated to the particular subscribers [22-25].

Content-based filter models can be classified into two major types: primitive type expressions and executable code expressions. In the former expressions, filters are in the form of name-value pairs which consist of a named attribute, an operator and a constant. The operator and constant represent constraints imposed on the named attribute, i.e. X > 7. Most systems, such as SIENA [26], only allow the use of predefined primitive types in subscriptions.

CORBA(Object Management Group (OMG), 2000) is the only system allowing the use of abstract data types in subscriptions. In fact, filters consisting of abstract data types can be regarded as one of the types of the executable code expressions [27]. In these expressions, filters are expressed in terms of invocations, which call the self-defined libraries in the event broker or runtime executable code specified by application developers. In addition, the operators of abstract data types can also be

implemented as invocations. This type of expression allows almost any expression, maximizing the expressiveness of subscription filters. To the best of my knowledge, no existing DEBSs support spatial invocation expressions.

In the following paragraphs, I discuss my first idea of deploying the type of expressions. As different programming languages have different sets of predefined abstract data types, in the discussion below, I assume the primitive types are those existing in C.

2.4.3.1 Adopting a string-based location model

By adopting string-based location models, two locations are equivalent if two location names are exactly the same. Because of the high degree of flexibility in name-value pair subscriptions of content-based systems, containment relations can hardly be handled as subscriptions and events may not agree to the same granularity of expressing locations. For instance, a subscription *Location=Toronto* cannot be matched with an event *Location=Canada* by merely comparing two strings. To tackle this problem, [28] proposed a subject space-based filter limiting the flexibility of expressions so that application users can only impose constraints on the predefined dimensions. As a result, if the dimensions of the subject space *Location* contains *Country* and *City*, the subscription should be *Country=Canada AND City=Toronto* and the event would be *Country=Canada AND City=Null*. Hence, the containment relations can be evaluated based on string pattern matching. In the literature, most systems expressing locations as symbolic coordinates adopt a string-based location model because of the relatively simple implementation.

Executable code expressions can also be used to support string-based location models. Consider a spatial expression *Country=Canada* used to subscribe to events with the attribute *Country* equal to *Canada*. The expression can be replaced by another expression involving an invocation *country("Canada")*, provided that event brokers are able to translate the function call to its corresponding implementation shown in Figure 4.

For the other location-based abstractions, a mapping layer has to be implemented in order to translate the secondary identities into primary identities before the matching process. Therefore, only invocation expressions can support this type of relation. For example, application developers may design a *country_adjacency(loc)* as illustrated in Figure 5. This filter is implemented as a mapping layer which translates *loc* to its adjacent locations. However, as mentioned, a lot of effort is required to implement mapping layers, which may not be practical when the number of locations in the model increases.

```
country (loc) {
        if event.country == loc
           return true
        else
           return false
     }
```

Figure 4: Executable code for country()

```
country_adjacency (loc) {
        temp[] = adjacency(loc)
        if event.country ⊇ temp[]
           return true
        else
           return false
    }
```

Figure 5: Executable code for country_adjacency()

2.4.3.2 Adopting a set-based location model

Set is not a predefined primitive type in C. Therefore, the set-based location models are considered not to be supported by filter expressions based on primitive types only. Nevertheless, the abstract data type *set* and its operators, including *equivalence*, *containment* and *intersection*, can be declared. Each location can be declared as an instance of the *set* type so that the locations can be referenced in the subscriptions. For example, three instances of *set* type, *street1={shopA, shopB}, street2={shopB, shopC}, street3={shopA, shopB},* are declared. Then, a subscription *street1.intersection* is able to match with street2, as both *street1* and *street2* contain *ShopB.* Similarly, an expression *street1.equivalence* can be matched with *street3* because they contain the same objects.
For other operations such as adjacency, the relations between two sets cannot be determined by comparing their containing objects, and therefore, a mapping layer is required, which can be implemented based on either abstract data types or invocation expressions. The relations can be handled by defining new location sets named by primary identities plus spatial relations. For instance, the countries close to the *USA* can be defined as a new set *USA-short = {Canada, Mexico}*. The relations between sets are stored by a mapping table. A synchronous mechanism is needed to share the declared instances between event brokers and applications so that the location sets can be expressed in the spatial subscriptions/events.

2.4.3.3 Adopting a graph-based location model

To express graph-based filters, distance and direction abstractions are required. Primitive types are not capable of expressing the filters in subscriptions. By using abstract data types, distance and direction abstractions can be implemented as operators. For example, a distance abstraction expressed based on abstract data type is *Location distance (Toronto, 7km)*, where *Location* is an attribute,

distance is operator and *(Toronto, 7km)* is a constant. By using executable code expressions, distance and direction abstractions are expressed as invocations. Attributes, operators and constants can be expressed in term of parameters of an invocation, such as *Location("distance", "Toronto", 7km)*.

2.4.3.4 Adopting a refined geometric location model

As a mapping layer is needed in order to translate symbolic representations to geometric ones, primitive types are not capable of expressing the spatial relations in refined geometric location models. Similar to the approach used in graph-based models, abstract data types and executable code filters can be used to express all kinds of spatial relations supported by the refined geometric location model, although the actual implementation for the invocations is different.

2.5 Existing Spatial DEBSs

Although much research has been devoted to developing DEBSs support for LBS, to the best of my knowledge, none are designed to enhance the expressiveness of spatial subscriptions.

Most existing generic DEBSs , such as [22, 23, 26, 29-31], support only simple predicate in subscriptions. Therefore, for subscriptions formed by symbolic locations, the matching process is performed by simple string pattern comparison. Hence, only string-based subscriptions are supported by those systems, which are not able to evaluate the locations specified by their actual physical areas.

Another disadvantage of string-based subscriptions is that users have to know the exact labels of the locations defined in the system. To the best of my knowledge, none of the existing DEBSs can support spatial subscriptions reference to locations relative other more well-known locations, as the two location expressions are not exactly the same in words. However, this is a common practice in real-life experience for people forget a location's name. As a result, the expressiveness of existing spatial subscriptions is not expressive enough.

Recently, researchers focused on the subscription used to monitor a series of events, which are commonly used in composite event-based systems or spatio-temporal event-based systems such as [32-38]. In those systems, a subscription is defined as a match when a series of specified events happened, while these events may or may not need to be happened in a specific order, depending on the actual implementation of such systems. On the other hand, in this thesis, I focus on the subscription used to specify the dynamic properties of a single location, while these properties can be

changed in real-time. For example, the subscriptions specify *the temperature or precipitation level of a city*, and *the number of people in a shop*.

L-ToPSS [39], a LBS system based on a DEBS, adds an extra location processing module to a typical event broker to manage the possible spatial events and subscriptions. The system aims at supporting window queries and N-nearest queries. Based on L-ToPSS, [40, 41] proposes efficient algorithms for the location constraint evaluation. CAMEL [42, 43] is a push-based middleware construct based on a database. Similar to L-ToPSS, the system is designed to support window queries and N-nearest queries. The papers describe the spatial event model, the spatial subscription and the overall architecture of the system, but the representations of locations are not discussed.

In fact, most of the research, such as [5, 32, 37, 44-47], investigates how the constraints or predicates in subscriptions can be evaluated more efficiently. In contrast, this thesis aims at enhancing the expressiveness of spatial subscriptions for existing DEBS.

2.6 Summary

Table 3 shows the capability of handling the location-based abstractions of each type of DEBSs while combining each with the location models. It summarizes the extent to which the filter models support the relevant location-based abstractions based on specific location models. For example, the type-based filter model does not support any of the location-based abstractions independently of the location model used. Further, the content-based filter model only partially supports the equivalence and containment abstractions when a string-based location model is used.

As previously mentioned, it is possible to adopt two location models in a single DEBS, but the discussion is beyond the scope of this thesis.

	Subject-based							Type-based						
Location abstractions	E	C	I	A	P	L	D	E	C	I	A	P	L	D
String-based	◒	◒	○	○	◡	◡	○							
Set-based [a]														
Graph-based [b]	◒													
Refined Geometric-based														
	Content-based (primitive type expressions)							Content-based (abstract data type and invocation expression)						
Location abstractions	E	C	I	A	P	L	D	E	C	I	A	P	L	D
String-based	◒	◒						●	●	○	○	◡	◡	○
Set-based [a]								●	●	●	○	◡	◡	○
Graph-based [b]								◒				●	●	
Refined Geometric-based								○	○	○	○	○		○

a = comparable type evaluation support is needed

b = graph computation support is needed

E = equivalence, C = containment, I = intersection,
A = adjacency, P = Physical distance,
L = Logical distance, D = direction

● = Completely supported

◒ = Supported with limitations

○ = Completely supported (extra mapping layer is required)

◡ = Supported with limitations (extra mapping layer is required)

Table 3: The capabilities of DEBSs to support various location-based abstractions after deploying different location models

Chapter 3

A Three-level Mobile Spatial Subscription Model

3.1 Overview

The core part of spatial subscriptions is to specify the locations of interests by using appropriate expressions. In this chapter, I present a three-level mobile spatial subscription model which is able to handle the abstractions needed for this propose. The requirements of mobile spatial subscriptions will be discussed, and then, I will illustrate the three levels of the model by using a bottom-up and incremental approach.

3.2 Mobile Spatial Subscription Requirements

Before introducing the three-level model, I discuss the essential requirements for mobile spatial subscriptions in this section.

3.2.1 Precise Spatial Expressions

One of the most basic requirements of spatial expressions is to allow users to describe locations in a precise way. This requirement implies that spatial expressions should be deterministic and should not be ambiguous. A spatial expression should not denote different locations from different users' point of view. For instance, *bus stops* should have the same meaning for all users in the same time frame.

In addition, users should be able to specify precisely any location, provided that such a location exists in the real physical environment. However, this might not be possible for all existing location models [11]. For example, a 2-dimensional model is not able to model precisely a 3-dimensional object.

3.2.2 More Expressive Spatial Expressions

The second essential requirement of expressing spatial expressions is that the subscriptions can be specified in more expressive ways, since most ordinary users do not have expertise in specifying geometric locations and do not know how to express locations in formal ways.

Unfortunately, there are conflicts between the previous requirement and this one. One of the problems of using natural languages is that they are sometimes ambiguous, and one expression may be interpreted as several different locations. To ensure locations can be preciously specified, a formal naming approach can be employed. However, such approaches usually require users to have technical knowledge and to use sophisticated statements to express locations, such as the ones involving geometric coordinates. For example, *x = 100 AND y = 50*. Consequently, in order to use technical expressions, the learning effort for users is high.

3.2.3 Runtime Extensions

Another requirement for specifying spatial expressions is that location sets should be extensible at runtime. In general, there is an unlimited number of locations and system developers can only register a limited number of locations in the system. As a result, there is a chance of specifying non-registered location expressions at runtime. Thus, a comprehensive model should provide ways for users to specify unregistered location expressions.

3.2.4 Dynamic Spatial Expressions

The last requirement I focus on is to handle dynamic spatial expressions, which are mapped to different locations over time. In mobile DEBSs, objects move and change to different locations in different time frames. Therefore, the spatial relations between objects also change over time. By assuming this change, a single spatial expression can be mapped to different location over time. For example, a dynamic spatial expression, such as *100m ahead of John,* can involve moving objects and spatial relations in its expression.

3.3 First level: Primitive Elements (PEs)

In this section, I begin presenting the first level of the model. I focus on the Primitive Elements (PEs) used to specify locations. Users should be able to specify precisely any locations by using PEs.

In the first level, I aim at satisfying the precise and extensible expression requirements. As discussed in Chapter 2, the specification of spatial expressions relies on a location model, and there are two major types of location models, namely the symbolic location model and the geometric location model. There are both pros and cons for both of them.

Symbolic models do not satisfy the requirements I have set in the level, as they neither provide systemic ways to define locations, nor are extensible at runtime. To add a new location to a symbolic

model, extra effort is needed to define its spatial relations with respect to other existing locations. As mentioned in Chapter 2, the spatial relations between locations increase factorially with the number of location objects. Therefore, it is almost impossible to register all locations, in case the location set is too large. As a result, there might not be enough registered locations to represent precisely the locations in the real environment.

In order to allow users to express all the possible locations, I propose to deploy a geometric location model to specify locations in the first level of the model. As mentioned in Chapter 2, geometric model provides all the primitive elements needed to specify locations.

