

**UCGE Reports
Number 20289**

Department of Geomatics Engineering

Geospatial-based Publish/Subscribe: Improving Real-time Notification and Situational Awareness in Fire Emergency

(URL: <http://www.geomatics.ucalgary.ca/research/publications>)

by

Ala' Kassab

August 2009



UNIVERSITY OF CALGARY

Geospatial-based Publish/Subscribe: Improving Real-time Notification and Situational
Awareness in Fire Emergency

by

Ala' S. Kassab

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF GEOMATICS ENGINEERING

CALGARY, ALBERTA

AUGUST, 2009

© Ala' S. Kassab 2009

Abstract

The mission of fire services plays an essential role in protecting lives, property and natural resources. The dynamics of emergency situations and developing fire hazards require integrating disparate sources of information and distributed parties or clients to maintain situational awareness and timely decision making. The traditional request/reply interaction style may function inefficiently in time-sensitive scenarios, where real-time notification about critical events is needed. This thesis proposes *Geospatial-based Publish/Subscribe*, an event-based interaction framework for transacting dynamic geospatial events in real-time manner. Also, a system prototype, called *Real-time Fire Emergency Response System (RFERS)*, has been developed. The system has been designed to integrate multiple sources of geospatial events required in the context of fire emergencies. The system clients can be notified about geospatial events of interests and interactively visualize the events in a GIS application. Simulated emergency scenarios have been demonstrated and experiments have been conducted to evaluate the performance of the interaction. The results have shown the efficiency of the developed RFERS matching engine in searching for matched interests registered by subscribers.

Acknowledgements

This thesis would not have been possible without the grace of God and the support of many people.

First and foremost, I am sincerely grateful to my supervisor, Dr. Yang Gao, who gave me the opportunity to join his research group at the Department of Geomatics Engineering, University of Calgary. His continuous guidance, support, and encouragement assisted me throughout the research for and writing of this thesis. I would like also to thank my co-supervisor, Dr. Steve Liang, whose stimulating suggestions and creative ideas have always been the source of my inspiration for carrying out the research challenges. I would like to acknowledge NSERC, GEOIDE, and the Department of Geomatics Engineering for their funding and financial support.

Thanks to my colleagues, Dr. Hamid Assilzadeh, Nand Jha, Tim Liu, and Dr. Zhinong Zhong, for their scholastic support. I am speechless to describe the immense support I have had from Nour El Messiri, Abdel Rahman Muhsen, and Yaser Ghanam in refining the writing of this thesis. Special thanks to my friends, Derar Alassi, Hosam Al-Kordi, Mohammed Alshalalfa, Mohannad Al-Durgham, and Thaer Shunnar who have always been there for me all the way through my study.

This acknowledgement would not be completed without mentioning the endless love of my family. Thank you.

Dedication

To whom I owe everything, to my beloved parents,

Shawqi and Nabeela...

Table of Contents

Abstract	ii
Acknowledgements	iii
Dedication	iv
Table of Contents	v
List of Tables	viii
List of Figures and Illustrations	ix
List of Symbols, Abbreviations and Nomenclature	xi
CHAPTER ONE: INTRODUCTION	1
1.1 Introduction	1
1.2 Publish/Subscribe Paradigm	1
1.3 Fire Emergency Response and Situational Awareness	3
1.4 Problem Definition and Motivation	5
1.5 Research Objectives	8
1.6 Research Methodology	8
1.7 Contribution	10
1.8 Thesis Organization	11
1.9 Summary	12
CHAPTER TWO: EVENT-BASED AND PUBLISH/SUBSCRIBE SYSTEMS	13
2.1 Introduction	13
2.2 Event and Notification	14
2.3 Producers and Consumers	17
2.4 Event-Notification Service	19
2.4.1 Centralized	21
2.4.2 Distributed	22
2.5 Subscription and Expressiveness	23
2.5.1 Topic-based Publish/Subscribe	25
2.5.2 Content-based Publish/Subscribe	26
2.6 Notification Matching Algorithms	28
2.6.1 Brute Force Method	29
2.6.2 Counting-based Algorithm	30
2.6.3 Tree-based Algorithm	32
2.7 Related Work	34
2.8 Summary	39
CHAPTER THREE: GEOSPATIAL-BASED PUBLISH/SUBSCRIBE INTERACTION	42
3.1 Introduction	42
3.2 Geospatial Semantics of Events	43
3.3 Geospatial Event Clients and the Interaction Flow	45
3.4 Design Considerations	46
3.5 Geospatial-based Publish/Subscribe Data Models	48
3.5.1 Geospatial Notification Data Model	48
3.5.2 Geospatial Subscription Data Model	50