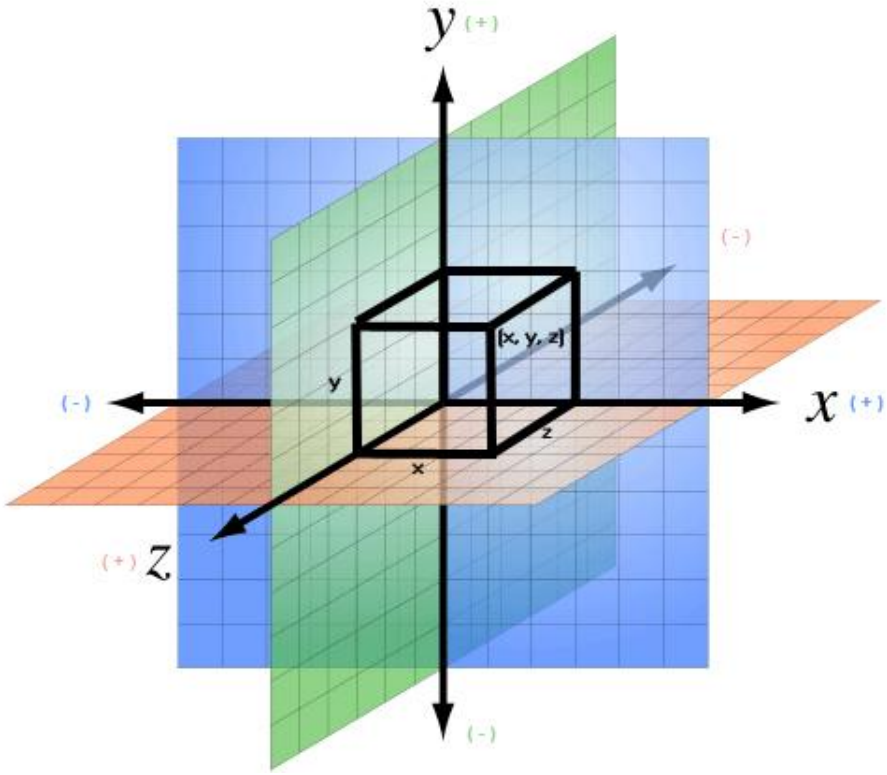

Figure 6 An example of geometric coordinate system

An example of a geometric coordinate system is the Euclidian 3-dimensional space shown in Figure 6 , in which every point is determined by three coordinates, for example, *(x, y, z)*.

3.3.1 Representations of Subscription

By using a geometric location model, three types of spatial expressions can be specified, namely single equation approach, simple predicate approach, and rectangular-based approach. Table 4 shows some examples of the three approaches. The simple predicate approach can specify only rectangular shapes in 2-D or cubic volume in 3-D spaces, while the equation approach enables users to specify any shapes or volumes they want, provided that they are able to define the equations. The rectangular-based approach states the x and y coordinates in the top left corner and the bottom right corner of a rectangle.

Representation	**Example**
Simple predicate	X > 0 && X < 10 && Y > 0 &&Y < 10
Equations	$X^2 + Y^2 < 10$
Multiple equations	$X^2 + Y^2 < 10$ && $X^2 + Y^2 < 10$
Rectangular-based	(0,0, 10, 10)

Table 4 Examples of first-level spatial subscriptions

The elements used in the first level can be summarized in Figure 7. Variables can be denoted by x-, y-, and z- coordinates, operators are denoted by +,-, *, /, >, <, =, and constraints are denoted by the real number set.

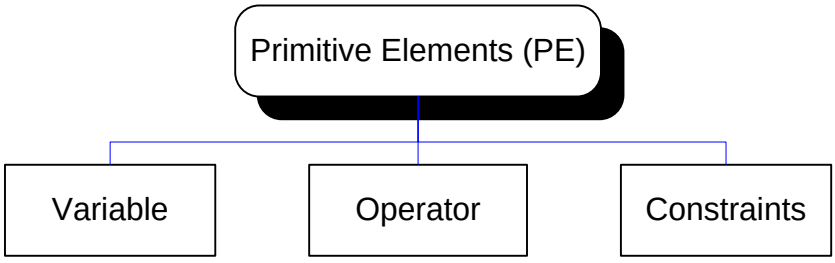

Figure 7 First-level elements

3.3.2 Advantages and Disadvantages

As mentioned before, one of the advantages of the first-level spatial subscriptions is to allows users to specify any point, or any set of points, on a map.

One of the disadvantages of this approach is that it is almost impractical to ask ordinary users, who do not have expertise on specifying locations precisely, to express an equation in order to specify a spatial requirement. Therefore, the systems based on this approach are often normally assisted by a graphic interface input, allowing users to select areas based on a map. However, a problem is that not every mobile device has a touchscreen allowing users to do so. Even if there is a touchscreen, it is very difficult to select the exact area on a coordinate system, and the situation becomes even worse if the coordinate system is 3-dimensional.

3.4 Second level: Fixed Derived Elements (FDEs)

We introduce Fixed Derived Elements (FDEs) in the second level of the model. The objective of introducing this layer is to allow users to specify locations in a more expressive way. There are three major objectives to be achieved as I propose this level. The first objective is to allow users to specify a location by expressing names instead of mathematical formulas. In order to achieve that, every point, area and volume in the level one is mapped to a unique label. In addition, users can also create labels to represent a group of locations, so that a set of non-uniformly distributed areas can be easily be specified. For example, users can adopt a single label *bus stops*, instead of stating multiple points or even multiple equations, to specify all the bus stops registered in the system.

The second objective is to allow users to specify a location by stating its static properties. These static properties include the unchangeable properties of each location defined in the system. An example is to specify the type of locations, such as *building, city or street*.

The last objective is to allow users to specify a location by referring to other locations. To illustrate its applicability, consider a location with a name that is difficult to remember. Users may be occasionally unable to express some locations because they do not know their labels, as they may not be familiar with some of the locations. An intuitive way to solve this problem is to specify a location by specifying its relative position with respect to a more well-known location. For example, *Canada* is a label defining a country's name, yet it can be specified also as *the country north of the USA*. Although there is always more than one such expression to specify a location, normally only a few of them are useful.

3.4.1 Elements

The elements that can be used in the second level of the model are shown in Figure 8. Despite overlaying a symbolic layer over the first layer, users can also specify spatial expressions by using the PEs introduced in the first level.

In the second level, each subscription consists of a least one fixed reference plus one or more optional spatial relations. In the following, I first discuss different types of fixed references and spatial relations. Then, I explain how the references and spatial relations can be combined with each other and form more complex subscriptions.

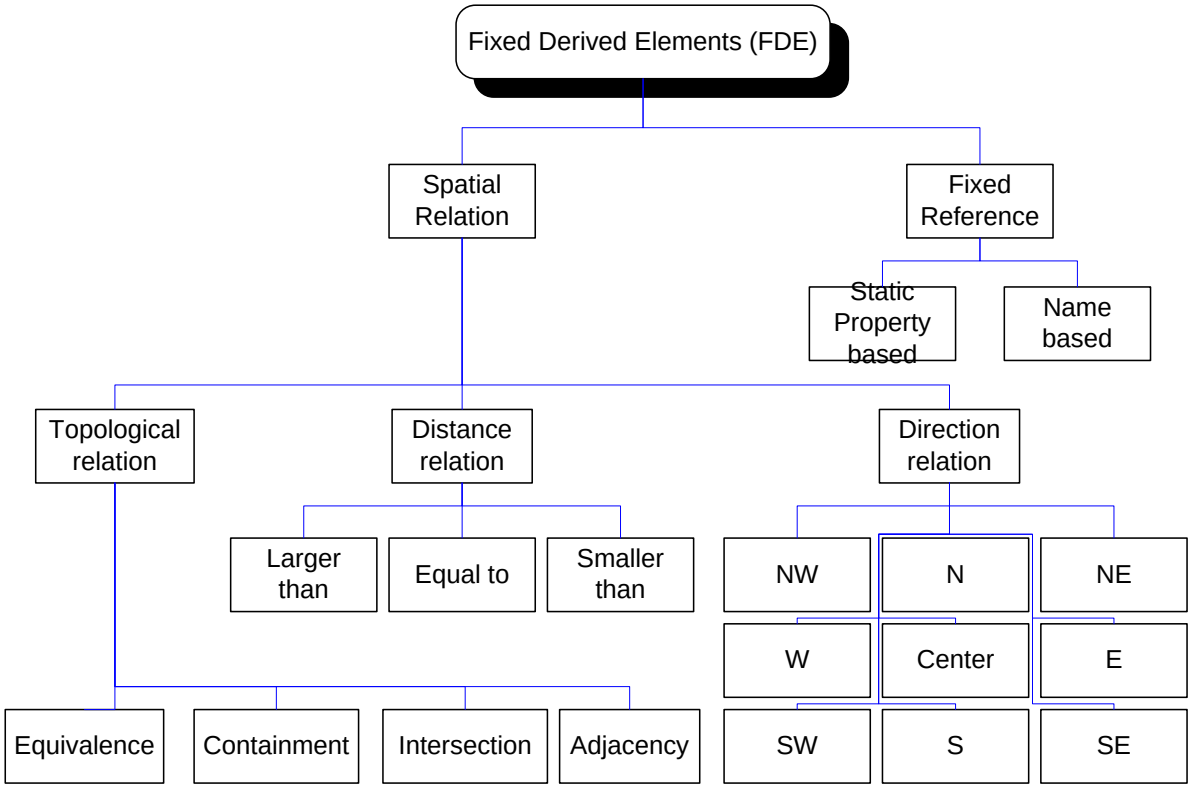

Figure 8 Location elements in second level

3.4.1.1 Fixed Reference

Each fixed reference is mapped to one or more points or equations in layer one. These references can be classified into two types, namely Name-based reference and Static-property reference. Table 5 shows how the locations represented by level one can be represented in level two.

First-level representation	Name-based reference	Static property-based reference
$X > 0$ && $X < 10$ && $Y > 0$ && $Y < 10$	Area A	*type = building, color = blue*
$X^2 + Y^2 < 10$	Area B	*type = Street, numbers_of_lanes = 4*
$X^2 + Y^2 < 10$ && $X^2 + Y^2 < 10$	Area C	*type = City, population = 1M*

Table 5 Examples of second-level spatial subscriptions corresponding to the first-level ones

Name-based reference

Name-based references are labels initially registered in the systems. It is a reference commonly used in daily life to refer to locations, such as Canada, New York, and King Street.

Static property-based reference

Static property-based reference is created by referring to the static properties of the locations. For example, *the types of the locations, and the colors of the buildings*. As a result, the representation of static property-based reference is more complex. Normally, there are more than one location objects with the same set of properties. Hence, a property-based reference usually consists of more than one location objects.

3.4.1.2 Spatial Relations

We have discussed so far all the spatial relations I have introduced in Chapter 2. A detailed description of each spatial relation can be found in Section 2.2.

3.4.2 Representations of Subscriptions

Having described the elements introduced in the second level of the model, I focus on the representation of spatial expressions that use these elements. In general, a new location label can be created by combining fixed references with spatial relations. The representation of a hybrid location reference is denoted as follows:

Location Set + Spatial relation + Reference object

Location Set is a group of location objects which consists of the possible location objects users are looking for. Spatial relation and reference object are used to specify the constraints and refine the set.

In general, Location Set is a static property-based reference and Reference object is a name-based reference, since a static property-based reference usually maps to a group of location objects, while a name-based reference usually maps to a single location object. These expressions rely on basic set theory: a subset can be formed by choosing some of the objects in the large set; a single location cannot be restricted to another subset.

In an expression, the reference object must be specified, while the Location Set and Spatial relation are optional. If Location Set is not specified, it is assumed to be *all the location objects* registered in the system. If spatial relation is not specified, it is assumed that to be the *equivalence* relation between sets of points. So, an expression *Bus Stops* means *all location objects equivalence to Bus Stops*. Table 6 shows examples second-level spatial subscriptions.

Representation	**Example**
Fixed reference [Property] + Equivalence	*(All location objects equivalence to) Bus Stops*
Fixed reference [Property] + Equivalence	*(All location objects equivalence to) Houses in red color*
Fixed reference [Name] + Containment	*(All location objects) Within Toronto*
Fixed reference [Property + Name] + Containment	*Cities within Canada*
Fixed reference [Property + Name] + Intersection	*Streets intersect with "ABC Street"*
Fixed reference [Property + Name] + Adjacency	*Countries next to the US*
Fixed reference [Property + Name] + Distance	*Cities near Toronto;*
Fixed reference [Property + Name] + Direction	*Countries south to the US*
Fixed reference [Property + Name] + Distance + Containment	*Area within 100km of Toronto*

Table 6 Representations and examples of second-level spatial subscriptions

3.4.3 Advantages and Disadvantages

Although the second layer enhances the expressiveness of the abstraction used to specify spatial subscription, much effort has to be made in order to define all the location labels and the properties of locations. Existing systems tend to omit the first level and directly deploy the second level. These systems often also allow users to specify symbolic location names in subscriptions. However, without the first level, subscriptions and events are matched by performing string pattern matching based on location expressions instead of evaluating the actual areas defined by the expressions. The drawback is these systems have all the possible relations between areas hard-coded, which appears to be impractical as the number of spatial relations between location objects increases factorially with the number of location objects.

Another advantage of overlaying the second level on top of the first one is that users can also specify the first-level abstractions in spatial subscriptions, in case there are locations which have not yet been given a name while the systems are executing at runtime. So, spatial expressions are not restricted to those registered in the system.

3.5 Third level: Dynamic Derived Elements (DDEs)

We aim at supporting mobile spatial subscription in this thesis. In mobile DEBSs, object positions and states change over time. Therefore, users should be able to take advantage of these dynamic properties to specify locations. For instance, *raining cities* should be referred as different cities over time. However, the elements provided by the second level allow users to specify only static locations on a map. The third level defines the elements to specify dynamic locations, which means location labels can refer to different points on a map in different time frames.

One of the advantages of deploying the third level is to minimize the numbers of subscription and unsubscription processes. Consider a case in which I would need to subscribe to the cities where it is currently raining. Without the DDEs, users have to issue two subscriptions. A first subscription would be used to subscribe for raining events and a second subscription to subscribe to cities according to the information of the first subscription. As a result, the second subscription has to be re-subscribed to a new location set as the set of raining cities changes according to the first subscription. In addition, the re-subscription approach would lead to a possible failure to receive events, in case events are issued from those locations during the re-subscription period.

3.5.1 Elements

There are two types of dynamic references in this level. Location expressions are represented by combining elements in the Third level with elements in the second level, FDEs. Figure 9 shows the elements in the third level defined.

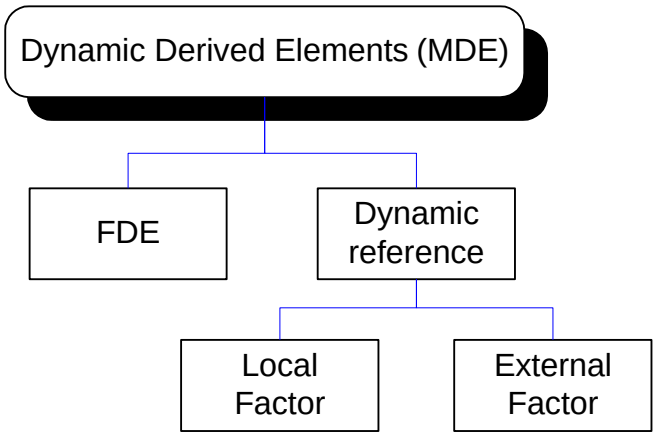

Figure 9 Third-level location abstractions

Local factor

Every location object has a set of properties. Some properties are changeable while some are not. Practically, properties such as weather changes over time while the color or even the type of the location object usually does not change. However, the set of properties that are changeable is chosen by the system developers. For instance, a system might be designed to have static weather for each location but a variable type of locations, such as *a land* can become *a house*. So, the changeable properties of locations are ultimately designed by the system developers.

In terms of syntax, the Local dynamic factor is similar to the static property-based reference discussed in the second level. The major difference is that the label of local factor can be mapped to different locations in different time frames.

External factor

The external factors of a location object are affected by other dynamic objects. For example, consider a group of students, and a user who wants to specify his/her interests in areas around the students. By using the point of view of the students, the location expression can be specified as *areas within the 100m of each student*; by using the point of view of the areas of interest, the location expression can be specified as *areas where there is a student within 100m*.

3.5.2 Representations of Subscription

Each third-level location expression consists of at least a Fixed reference and a Dynamic reference, and a optional spatial relation. If a spatial relation is not included in a subscription, *equivalence* is chosen as a default relation. The representation of a third-level location expression can be specified as follows:

Fixed reference + Dynamic reference + Spatial relation

Table 7 shows some examples of the third-level subscriptions.

Representation	**Example**
Local factor [Weather] + Fixed reference	*Cities where it is raining* Fixed reference + Local Factor
External factor [Object entered] + Fixed reference + Equivalent	*Classrooms where there are students* Fixed reference + External Factor + Equivalent
External factor [Object moving] + Fixed reference + Containment + Direction + Distance	*Area within 100m ahead of John* Fixed reference + Containment + Local Factor + Distance + Direction
External factor [Object moving] + Fixed reference + Distance	*Areas which are within 100m of each student* *Areas where there is a student within 100m* Fixed reference + External Factor + Distance
External factor [(Multiple) Object Moving] + Fixed reference + Equivalent	*Classrooms which both teachers and students are in* Fixed reference + External Factor

Table 7 Representations and examples of the third-level spatial subscriptions

3.6 Hybrid Expressions

Different levels of expression can be combined to form a new location expression. In the following sections, I focus on two types of combinations.

3.6.1 Multi-level Expressions

The first-level expressions can be combined with the second and the third level to form location expressions. Normally, the Name-based references in the second-level expressions can be replaced by a level one expression. Table 8 shows an example specifying all cities within certain areas.

Representation	**Example**
Fixed reference [Property] + Containment + First-level abstractions	*Cities within (0, 0, 100, 100)*

Table 8 Representation of hybrid spatial subscriptions

3.6.2 Cascading Expressions

Location expressions can be joined to form a cascading expression. The Fixed reference is a second-level location expression can be replaced by another second level or third-level expression. Table 9 shows some examples.

Representation	**Example**
Fixed reference [Property] + Containment + Fixed reference [Property + Name] + Adjacency	*Cities in Countries next to the US*
Fixed reference [Property] + Containment + Fixed reference [Property] + Local Factor	*Bus Stops in Cities where it is raining*
Fixed reference [Property] + Distance + Fixed reference + External factor [Object entered] + Equivalent	*Bus Stops within 100m of School where there are students in*

Table 9 Representations and examples of cascaded spatial subscriptions

3.7 Summary

In this chapter, I have presented a three-level mobile spatial subscription model. Examples are used to illustrate the spatial subscriptions in each level, as well as how expressions in different levels can be combined.

The first-level is designed to handle spatial subscriptions consisting of geometric coordinates. Table 6 shows that the three-level model supports all the essential location-based abstractions

proposed in Chapter 2, while only the *equivalence* relation is supported in existing string-based DEBSs.

The second level allows subscribers to specify locations using labels and spatial relations instead of mathematical formulas. For example, users can now specify a location by referring to another more well-known location, while existing DEBSs only allow users to specify a location by its pre-defined name in the system.

The third level extends the second level and allow subscribers can to specify locations by stating their dynamic properties. Users can subscribe to different locations over time, even if they do not modify their issued subscriptions. Compared with using static subscriptions in existing DEBSs, this new type of subscription avoids re-submission of subscriptions and reduces overhead [10, 48], especially when the DEBS approach is deployed in a mobile dynamic environment.

Chapter 4

A Plug-in Approach for Subscription Conversion

4.1 Overview

In this chapter, I present a plug-in implementation approach to extend existing DEBSs to support mobile spatial subscriptions. The goal is to minimize the changes required to deploy spatial subscription support to existing systems. In the literature [18, 28, 39, 42, 43], major modifications often need to be made to allow event brokers to handle spatial subscriptions. The required changes involve system-specific modifications, which do not constitute as a whole decoupled and re-usable solution.

As part of the plug-in approach, I introduce a Mobile Subscription Conversion Unit (MSCU) to handle all spatial subscriptions. This is a separate component and can plug into most of the existing DEBSs without many modifications. Hence, subscribers are allowed to issue new types of subscriptions, based on the model presented in the last chapter, without the modifying the existing middleware.

In the following sections, I first describe the procedures to integrate MSCU into existing DEBSs and the interaction between the MSCU and other components. Then, each sub-component of MSCU is defined and discussed. At last, I discuss some of the problems introduced by this plug-in approach.

4.2 The Architectural View of DEBSs

The major objective of this plug-in approach is to minimize the modifications required to enable existing DEBSs to support spatial subscriptions. Traditionally, subscribers issue subscriptions to the middleware. Therefore, to support spatial subscriptions, modifications have to be made to the middleware, so that spatial subscriptions can be recognized.

To avoid modifying the middleware of DEBS, I propose to handle all the spatial subscriptions by introducing another Publish/Subscribe component. The new component is named Mobile Subscription Conversion Unit (MSCU) and is not hard-coded into the middleware. Figure 10 shows an architectural view of a DEBS with a MSCU. The arrows with dotted line borders represent the new

data flow of spatial events/subscriptions; the arrows with solid line borders represent the original data flow of events.

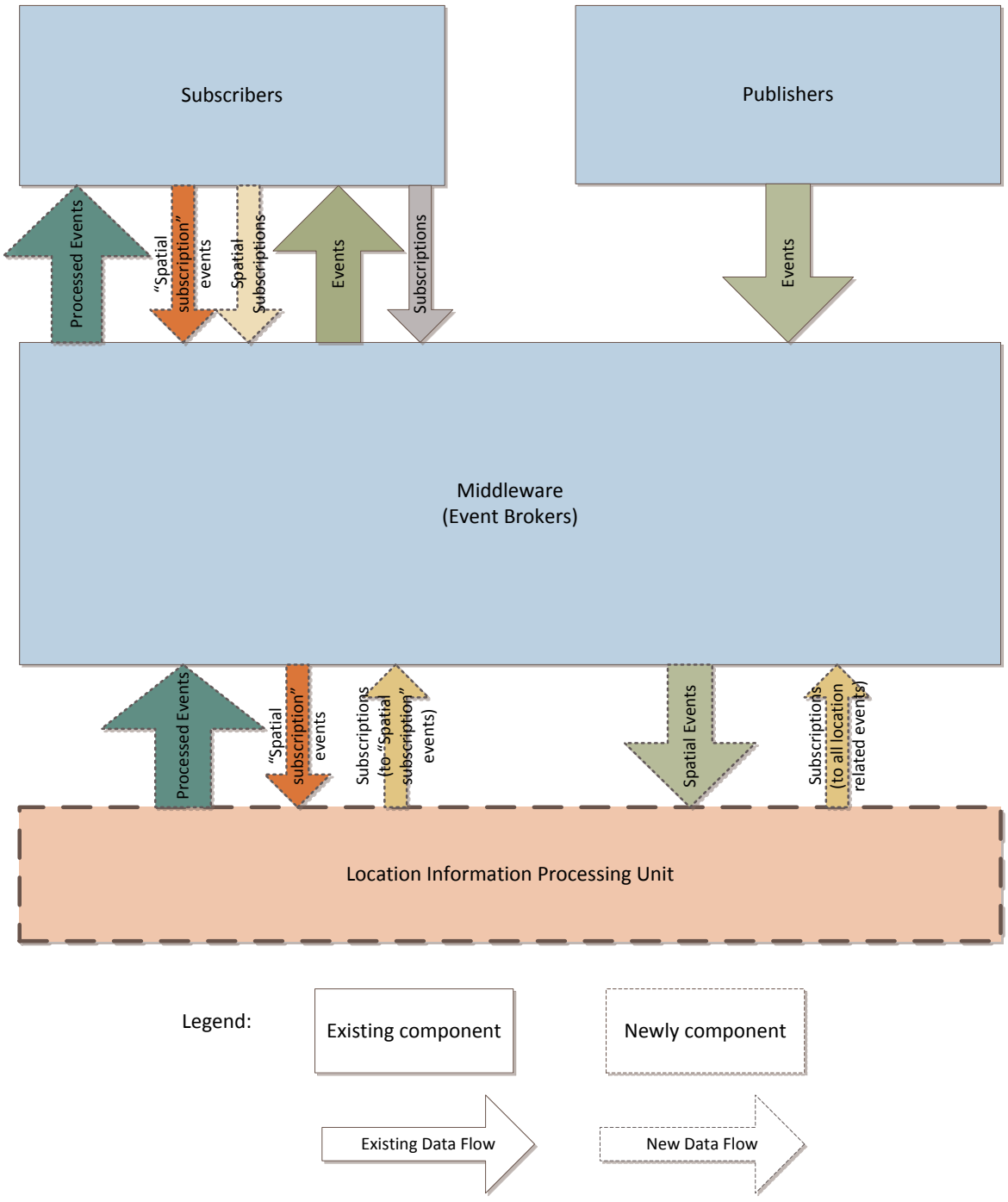

Figure 10 Architectural view of a DEBS with the MSCU

Traditionally, subscribers issue subscriptions to the middleware. When publishers publish events to the middleware, the middleware forwards the events to the corresponding subscribers. However, in the proposed plug-in approach, the basic idea is that the MSCU subscribes to all the location-related events and the subscribers issue spatial subscriptions to the MSCU. When publishers publish location-related events, all these events are forwarded to the MSCU. The MSCU then forwards the events to the corresponding subscribers.

Under the DEBS paradigm, subscribers are only allowed to issue subscriptions to the middleware, but not to other publishers. Therefore, subscribers are not allowed to subscribe directly to the MSCU. In order to subscribe to the MSCU, I introduce event-like subscriptions in the plug-in approach. On the subscriber side, a spatial subscription is encapsulated in an event before it is forwarded to the middleware. On the MSCU side, it subscribes to the event-like subscription event. Hence, subscribers are able to issue subscriptions virtually to the MSCU.

4.3 Data Flow

In practice, all the communication between subscribers and the MSCU are via the middleware. As a result, the data flow between them is more complex. Figure 11 and Figure 12 are examples illustrating the complete data transmission processes between a subscriber and a publisher. The example assumes that the event issued by the publisher satisfies the spatial subscription issued by the subscriber.

The process consists of two phases, an initial phase and a processing phase. During the initial phase, the subscriber indicates its spatial events of interests. At the beginning, the MSCU issues two types of subscriptions to the middleware. The first type is to subscribe all the location-related events, so that all location events issued by publishers will be forwarded as a copy to the MSCU. The second type is to subscribe the all event-like spatial subscriptions. Then, the subscriber issues event-like spatial subscriptions, with a location expression, to the event brokers which are forwarded to the MSCU. The subscriber also has to issue spatial subscriptions, with the same location expression as the event-like spatial subscriptions, to the middleware.

During the processing phase, the middleware keeps waiting for incoming published events. When a location-related event arrives, it is forwarded to the MSCU. The MSCU processes the event and checks whether the event satisfies with any subscription received earlier. For each subscription that matches the event, a copy of the event, with the location expression of the subscription, is published

to the event brokers. The event brokers then forward the event to the corresponding subscriber that has subscribed the same location expression.