3.6 Geospatial Notification Matching.....	54
3.6.1 Matching Problem	55
3.6.2 Matching Challenges	57
3.6.3 Improving the Matching Process.....	59
3.6.3.1 Spatial Data Indexing.....	59
3.6.3.2 Indexing of Geospatial Subscriptions	61
3.6.3.3 The Proposed Geospatial Notification Matching Approach.....	63
3.7 Delivery of Geospatial Notifications	67
3.8 Summary	69
CHAPTER FOUR: DEVELOPMENT OF REAL-TIME FIRE EMERGENCY RESPONSE SYSTEM.....	71
4.1 Introduction.....	71
4.2 System Design and Architecture.....	72
4.2.1 RFERS Topics and Data Models.....	74
4.2.1.1 Emergency Asset Locations.....	75
4.2.1.2 Wireless Sensor Observations	76
4.2.1.3 Fire Incident Reports	78
4.2.1.4 Wildfire Thermal-Infrared Images.....	81
4.2.2 Client Tier.....	84
4.2.3 Business Logic Tier	87
4.2.4 Database Tier.....	92
4.3 Prototype Implementation.....	94
4.3.1 Development Tools and Software Packages	95
4.3.2 Simulation of Geospatial Notifications	95
4.3.3 Subscriber GIS Application.....	98
4.3.4 Middleware Server	102
4.4 Summary.....	103
CHAPTER FIVE: TESTING RESULTS AND PERFORMANCE.....	106
5.1 Introduction.....	106
5.2 RFERS Prototype Testing.....	107
5.2.1 Emergency Assets Close by a Fire Incident	108
5.2.2 High Temperature and Low Relative Humidity Observations of Wireless Sensors	112
5.2.3 Wildfire Monitoring Using Remote Sensing Thermal-Imaging	114
5.2.4 Discussion.....	116
5.3 Performance Evaluation.....	122
5.3.1 Evaluation Metric	122
5.3.2 Simulation Environment.....	123
5.3.3 Experiments	125
5.3.3.1 Number of Geospatial Subscriptions	126
5.3.3.2 Brute Force versus the Proposed Matching Approach	128
5.3.3.3 Effectiveness of Spatial Indexing	130
5.3.4 Discussion.....	132
5.4 Summary.....	133

CHAPTER SIX: CONCLUSIONS AND FUTURE WORK	135
6.1 Introduction.....	135
6.2 Conclusions and Limitations	136
6.3 Future Work.....	141
6.4 Summary.....	142
REFERENCES	143
APPENDIX A: TOPOLOGICAL RELATIONSHIPS BETWEEN SIMPLE GEOMETRIES	149

List of Tables

Table 3.1: Examples of name/value pair formatting with a geometry data type	50
Table 3.2: Examples of spatial interests' expressions and their respective spatial predicates formation.....	54
Table 3.3: An example of content data structure for geospatial notifications	69
Table 4.1: Geospatial notification content-data model in Emergency Asset Locations topic.....	76
Table 4.2: Geospatial notification content-data model in Wireless Sensor Observations topic	77
Table 4.3: Geospatial notification content-data model in Fire Incident Reports topic.....	81
Table 4.4: Geospatial notification content-data model in Wildfire Remote Sensing Images topic	84
Table 4.5: The spatial queries executed in the spatial matching process.....	91
Table 4.6: Data fields structure of the geospatial subscription tables.....	93

List of Figures and Illustrations

Figure 2.1: Publish/Subscribe system components.....	14
Figure 2.2: Publish/Subscribe Model.....	20
Figure 2.3: (a) Centralized and (b) Distributed architecture of publish/subscribe notification service.....	21
Figure 2.4: An example of content-based matching using Counting algorithm.....	32
Figure 2.5: An example of content-based matching using Tree-based algorithm.....	33
Figure 3.1: Geometrical representations of simple geographic features.....	50
Figure 3.2: Geospatial notification matching using the Brute Force method.....	58
Figure 3.3: An example of geospatial subscriptions clustering process.....	65
Figure 4.1: High-level architecture of RFERS.....	74
Figure 4.2: UAV in monitoring wildfire disasters.....	82
Figure 4.3: RFERS subscriber client application.....	86
Figure 4.4: The notification service process flow upon issuing geospatial subscription..	89
Figure 4.5: The notification service process flow upon publishing a geospatial notification.....	91
Figure 4.6: The simulation programs for publishing geospatial notifications into RFERS topics: (a) Emergency Asset Locations topic, (b) Wireless Sensor Observations topic, (c) Fire Incident Reports topic, and (d) Wildfire Thermal-Infrared Images topic.....	97
Figure 4.7: Subscriber’s application login screen.....	98
Figure 4.8: Subscriber’s application Map tab.....	99
Figure 4.9: Subscriber’s application Subscription/Notifications tab.....	101
Figure 5.1: Subscription and publishing scenario utilizing the “FireIncReports” topic of the RFERS prototype.....	110
Figure 5.2: Subscription and publishing scenario utilizing “Emergency Asset Location” topic of RFERS prototype.....	112
Figure 5.3: Subscription and publishing scenario utilizing “Wireless Sensor Observations” topic of RFERS prototype.....	114