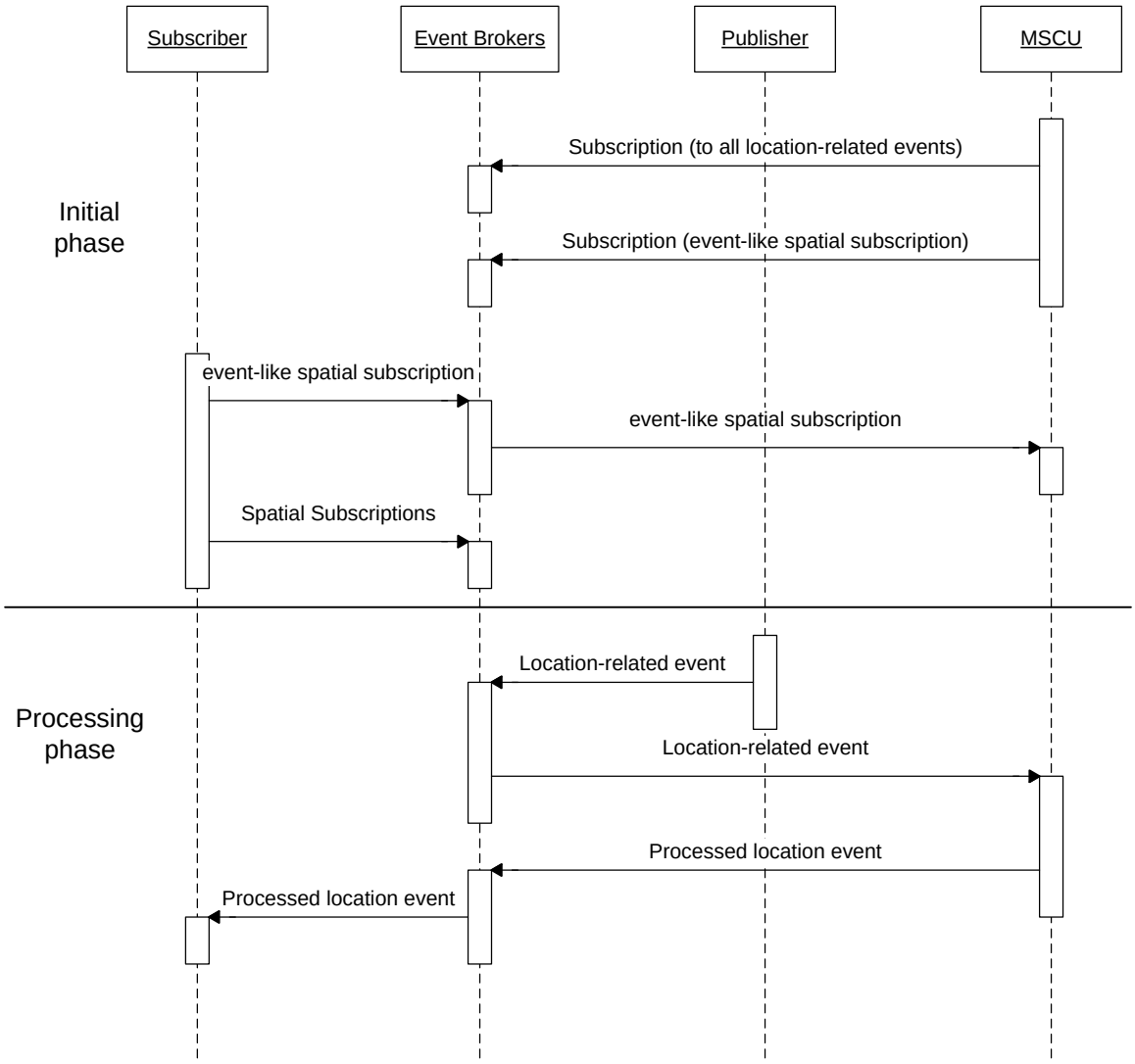

Figure 11 A sequence diagram illustrating data transmission in a DEBS with a MSCU

	Sender	Receiver	Data	Description
1.	MSCU	Event Brokers	SS	Subscribe to all location-related events.
2.	MSCU	Event Brokers	SES	Subscribe to all event-like spatial subscriptions.
3.	Subscriber	Event Brokers	ESS	Publish event-like spatial subscriptions contain a location expression.
4.	Event Brokers	MSCU	ESS	The event brokers dispatch the event-like spatial subscriptions.
5.	Subscriber	Event Brokers	SPE	Subscribe to events with the same location expression.
6.	Publisher	Event brokers	LE	Publisher publishes a location-related event.
7.	Event brokers	MSCU	LE	The event brokers dispatch the location-related event.
8.	MSCU	Event brokers	PLE	After conversion process, the MSCU publish the processed event.
9.	Event brokers	Subscriber	PLE	The event brokers dispatch the processed event.

SS = Spatial Subscription; SES = Subscription to Event-like Spatial Subscription;

ESS = Event-like Spatial Subscription; SPE = Subscription to Processed Location Event;

LE = Location-related Event; PLE = Processed Location Event

Figure 12 Descriptions of the sequence diagram in Figure 11

4.4 Subscription and Event Data

In this section, I focus on the subscription and event data. The data varies according to different types of DEBSs. Table 10 shows how the data can be modeled in different types of DEBSs.

	Subject-based DEBSs	**Content-based DEBSs / Type-based DEBSs**
Location-related Event (LE)	Events disseminated by channels with location labels.	Events with the name-value pair *Location*
Spatial Subscription (SS)	To issue a spatial subscription, subscribe to all the channels with location names.	Subscribe to all events with the name-value pair *Location*.
Subscription to Event-like Spatial Subscription (SES)	Subscribe to a virtually created channel.	Subscribe to events with the name-value pair *Location_Expression*
Event-like Spatial Subscription (ESS)	Issue events to a virtually created channel.	Events with two new name-value pairs: Location_Expression and subscriberID
Subscription to Processed Location Event (SPE)	Subscribe to a channel named subscriber's ID.	Subscribe to the events with Subscriber's ID and Location name-value pairs.
Processed Location Event (PLE)	Issue event to a channel named subscriber's ID.	Convert matched event to a new one with Subscriber's ID.

Table 10 The event data of different types of DEBS

4.5 Internal components of the MSCU

The internal components of the MSCU will be discussed in this section. Figure 13 shows the architectural view of the internal components of the MSCU. To simplify the internal view of MSCU, only the data flows of events is shown and the original subscriptions issued by MSCU to subscribe to PLE and SS are not shown in the figure. The MSCU can be divided into three parts based on their functions. The Spatial Subscription Manager is responsible for managing the received event-like subscriptions. The Dynamic Property Manager is responsible for monitoring and managing the dynamic properties of each location object. The Spatial Event Manager is responsible for converting

the incoming location events into events that can be recognized by the event brokers. In the following subsections, I illustrate how each component manages the received events.

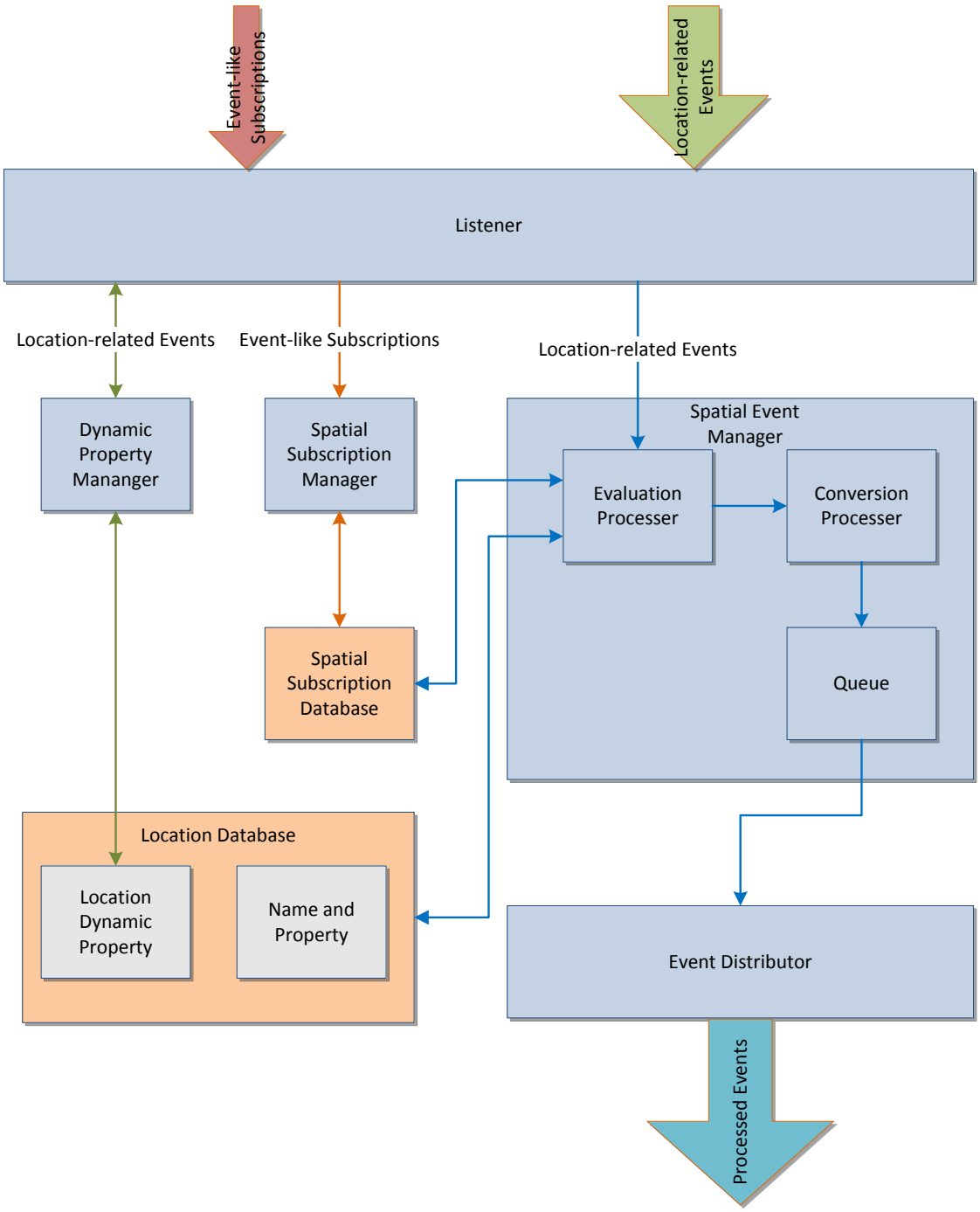

Figure 13 Architectural view of the internal components of MSCU

4.5.1 Listener

After receiving events from the event brokers, the Listener forwards the Event-like Subscriptions (ESS) to the Spatial Subscription Manager, and forwards the Location-related Events (LE) to both the Dynamic Property Manager and the Spatial Event Manager.

4.5.2 The Spatial Subscription Manager

The spatial Subscription Manager (SSM) stores the incoming three-level mobile spatial subscription, which is introduced in Chapter 3, into the Spatial Subscription Database. The SSM extracts two pieces of information from the three-level subscription: a unique subscriber ID and a location expression. Then, this information is compared with those in the Spatial Subscription Database to ensure it is not duplicated data. At last, the information is added to the database as a new spatial subscription.

4.5.3 The Dynamic Property Manager

Dynamic Property Manager (DPM) monitors the latest states of locations. It helps to keep an updated copy of the dynamic properties of location in the Location Database. When the Listener receives a location-relation event, it first forwards the event to the DPM. The DPM examines the incoming event to check if the event causes any location state changes. Then, the affected locations, which are in the Location Database, are updated. Finally, the DPM notifies the Listener to continue its tasks.

4.5.4 The Spatial Event Manager

After the DPM updates the location state, the Listener forwards the same location-related event to the Spatial Event Manager (SEM). The DPM has to be executed before the SEM, so that the SEM can make use of the most updated location states to perform the subscription-event matching evaluation.

The responsibilities of the SEM are to match the incoming event with each spatial subscription in the database, and publish processed events to the corresponding subscribers. There are three stages in this process and are handled by three different components, namely Evaluation Processor, Conversion Processor and Queue.

Evaluation Processor

During the evaluation process, a location expression is extracted from the received spatial event. Then, the high level expression is converted into a first-level one, which is in equation or coordinate form.

The conversion process is based on the model I presented in Chapter 3. The first-level expression is then evaluated with each subscription in the spatial subscription database.

Similarly, location expressions are also extracted from the stored spatial subscriptions. Either a first-level, second-level and third-level expression can be extracted from each subscription. The second-level and the third-level expressions are then converted into a first-level one. Consequently, the matching processes between the subscriptions and events are all based on the first-level expressions, which evaluate the actual physical areas specified by these expressions. All the subscriptions that match the event are passed to the conversion stage.

Conversion Processor

For each subscription that matches the received event, a cloned copy of the event is generated with the subscriber's ID. I skip the new event generation process here, as it has been discussed in Section 4.4. The cloned copy is then queued up for publishing.

Queue

The processed event must be queued up instead of being publishing directly, to prevent the conversion process to be blocked if the connection to the event brokers fails temporarily due to unstable network connections. Eventually, the processed events are forwarded to the Event Distributor, where they wait for further dissemination.

4.5.5 Event Distributor

Once the events are ready, the Event Distributor sends out all the processed events to the event brokers.

4.6 Limitations

Although the plug-in approach minimizes the modification needed for existing DEBSs to support mobile spatial subscription, some problems are introduced into the systems.

4.6.1 Overhead

The introduction of MSCU generates overhead into the existing system. In the initial phase, four more data transmissions are needed in order to issue a spatial subscription. In the processing phase, two more data transmissions are needed to forward the related events to the corresponding subscribers. A

solution is to allocate both the MSCU and the middleware to the same server or the same set of servers, so that the data transmissions between them can be treated as internal data transfers.

4.6.2 Privacy Issue

When a subscriber issues an event-like subscription, its subscriber's ID is embedded in the event. Consequently, their ID is revealed, which violates the proposed design of DEBS. A possible solution to this problem is to use a random generated number in the event, instead of their real ID, so that identities can be concealed and the event matching process in the MSCU is not affected.

4.7 Summary

In this chapter, I have presented a plug-in implementation approach to support mobile spatial subscriptions. In the literature, to support spatial subscriptions, either a band new system is developed [2, 3, 5, 13, 18, 28, 34, 39, 43, 49-51] or major changes such as modifying the middleware are required [31].

Instead of modifying the existing DEBS, the proposed approach implements an extra component to support the spatial subscriptions, and this component can be easily plugged into the existing DEBSs. Although some overhead is introduced to the system, the component can be re-used in different DEBSs with minimum modification.

Chapter 5

A Case Study:

Dynamic Mobile Subscription System (DMSS)

In this chapter, I illustrate the usefulness and applicability of the proposed three-level mobile spatial subscription model and the plug-in approach by developing a prototype system, Dynamic Mobile Subscription System (DMSS). The system is built based on GEM [1, 52], an existing DEBS supports generic subscriptions. In the following sections, I first introduce some background information about GEM, and then, I describe how the prototype can support mobile spatial subscriptions by adding a plug-in component into GEM.

5.1 Introduction to GEM

GEM [1, 52] serves as a generic event-based framework which executes on a P2P network. After being deployed in participating hosts, GEM forms an overlay network for DEBSs over Sun Microsystem's JXTA [53]. A GEM host consists of a framework core service, administrative components, and any number of application components. Figure 14 shows the architectural view of GEM. I give a brief introduction to these components in the following sections.

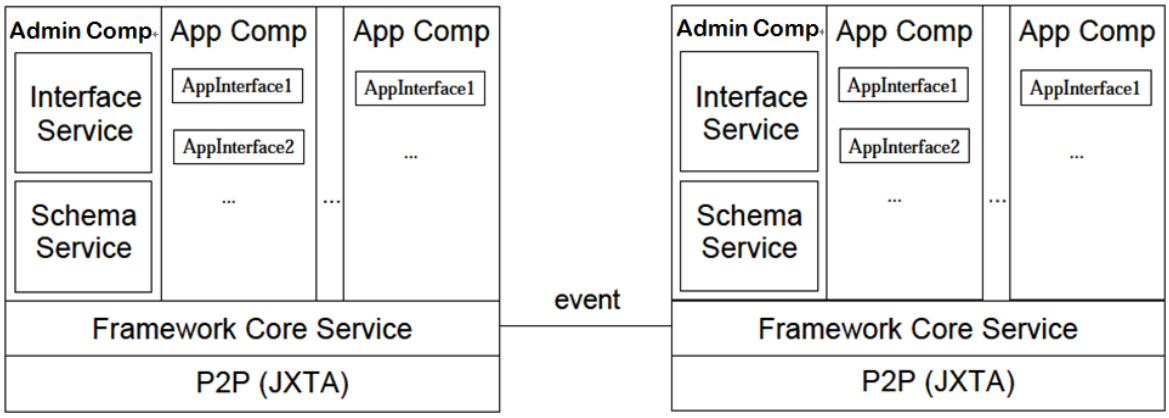

Figure 14 Architectural view of GEM [1]

5.1.1 Framework Core Service

The framework core service is the only system part that interacts with the underlying JXTA network and provides the essential framework services to the other parts of the framework. The main services provided by the framework core service includes, but is not limited to event handling (e.g., dispatching, receiving and forwarding to its target interfaces) and component handling (e.g., start and stop components).

5.1.2 Administrative Components

Administrative components provide management functionality, including two built-in services SchemaService and InterfaceService.

SchemaService

SchemaService allows the registration and de-registration of event schemas, which define the data organization of incoming and outgoing events. In GEM, an event schema must be registered before its events can be created and used. After an event schema has been removed, all its ongoing events will be treated as illegal and not recognized until this event schema is registered again.

InterfaceService

InterfaceService allows the registration and de-registration of component interfaces, which control the behavior of the particular GEM host. The behavior is defined by a finite state machine (FSM), specifying the transitions of states for different incoming events, from the start state to the end state.

5.1.3 Application Components

The action taken in each state of the FSM is further implement by an AppInterface, which is programmed in Java. In each GEM host, there is one and only one component interface, but there are one or more event schemas and AppInterface.

The implementation details of GEM are not discussed in this thesis. A more detailed discussion about GEM can be found in [1]. In the reminder of this chapter, I will illustrate the development process of the prototype system by discussing the FSM and its implementation in each of the hosts.

5.2 Prototype system: Dynamic Mobile Subscription System (DMSS)

In this section, the prototype system, Dynamic Mobile Subscription System (DMSS), is discussed. The purpose of DMSS is to show how the proposed spatial model presented in Chapter 3 can be

implemented based on the plug-in approach introduced in Chapter 4. The prototype system is implemented as a plug-in component to GEM. Although only the most critical functions in each category are implemented, it is enough to show that the three-level model and approach can be deployed into different existing DEBSs. As an indication of the reversible nature of DMSS, the location expressions of events generated by the publishers are in their original forms, and the hence, the middleware does need not to be modified. I add extra events and subscriptions to facilitate the information transmission processes.

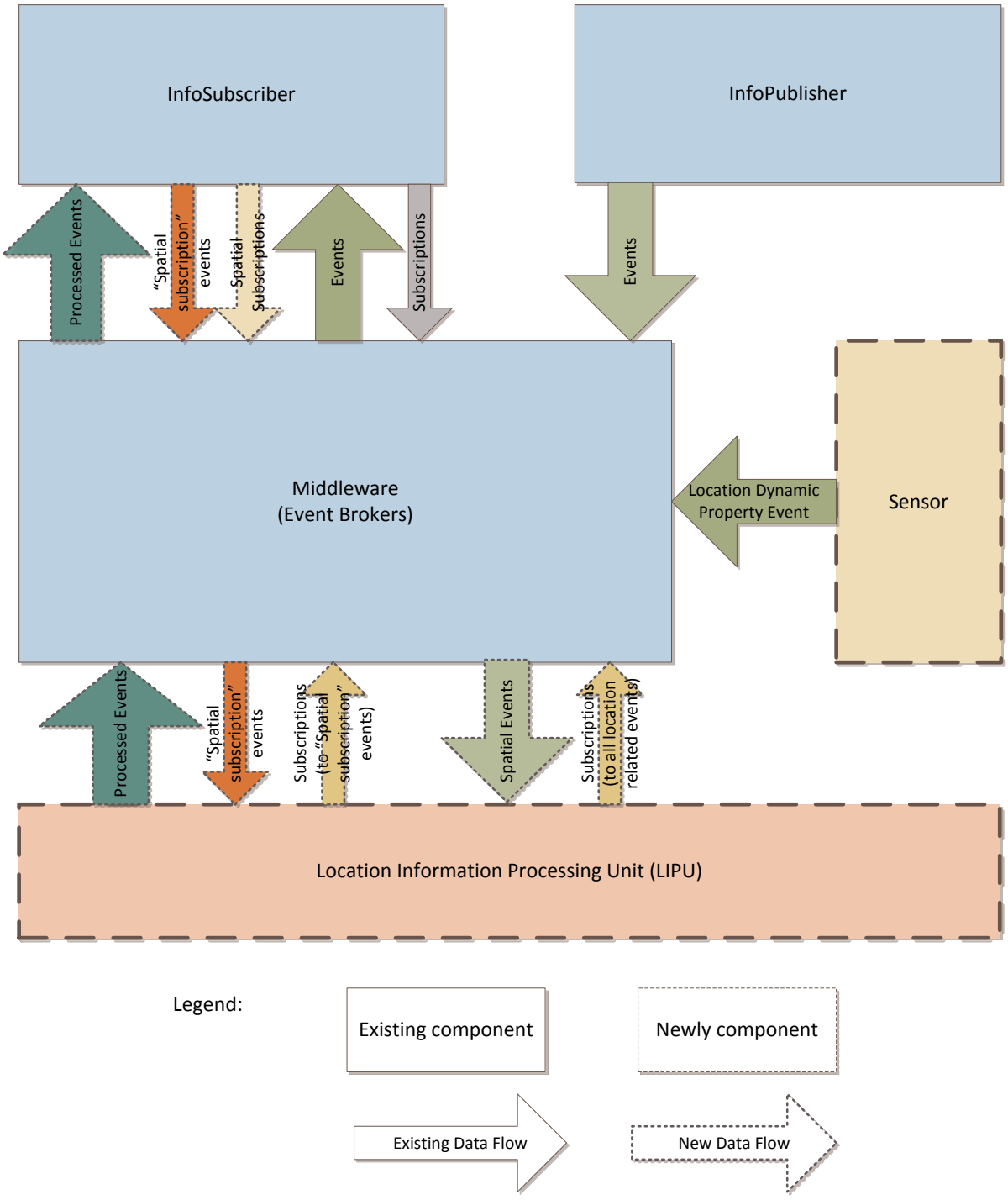

Figure 15 Architectural view of DMSS

The prototype system consists of a MSCU, and three other hosts for the purpose of demonstration. Figure 15 shows the interactions between these components. Basically, the prototype system follows the plug-in approach I presented in Chapter 4, except a Sensor is added to the system as well. The sensor is used to generate dynamic location property changes, so as to verify the third level of the proposed model fully.

5.2.1 Scenario

Without loss of generality, I use the prototype system to demonstrate how spatial events can be disseminated over a relatively simple map, which is shown in Figure 16. Six location objects, LocA to LocF, are created on a 2D geometric coordinate system. Subscribers can specify their locations of interest on these six location objects by using the first-level, second-level, or third-level spatial subscriptions introduced in Chapter 4.

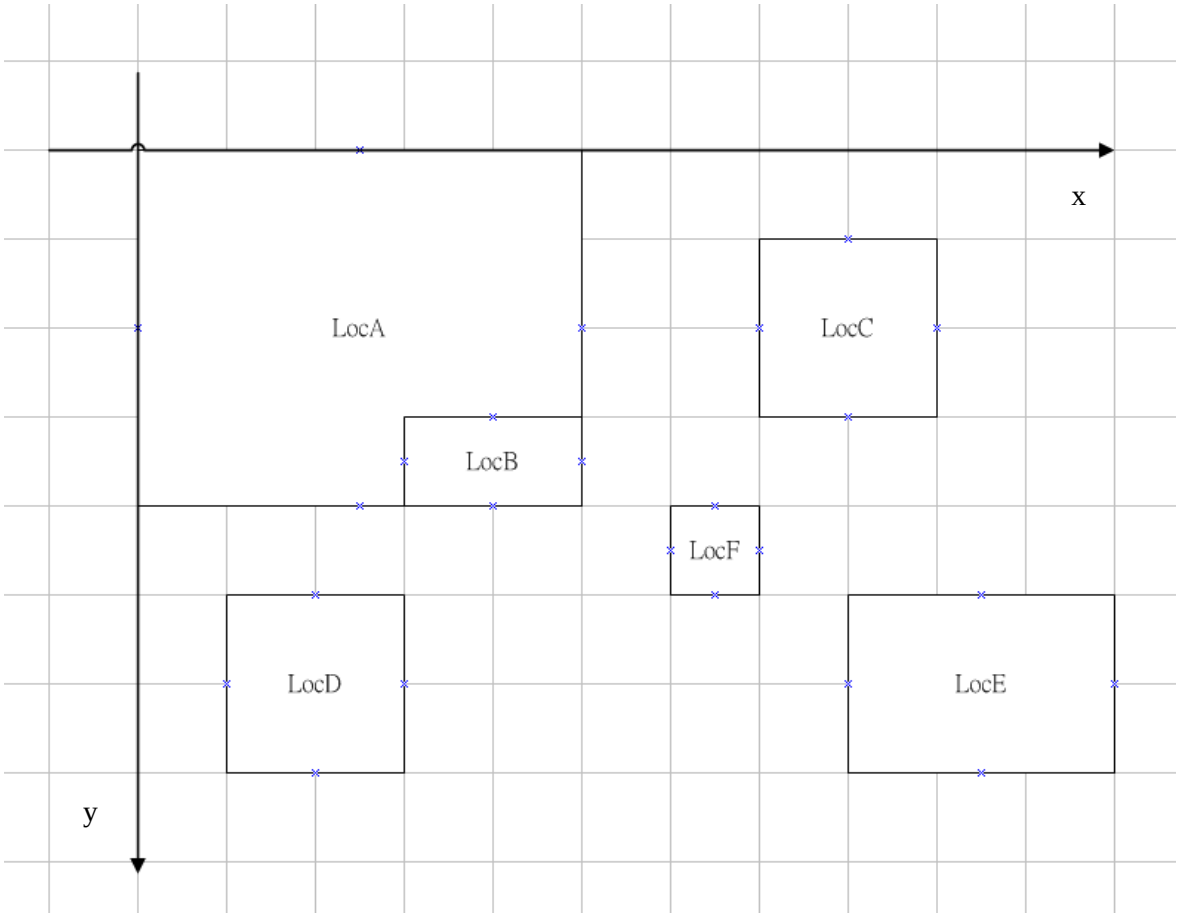

Figure 16 A simple map used to demonstrate the prototype system

5.2.2 Event Schemas

Before introducing the components of the prototype, I present the event and subscription data in this section.

Location-related Event (LE)

A weather event, gemdemo.sensor.WeatherData, is implemented. The event comprises two types of information, weather and location. The Location information is encapsulated by first-level expressions, as publishers are not necessarily to be modified to support higher-level location expressions.

Figure 17 shows the schema of the Location-related event. The Weather information is specified by using a string to maximize the flexibility of the descriptions. The reason is to enable publishers to describe unexpected weather in detail. Although only *Weather* and *Location* are defined as part of the event data, other types of contextual information can also be modeled.

```
<Attr Name="location"        Type="STRING"/>
<Attr Name="weather"         Type="STRING"/>
```

Figure 17 Schema of a Location-related event (gemdemo.sensor.WeatherData)

Event-like Spatial Subscription (ESS)

An ESS is a subscription modeled by using an event. Figure 18 shows the schema of the ESS. The schema is designed to handle different types of spatial subscriptions, including the first level, the second level, and the third-level subscriptions.

```
<ComplexType Name="Area">
        <Attr Name="x1" Type="FLOAT"/>
        <Attr Name="y1"  Type="FLOAT"/>
        <Attr Name="x2" Type="FLOAT"/>
        <Attr Name="y2"  Type="FLOAT"/>
</ComplexType>
<ComplexType Name="Contexts">
        <Attr Name="weather" Type="STRING" minOccur="0" maxOccur="1"/>
        <Attr Name="no_of_people"  Type="FLOAT" minOccur="0" maxOccur="1"/>
</ComplexType>
<ComplexType Name="gemdemo.sensor.SpatialSubData">
```

```
        <Attr Name="spatialtype"        Type="STRING" minOccur="1" maxOccur="1"/>
        <Attr Name="locationarea"       Type="Area" minOccur="0" maxOccur="1"/>
        <Attr Name="locationname"       Type="STRING" minOccur="0" maxOccur="1"/>
        <Attr Name="locationform"       Type="STRING" minOccur="0" maxOccur="1"/>
        <Attr Name="spatialrelation"    Type="STRING" minOccur="0" maxOccur="1"/>
        <Attr Name="othercontexts"      Type="Contexts" minOccur="0" maxOccur="1"/>
</ComplexType>
```

Figure 18 Schema of the Event-like Spatial Subscription

Processed Location Event (PLE)

The content of a proceed PLE is the same as the original Location-related event, except the location representation is converted from a first-level expression to higher-level expression, which matches the corresponding subscription.

Location Update Event (LUE)

A LUE is used to specify the updated state of a location. In the DMSS system, locations have only two contexts, namely weather and number of people. Figure 19 shows the schema of the LUE. A location is specified by the attributes x1, y1, x2, y2, and its contexts are *weather* and *no_of_people*.

```
<ComplexType Name="Area">
        <Attr Name="x1" Type="FLOAT"/>
        <Attr Name="y1"  Type="FLOAT"/>
        <Attr Name="x2" Type="FLOAT"/>
        <Attr Name="y2"  Type="FLOAT"/>
</ComplexType>
<ComplexType Name="Contexts">
        <Attr Name="weather" Type="STRING" minOccur="0" maxOccur="1"/>
        <Attr Name="no_of_people"  Type="FLOAT" minOccur="0" maxOccur="1"/>
</ComplexType>
```

Figure 19 Schema of Location Update Event (LUE)

5.2.3 Interactions

There are four different types of components in the prototype system. In the following sections, the behavior of each component will be discussed.

5.2.3.1 MSCU

The MSCU starts with Load state. In the Load state, the MSCU issues subscriptions to the event brokers to subscribe to two types of events. In the prototype system, a subscription is used to subscribe Weather events, to be issued by the Sensor; the other subscription is used to subscribe to "Spatial Subscription" events, to be issued by the InfoSubscriber.

After subscribing to the two types of events, the MSCU advances to the Started state. The Started state is an idle state which allows the MSCU to wait for incoming events. When an Event-like Subscription arrives, the MSCU changes to the Received Subscription state, extracting a weather subscription from the event and storing it. When a Weather event arrives, the MSCU enters the Matching state to evaluate each stored subscription. Afterwards, the MSCU generates an event for each subscription that matches the incoming event. The MSCU then enters the Publish state and sends out these generated events one by one.

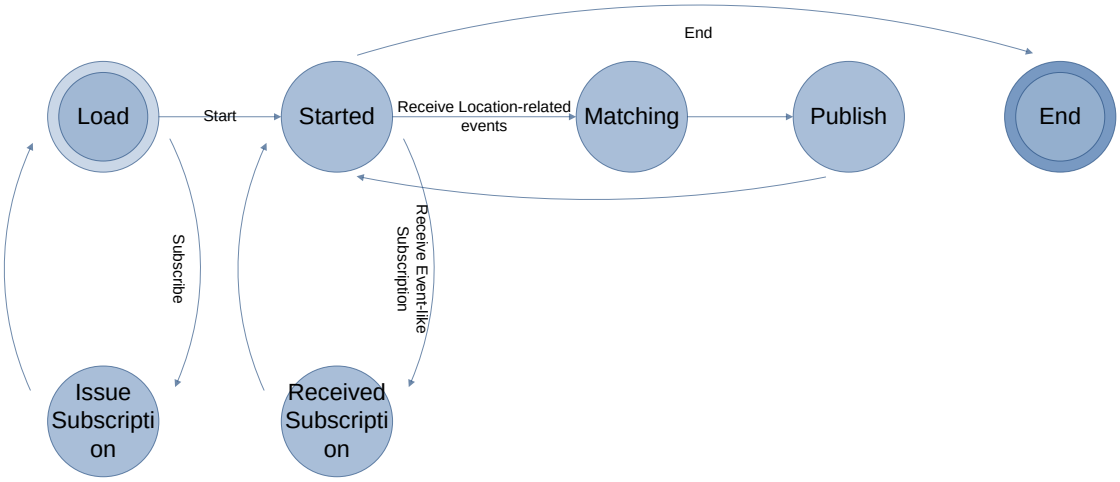

Figure 20 State diagram of MSCU

5.2.3.2 InfoPublisher

The InfoPublsiher sends out Weather event when it is triggered by users. Figure 21 shows the FSM of the InfoPublisher.

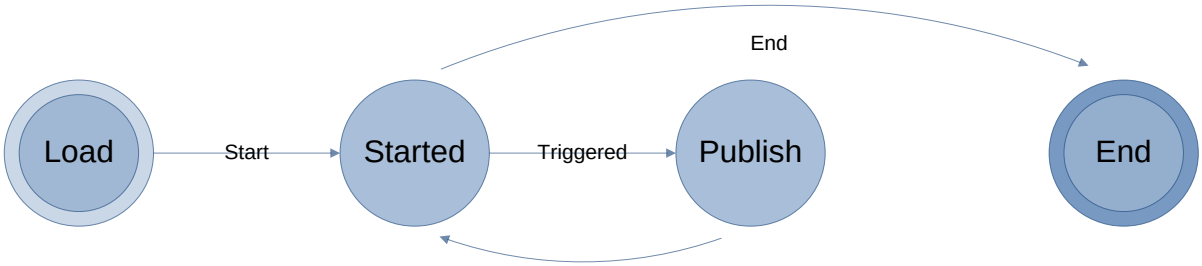

Figure 21 State diagram of InfoPublisher

5.2.3.3 InfoSubscriber

After initialization in the Load state, the InfoSubscriber enters the Started state. In this case study, a subscription is issued only when users trigger this component.

The InfoSubscriber enters the Received Event state when a processed Weather event is received. Conversely, InfoSubscriber would not be able to receive any original, unprocessed, Weather events. Figure 22 shows the FSM of the InfoSubscriber.

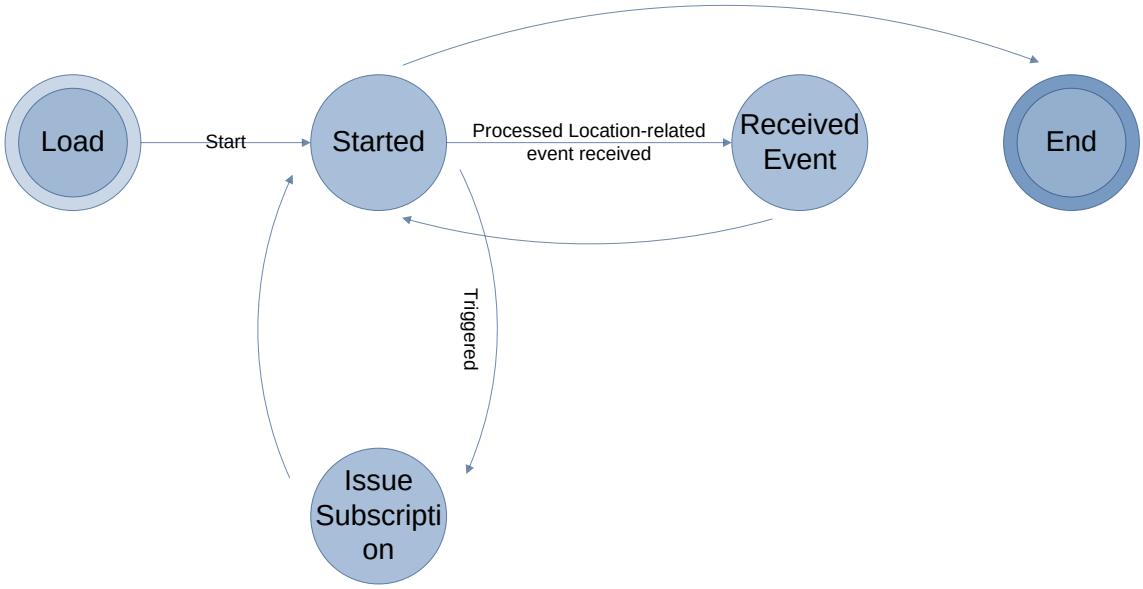

Figure 22 State diagram of InfoSubscriber

5.2.3.4 LocationMonitor

Location Monitor is introduced in the prototype system to generate events representing change of dynamic location properties. When users triggered the component, a Location Update Event (LUE) is generated and eventually forwarded to the MSCU. In this system, there are two dynamic properties for each location, weather and no_of_people. Hence, a LUE specifies either the weather or the number of people that change in a location. Figure 23 shows the FSM of the Location Monitor.

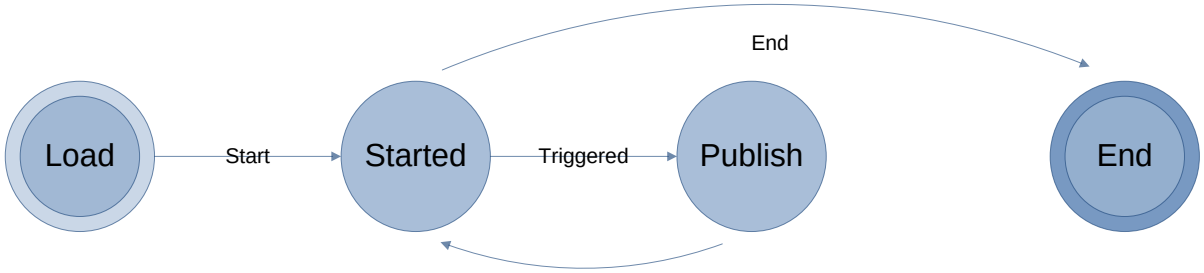

Figure 23 State diagram of LocationMonitor

5.3 Summary

In this chapter, I have presented a prototype system, DMSS, to illustrate the applicability of the three-level mobile spatial subscription model described in Chapter 3 and the plug-in approach implementation discussed in Chapter 4.

Referring to Chapter 2, the expressiveness of different types of DEBSs is shown in Table 11. Although the DMSS is based only on a simple map, it is has been suitable to show that the three-level mobile spatial subscription model, which is proposed in Chapter 3, is able to handle the currently proposed types of spatial subscriptions as well as new subscriptions types:

a. In contrast with pure string-based subscriptions that reply on matching names, I introduce matching based on physical area, and hence, DMSS is able to fully support the *equivalence* location-based abstraction;

b. Subscriptions that refer to locations relative to other more well-known locations;

c. The location-based abstraction *logical distance* can be supported.

Further, the proposed subscription model also supports dynamic subscriptions such as subscriptions involving locations with dynamic properties.

	Equivalence	Containment	Intersection	Adjacency	Physical Distance	Logical Distance	Direction
DMSS	●	●	●	●	●	●	●
String-based [18, 23, 28, 49]	◒	◒					
Geometric-based [31, 34, 39, 43, 54]	◒	●	●	●	●		●

● = completely supported ◒ = supported with limitations

Table 11 Expressiveness of different types of spatial subscriptions for DEBSs

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This thesis proposes a new model and prototype implementation for mobile spatial subscription. The contribution can be summarized as follows.

An analysis of existing DEBSs

First, a set of location-based abstractions are analyzed and their importance in expressing locations is discussed. The common location models that are able to express secondary identities are briefly introduced. Their expressiveness relative to the proposed abstractions is also analyzed. The possible ways to deploy the various location models to DEBSs are summarized and some extensions to enhance their support for location-based abstractions are suggested. The analysis ends by describing the capabilities of different types of DEBSs to handle the proposed abstractions.

A three-level mobile spatial subscription model

The thesis also proposes a three-level mobile spatial subscription model. The model allows subscribers to express spatial subscriptions proposed in Chapter 2. The first level is designed to handle spatial subscriptions consisting of geometric coordinates. The second level allows subscribers to specify locations using location labels and spatial relations instead of mathematical formulas. The third level extends the second level and allows subscribers to specify locations by stating their properties.

A plug-in approach for subscription conversion

In addition, the thesis presents a plug-in implementation approach to support spatial subscription conversions, so that existing DEBSs can support mobile spatial subscriptions with no modification to the middleware. Hence, software reusability is enhanced.

An implementation of a prototype system

Finally, I implemented a prototype system based on an existing DEBS, GEM. I introduced a plug-in component to the GEM, and illustrated, using a case study that the proposed model is able to handle most types of spatial subscriptions.

6.2 Future work

It is necessary in the future to extend the prototype system. Currently, DMSS is implemented based on GEM, a typed based DEBS. The extension needs to modify the data transmission protocol of the Mobile Subscription Conversion Unit (MSCU), so that it can be also adopted by subject-based and content-based DEBS.

Second, it is necessary to refine the MSCU by investigating how to reduce the overhead induced by the communication between the MSCU and the middleware.

Finally, the plug-in approach can be used to support advanced subscriptions in other domains. In this thesis, the plug-in approach is used only to convert new types of spatial subscriptions into existing types which the system can recognize. This approach can be adopted in other domains, such as *time*. These conversion components can be plugged into the systems when it is necessary, and unplugged when they are not in use. In this way, DEBSs can become more flexible to support different types of subscriptions.

Appendix A

Component Interfaces and Event Schemas of DMSS

Component Interface of InfoPublisher

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ComponentInterfaceAdvertisement>
<ComponentInterfaceAdvertisement xmlns:jxta="http://jxta.org">
	<InterfaceName>
		gemdemo.sensor.InfoPublisher
	</InterfaceName>
	<Extend>
		gem.core.GenericComponentInterface
	</Extend>
	<InputEventSchema>
		<SchemaName>
			gemdemo.sensor.IntervalChg
		</SchemaName>
		<SenderInterfaceName>
			anyone
		</SenderInterfaceName>
	</InputEventSchema>
	<OutputEventSchema>
		<SchemaName>
			gemdemo.sensor.WeatherSensorData
		</SchemaName>
		<ImplementationRole>
			anyone
		</ImplementationRole>
		<TTL>
			10000
		</TTL>
	</OutputEventSchema>
	<Behavior>
		<scxml xmlns="http://www.w3.org/2005/07/scxml" xmlns:gem="http://gem/CORE"
```

```
initialstate="loaded" version="1.0">
                        <state id="error" final="true">
                                <onentry>
                                        <gem:do action="error"/>
                                </onentry>
                                <transition event="gem.interface.commmand.generic.Stop">
                                        <target next="stopped"/>
                                </transition>
                        </state>
                        <state id="loaded">
                                <onentry>
                                        <gem:do action="load"/>
                                </onentry>
                                <transition event="gem.interface.commmand.generic.Start">
                                        <target next="started"/>
                                        <gem:do action="start"/>
                                </transition>
                        </state>
                        <state id="started">
                                <transition event="internal::timer">
                                        <target next="started"/>
                                        <gem:do action="reportData"
outputEventName="gemdemo.sensor.WeatherSensorData"/>
                                </transition>
                                <transition event="gemdemo.sensor.IntervalChg">
                                        <target next="started"/>
                                        <gem:do action="changeInterval"/>
                                </transition>
                                <transition event="gem.interface.commmand.generic.Stop">
                                        <target next="stopped"/>
                                </transition>
                        </state>
                        <state final="true" id="stopped">
                                <onentry>
                                        <gem:do action="stop"/>
```

```
                                    </onentry>
                            </state>
                    </scxml>
            </Behavior>
</ComponentInterfaceAdvertisement>
```

Component Interface of InfoSubscriber

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ComponentInterfaceAdvertisement>
<ComponentInterfaceAdvertisement xmlns:jxta="http://jxta.org">
        <InterfaceName>
                gemdemo.sensor.Sensor
        </InterfaceName>
        <Extend>
                gem.core.GenericComponentInterface
        </Extend>
        <InputEventSchema>
                <SchemaName>
                        gemdemo.sensor.WeatherSensorData
                </SchemaName>
                <SenderInterfaceName>
                        anyone
                </SenderInterfaceName>
        </InputEventSchema>
        <InputEventSchema>
                <SchemaName>
                        gemdemo.sensor.IntervalChg
                </SchemaName>
                <SenderInterfaceName>
                        anyone
                </SenderInterfaceName>
        </InputEventSchema>
        <OutputEventSchema>
                <SchemaName>
```

```
                gemdemo.sensor.SpatialSubData
            </SchemaName>
            <ImplementationRole>
                anyone
            </ImplementationRole>
            <TTL>
                10000
            </TTL>
        </OutputEventSchema>
        <Behavior>
            <scxml xmlns="http://www.w3.org/2005/07/scxml" xmlns:gem="http://gem/CORE"
initialstate="loaded" version="1.0">
              <state id="error" final="true">
                    <onentry>
                        <gem:do action="error"/>
                    </onentry>
                    <transition event="gem.interface.commmand.generic.Stop">
                        <target next="stopped"/>
                    </transition>
              </state>
                <state id="loaded">
                    <onentry>
                        <gem:do action="load"/>
                    </onentry>
                    <transition event="gem.interface.commmand.generic.Start">
                        <target next="started"/>
                        <gem:do action="start"/>
                    </transition>
                </state>
                <state id="started">
                    <transition event="internal::timer">
                        <target next="started"/>
                        <gem:do action="issueSubscription"
outputEventName="gemdemo.sensor.SpatialSubData"/>
                    </transition>
```

```
                                <transition event="gemdemo.sensor.WeatherSensorData">
                                        <target next="started"/>
                                        <gem:do action="collectWeatherSensorData"/>
                                </transition>
                                <transition event="gemdemo.sensor.IntervalChg">
                                        <target next="started"/>
                                        <gem:do action="changeInterval"/>
                                </transition>
                                <transition event="gem.interface.commmand.generic.Stop">
                                        <target next="stopped"/>
                                </transition>
                        </state>
                        <state final="true" id="stopped">
                                <onentry>
                                        <gem:do action="stop"/>
                                </onentry>
                        </state>
                </scxml>
        </Behavior>
</ComponentInterfaceAdvertisement>
```

Component Interface of LocEventProcessor

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ComponentInterfaceAdvertisement>
<ComponentInterfaceAdvertisement xmlns:jxta="http://jxta.org">
        <InterfaceName>
                gemdemo.sensor.LocEventProcessor
        </InterfaceName>
        <Extend>
                gem.core.GenericComponentInterface
        </Extend>
        <InputEventSchema>
                <SchemaName>
                        gemdemo.sensor.WeatherSensorData
```

```
            </SchemaName>
            <SenderInterfaceName>
                anyone
            </SenderInterfaceName>
        </InputEventSchema>
        <InputEventSchema>
            <SchemaName>
                gemdemo.sensor.LocationUpdateData
            </SchemaName>
            <SenderInterfaceName>
                anyone
            </SenderInterfaceName>
        </InputEventSchema>
        <InputEventSchema>
            <SchemaName>
                gemdemo.sensor.SpatialSubData
            </SchemaName>
            <SenderInterfaceName>
                anyone
            </SenderInterfaceName>
        </InputEventSchema>
        <OutputEventSchema>
            <SchemaName>
                gemdemo.sensor.IntervalChg
            </SchemaName>
            <ImplementationRole>
                anyone
            </ImplementationRole>
            <TTL>
                10000
            </TTL>
        </OutputEventSchema>
        <OutputEventSchema>
            <SchemaName>
```

```
                    gemdemo.sensor.WeatherSensorData
                </SchemaName>
                <ImplementationRole>
                    anyone
                </ImplementationRole>
                <TTL>
                    10000
                </TTL>
            </OutputEventSchema>
            <Behavior>
                <scxml xmlns="http://www.w3.org/2005/07/scxml" xmlns:gem="http://gem/CORE"
initialstate="loaded" version="1.0">
                  <state id="error" final="true">
                        <onentry>
                            <gem:do action="error"/>
                        </onentry>
                        <transition event="gem.interface.commmand.generic.Stop">
                            <target next="stopped"/>
                        </transition>
                  </state>
                    <state id="loaded">
                        <onentry>
                            <gem:do action="load"/>
                        </onentry>
                        <transition event="gem.interface.commmand.generic.Start">
                            <target next="started"/>
                            <gem:do action="start"/>
                        </transition>
                    </state>
                    <state id="started">
                        <transition event="gemdemo.sensor.WeatherSensorData">
                            <target next="started"/>
                            <gem:do action="convertWeatherData"
outputEventName="gemdemo.sensor.WeatherSensorData"/>
                        </transition>
```

```
                                <transition event="gemdemo.sensor.SpatialSubData">
                                        <target next="started"/>
                                        <gem:do action="collectSpatialSubData"/>
                                </transition>
                                <transition event="gemdemo.sensor.LocationUpdateData">
                                        <target next="started"/>
                                        <gem:do action="collectLocationUpdateData"/>
                                </transition>
                                <transition event="gem.interface.commmand.generic.Stop">
                                        <target next="stopped"/>
                                </transition>
                        </state>
                        <state final="true" id="stopped">
                                <onentry>
                                        <gem:do action="stop"/>
                                </onentry>
                        </state>
                </scxml>
        </Behavior>
</ComponentInterfaceAdvertisement>
```

Component Interface of Sensor

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ComponentInterfaceAdvertisement>
<ComponentInterfaceAdvertisement xmlns:jxta="http://jxta.org">
        <InterfaceName>
                gemdemo.sensor.Sensor
        </InterfaceName>
        <Extend>
                gem.core.GenericComponentInterface
        </Extend>
        <InputEventSchema>
                <SchemaName>
                        gemdemo.sensor.IntervalChg
```

```
                </SchemaName>
                <SenderInterfaceName>
                        anyone
                </SenderInterfaceName>
        </InputEventSchema>
        <OutputEventSchema>
                <SchemaName>
                        gemdemo.sensor.LocationUpdateData
                </SchemaName>
                <ImplementationRole>
                        anyone
                </ImplementationRole>
                <TTL>
                        10000
                </TTL>
        </OutputEventSchema>
        <Behavior>
                <scxml xmlns="http://www.w3.org/2005/07/scxml" xmlns:gem="http://gem/CORE"
initialstate="loaded" version="1.0">
                  <state id="error" final="true">
                                <onentry>
                                        <gem:do action="error"/>
                                </onentry>
                                <transition event="gem.interface.commmand.generic.Stop">
                                        <target next="stopped"/>
                                </transition>
                  </state>
                        <state id="loaded">
                                <onentry>
                                        <gem:do action="load"/>
                                </onentry>
                                <transition event="gem.interface.commmand.generic.Start">
                                        <target next="started"/>
                                        <gem:do action="start"/>
                                </transition>
```

```
                        </state>
                        <state id="started">
                                <transition event="internal::timer">
                                        <target next="started"/>
                                        <gem:do action="reportData"
outputEventName="gemdemo.sensor.LocationUpdateData"/>
                                </transition>
                                <transition event="gemdemo.sensor.IntervalChg">
                                        <target next="started"/>
                                        <gem:do action="changeInterval"/>
                                </transition>
                                <transition event="gem.interface.commmand.generic.Stop">
                                        <target next="stopped"/>
                                </transition>
                        </state>
                        <state final="true" id="stopped">
                                <onentry>
                                        <gem:do action="stop"/>
                                </onentry>
                        </state>
                </scxml>
        </Behavior>
</ComponentInterfaceAdvertisement>
```

Event Schema of LocationUpdateData

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ComponentInterfaceAdvertisement>
<ComponentInterfaceAdvertisement xmlns:jxta="http://jxta.org">
        <InterfaceName>
                gemdemo.sensor.Sensor
        </InterfaceName>
        <Extend>
                gem.core.GenericComponentInterface
        </Extend>
```

```
        <InputEventSchema>
                <SchemaName>
                        gemdemo.sensor.IntervalChg
                </SchemaName>
                <SenderInterfaceName>
                        anyone
                </SenderInterfaceName>
        </InputEventSchema>
        <OutputEventSchema>
                <SchemaName>
                        gemdemo.sensor.LocationUpdateData
                </SchemaName>
                <ImplementationRole>
                        anyone
                </ImplementationRole>
                <TTL>
                        10000
                </TTL>
        </OutputEventSchema>
        <Behavior>
                <scxml xmlns="http://www.w3.org/2005/07/scxml" xmlns:gem="http://gem/CORE"
initialstate="loaded" version="1.0">
                  <state id="error" final="true">
                                <onentry>
                                        <gem:do action="error"/>
                                </onentry>
                                <transition event="gem.interface.commmand.generic.Stop">
                                        <target next="stopped"/>
                                </transition>
                  </state>
                        <state id="loaded">
                                <onentry>
                                        <gem:do action="load"/>
                                </onentry>
                                <transition event="gem.interface.commmand.generic.Start">
```

```
                                        <target next="started"/>
                                        <gem:do action="start"/>
                                </transition>
                        </state>
                        <state id="started">
                                <transition event="internal::timer">
                                        <target next="started"/>
                                        <gem:do action="reportData"
outputEventName="gemdemo.sensor.LocationUpdateData"/>
                                </transition>
                                <transition event="gemdemo.sensor.IntervalChg">
                                        <target next="started"/>
                                        <gem:do action="changeInterval"/>
                                </transition>
                                <transition event="gem.interface.commmand.generic.Stop">
                                        <target next="stopped"/>
                                </transition>
                        </state>
                        <state final="true" id="stopped">
                                <onentry>
                                        <gem:do action="stop"/>
                                </onentry>
                        </state>
                </scxml>
        </Behavior>
</ComponentInterfaceAdvertisement>
```

Event Schema of SensorData

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE EventSchema>
<EventSchema>
        <ComplexType Name="GPSPosition">
                <Attr Name="longitude" Type="FLOAT"/>
                <Attr Name="latitude"  Type="FLOAT"/>
```

```
        </ComplexType>
        <ComplexType Name="gemdemo.sensor.SensorData">
                <Attr Name="location"      Type="GPSPosition"/>
                <Attr Name="temperature" Type="FLOAT" minOccur="1" maxOccur="5"/>
        </ComplexType>
        <EventSchemaName>gemdemo.sensor.SensorData</EventSchemaName>
</EventSchema>
```

Event Schema of SpatialSubData

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE EventSchema>
<EventSchema>
        <ComplexType Name="Area">
                <Attr Name="x1" Type="FLOAT"/>
                <Attr Name="y1"  Type="FLOAT"/>
                <Attr Name="x2" Type="FLOAT"/>
                <Attr Name="y2"  Type="FLOAT"/>
        </ComplexType>
        <ComplexType Name="Contexts">
                <Attr Name="weather" Type="STRING" minOccur="0" maxOccur="1"/>
                <Attr Name="no_of_people"  Type="FLOAT" minOccur="0" maxOccur="1"/>
        </ComplexType>
        <ComplexType Name="gemdemo.sensor.SpatialSubData">
                <Attr Name="spatialtype"          Type="STRING" minOccur="1" maxOccur="1"/>
                <Attr Name="locationarea"         Type="Area" minOccur="0" maxOccur="1"/>
                <Attr Name="locationname"         Type="STRING" minOccur="0" maxOccur="1"/>
                <Attr Name="locationform"         Type="STRING" minOccur="0" maxOccur="1"/>
                <Attr Name="spatialrelation"      Type="STRING" minOccur="0" maxOccur="1"/>
                <Attr Name="othercontexts"        Type="Contexts" minOccur="0" maxOccur="1"/>
        </ComplexType>
        <EventSchemaName>gemdemo.sensor.SpatialSubData</EventSchemaName>
</EventSchema>
```

Event Schema of WeatherSensorData

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE EventSchema>
<EventSchema>
        <ComplexType Name="gemdemo.sensor.WeatherSensorData">
                <Attr Name="location"      Type="STRING"/>
                <Attr Name="weather" Type="STRING"/>
        </ComplexType>
        <EventSchemaName>gemdemo.sensor.WeatherSensorData</EventSchemaName>
</EventSchema>
```

Bibliography

[1] R. Blanco, J. Wang and P. Alencar, "A metamodel for distributed event based systems," in Proceedings of the 2nd International Conference on Distributed Event-based Systems (DEBS), pp. 221-232, 2008.

[2] J.H. Schiller and A. Voisard, Location-based services, San Francisco, CA: Morgan Kaufmann Publishers, 2004,

[3] S. Spiekermann and H. Berlin, "General Aspects of Location-Based Services," in In Location-Based, 2004.

[4] J. Mathew, S. Sarker and U. Varshney, "M-commerce services: Promises and challenges," Communications of the Association for Information Systems, pp. 1-11, 2004.

[5] H. Jacobsen, "Middleware for Location-Based Services," in Location-Based Services, J. Schiller and A. Voisard, Morgan Kaufmann, 2004, pp. 83-114.

[6] P. Eugster, B. Garbinato and A. Holzer, "Location-based Publish/Subscribe," in Proceedings of the 4th IEEE International Symposium on Network Computing and Applications (NCA), pp. 279-282, 2005.

[7] V. Muthusamy, M. Petrovic, D. Gao and H. Jacobsen, "Publisher Mobility in Distributed Publish/Subscribe Systems," in Proceedings of the 4th International Workshop on Distributed Event-Based Systems (ICDCSW), pp. 421-427, 2005.

[8] P. Eugster, B. Garbinato and A. Holzer, "Pervaho: A Development & Test Platform for Mobile Ad hoc Applications," 2006 3rd Annual International Conference on Mobile and Ubiquitous Systems: Networking & Services, pp. 1-5, 2006.

[9] B. Jiang and X. Yao, "Location-based services and GIS in perspective," Comput., Environ.Urban Syst., vol. 30, pp. 712-725, November. 2006.

[10] Y. Huang and H. Garcia-Molina, "Publish/subscribe in a mobile environment," Wireless Networks, vol. 10, pp. 643-652, 2004.

[11] C. Becker and F. Dürr, "On location models for ubiquitous computing," Personal and Ubiquitous Computing, vol. 9, pp. 20-31, 2005.

[12] U. Leonhardt, "Supporting Location-Awareness in Open Distributed Systems," 1998.

[13] J. Zhang, M. Zhu, D. Papadias, Y. Tao and D.L. Lee, "Location-based spatial queries," in Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD), pp. 443-454, 2003.

[14] M. McKenney, A. Pauly, R. Praing and M. Schneider, "Preserving local topological relationships," in Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems (ACM-GIS), pp. 123-130, 2006.

[15] G. Cugola and de Cote, Jose Enrique Munoz, "On Introducing Location Awareness in Publish-Subscribe Middleware," in Proceedings of the 4th International Workshop on Distributed Event-Based Systems (DEBS), pp. 377-382, 2005.

[16] P. Eugster, P. Felber, R. Guerraoui and A.-. Kermarrec, "The many faces of publish/subscribe," ACM Computing Surveys, vol. 35, pp. 114-131, 2003.

[17] P. Eugster, "Type-based publish/subscribe: Concepts and experiences," ACM Trans.Program.Lang.Syst., vol. 29, pp. 6, 2007.

[18] H. Leung, I. Burcea and H. Jacobsen, "Modeling location-based services with subject spaces," in Proceedings of the 2003 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON), pp. 171-181, 2003.

[19] M. Altinel and M. Franklin, "Efficient Filtering of XML Documents for Selective Dissemination of Information," in Proceedings of the 26th International Conference on Very Large Data Bases, pp. 53-64, 2000.

[20] P. Eugster, "Type-based publish/subscribe," 2001.

[21] P. Pietzuch and J. Bacon, "Hermes: A Distributed Event-Based Middleware Architecture," in Proceedings of the 22nd International Conference on Distributed Computing Systems, pp. 611-618, 2002.

[22] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. Strom and D. Sturman, "An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems," in Proceedings of the 19th IEEE International Conference on Distributed Computing Systems, pp. 262, 1999.

[23] A. Carzaniga, D. Rosenblum and A.L. Wolf, "Design and evaluation of a wide-area event notification service," ACM Trans.Comput.Syst., vol. 19, pp. 332-383, 2001.

[24] G. Cugola, E. Di Nitto and A. Fuggetta, "The JEDI Event-Based Infrastructure and Its Application to the Development of the OPSS WFMS," IEEE Trans.Softw.Eng., vol. 27, pp. 827-850, 2001.

[25] J. Bacon, K. Moody, J. Bates, R. Hayton, C. Ma, A. McNeil, O. Seidel and SpiteriMark, "Generic Support for Distributed Applications," Computer, vol. 33, pp. 68-76, 2000.

[26] A. Carzaniga, "Architectures for an Event Notification Service Scalable to Wide-area Networks," 1998.

[27] P. Eugster and R. Guerraoui, "Content-based publish/subscribe with structural reflection," in Proceedings of the 6th conference on USENIX Conference on Object-Oriented Technologies and Systems, pp. 10-10, 2001.

[28] H. Leung, "Subject space: a state-persistent model for publish/subscribe systems," in Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research (CASCON), pp. 7, 2002.

[29] S. Arulampalam, S. Maskell, N. Gordon and T. Clapp, "A Tutorial on Particle Filters for On-line Non-linear/Non-Gaussian Bayesian Tracking," IEEE Transactions on Signal Processing, vol. 50, pp. 174-188, February. 2002.

[30] L. Fiege and G. Muhl, "Rebeca Event-Based Electronic Commerce Architecture," 2000.

[31] Object Management Group (OMG), "CORBA event service specification, version 1.0," 2000.

[32] K. Walzer, T. Breddin and M. Groch, "Relative temporal constraints in the Rete algorithm for complex event detection," in Proceedings of the 2nd International Conference on Distributed Event-based Systems (DEBS), pp. 147-155, 2008.

[33] J. Aasman, "Unification of geospatial reasoning, temporal logic, & social network analysis in event-based systems," in Proceedings of the 2nd international conference on Distributed event-based systems (DEBS), pp. 139-145, 2008.

[34] S. Schwiderski-Grosche and K. Moody, "The SpaTeC composite event language for spatio-temporal reasoning in mobile systems," in Proceedings of the 3rd ACM International Conference on Distributed Event-Based Systems, pp. 1-12, 2009.

[35] N. Schultz-Moller, M. Migliavacca and P. Pietzuch, "Distributed complex event processing with query rewriting," in Proceedings of the 3rd ACM International Conference on Distributed Event-Based Systems, pp. 1-12, 2009.

[36] I. Kellner and L. Fiege, "Viewpoints in complex event processing: industrial experience report," in Proceedings of the 3rd ACM International Conference on Distributed Event-Based Systems, pp. 1-8, 2009.

[37] Y. Turchin, A. Gal and S. Wasserkrug, "Tuning complex event processing rules using the prediction-correction paradigm," in Proceedings of the Third ACM International Conference on Distributed Event-Based Systems, pp. 1-12, 2009.

[38] D. Anicic, P. Fodor, N. Stojanovic and R. Stuhmer, "An approach for data-driven and logic-based complex Event Processing," in Proceedings of the 3rd ACM International Conference on Distributed Event-based Systems, pp. 1-2, 2009.

[39] I. Burcea and H. Jacobsen, "L-ToPSS – Push-Oriented Location-Based Services," Technologies for E-Services, pp. 131-142, 2003.

[40] Z. Xu and H. Jacobsen, "Efficient constraint processing for highly personalized location based services," in Proceedings of the 30th International Conference on Very Large Database (VLDB), pp. 1285-1288, 2004.

[41] Z. Xu and H. Jacobsen, "Evaluating Proximity Relations Under Uncertainty," in Proceedings of the 23rd International Conference on Data Engineering (ICDE), pp. 876-885, 2007.

[42] X. Chen, Y. Chen and F. Rao, "An efficient spatial publish/subscribe system for intelligent location-based services," in Proceedings of the 2nd International Workshop on Distributed Event-based Systems (DEBS), pp. 1-6, 2003.

[43] Y. Chen, F. Rao, X. Yu and D. Liu, "CAMEL: A Moving Object Database Approach for Intelligent Location Aware Services," in Proceedings of the 4th International Conference on Mobile Data Management (MDM), pp. 331-334, 2003.

[44] Z. Xu and H. Jacobsen, "Efficient constraint processing for location-aware computing," in Proceedings of the 6th International Conference on Mobile Data Management (MDM), pp. 3-12, 2005.

[45] K. Pripužić, I.P. Žarko and K. Aberer, "Top-k/w publish/subscribe: finding k most relevant publications in sliding time window w," in Proceedings of the 2nd International Conference on Distributed Event-based Systems (DEBS), pp. 127-138, 2008.

[46] V. Muthusamy, H. Liu and H. Jacobsen, "Predictive publish/subscribe matching," in Proceedings of the 4th ACM International Conference on Distributed Event-Based Systems, pp. 14-25, 2010.

[47] S. Baehni, J. Barreto, P. Eugster and R. Guerraoui, "Efficient distributed subtyping tests," in Proceedings of the 2007 1st International Conference on Distributed Event-based Systems (DEBS), pp. 214-225, 2007.

[48] Y. Huang and H. Garcia-Molina, "Parameterized subscriptions in publish/subscribe systems," Data Knowl.Eng., vol. 60, pp. 435-450, 2007.

[49] H. Leung and H. Jacobsen, "Efficient matching for state-persistent publish/subscribe systems," in CASCON '03: Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research, pp. 182-196, 2003.

[50] Z. Xu and A. Jacobsen, "Adaptive location constraint processing," in Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data (SIGMOD), pp. 581-592, 2007.

[51] F. Chen, C. Yang, W. Yu, X. Le and J. Yang, "Research on mobile GIS based on LBS," Proceedings of the 2005 IEEE International Geoscience and Remote Sensing Symposium (IGARSS), vol. 2, pp. 4 pp., 2005.

[52] J. Wang, "Interface-based modular approach for designing distributed event-based systems," pp. xi, 105 p., 2008.

[53] JXTA(TM) Community, "JXTA JavaTM Standard Edition v2.5: Programmers Guide." September 2007.

[54] P. Eugster, B. Garbinato and A. Holzer, "Design and Implementation of the Pervaho Middleware for Mobile Context-Aware Applications," in Proceedings of the 2008 International MCETECH Conference on e-Technologies, pp. 125-135, 2008.