Figure 5.4: Publishing and subscription scenario utilizing “Wildfire Remote Sensing Images” topic of RFERS topic.....	116
Figure 5.5: Simulated geospatial subscriptions dataset	125
Figure 5.6: Average matching time of a single geospatial notification using the proposed matching approach	127
Figure 5.7: Average matching time of a single geospatial notification using the brute force method	129
Figure 5.8: The effect of using spatial index on the average matching time	131

List of Symbols, Abbreviations and Nomenclature

Symbol	Definition
2D	Two Dimensional
3D	Three Dimensional
API	Application Programming Interface
CPU	Central Processing Unit
EMS	Enterprise Messaging Service
FSA	Forward Sortation Area
GB	Gigabyte
GeoTIFF	Georeferenced Tagged Image File Format
GIS	Geographic Information System
GPS	Global Positioning System
GUI	Graphical User Interface
LBS	Location Based Service
PDA	Personal Digital Assistant
RAM	Random-Access Memory
RFERS	Real-time Fire Emergency Response System
SDK	Software Development Kit
SMS	Short Message Service
SQL	Structured Query Language
UAV	Unmanned Aerial Vehicle
USDA	United States Department of Agriculture
WSN	Wireless Sensor Network

Chapter One: Introduction

1.1 Introduction

Fire emergency services and public safety agencies play an essential role in preventing loss of lives and property as well as in protecting environment resources. Acquiring the relevant geospatial information about an emergency event and its surroundings provides decision makers better insight to effectively determine the appropriate response tactics and allocate safety resources and personnel. The dynamics of the emergency situation and its circumstances require collecting information from disparate sources. Timely disseminating this volume of dynamic information among multiple users and responsible parties can be challenging. This research aims to develop an information communication framework based on the publish/subscribe interaction paradigm. By disseminating geospatial events (*i.e.*, events that are associated to the 2D geographic space) of interest to the right party at the right time, the proposed framework shall support public safety and heighten situational awareness.

This chapter starts with a brief introduction on the publish/subscribe interaction model and the application scope in fire emergency in Section 1.2 and Section 1.3, respectively. The remaining discusses the problem addressed in this research, the objectives, the research methodology, the contribution, and ends with the organization of the thesis document and the summary of this chapter.

1.2 Publish/Subscribe Paradigm

The Internet has become a primary and effective medium for distributed users to share information and resources all over the world. In such distributed computing

environment, prompt discovery and delivery of relevant information are highly demanded by users to help them make responsive and timely decisions. The interaction style, namely client/server or request/reply, has been traditionally adopted for transacting information in distributed systems. To obtain particular information, clients have to submit a request (*e.g.*, a database query) then wait to get the response containing the information from the addressed server (*i.e.*, information provider). This point-to-point and synchronous interaction has served information browsing activities very well. However, this interaction model is not necessarily efficient in all data communication activities, particularly, in responsive systems. Several deficiencies of the request/reply interaction model will be detailed in the problem definition of this research (Section 1.4). This issue has raised the need for more flexible communication model especially for dynamic, scalable, and responsive distributed applications.

Recently, the publish/subscribe paradigm has gained increasing attention from researchers and industry to overcome the burden of the request/reply model (Pietzuch 2004, Eugster *et al.* 2003). The publish/subscribe is an asynchronous, powerful, and event-driven communication paradigm that supports many-to-many interaction between event clients, where an event is a piece of information that represents an instantaneous occurrence or happening of interest. An event client can be an information producer (publisher), an information consumer (subscriber), or both at the same time. Event clients are exchanging events in form of data messages, called notifications, throughout the system network. Producers publish events that might be of interest to other clients, and consumers subscribe their theme of interests in events that they would like to receive. Consider the following example, a client monitors stocks and bases his decision in

trading certain stocks on current real-time quotes. The quoting server (*i.e.*, publisher) broadcasts current quotes for trading and the client (*i.e.*, subscriber) registers his interest to be notified about certain quotes' prices. The fundamental component of any publish/subscribe system is the middleware layer, namely the event-notification service. It delivers published events to their corresponding subscribers asynchronously and in timely manner. Event clients are loosely coupled in space and time in this style of interaction. This communication paradigm aids distributed systems in terms of scalability, integration of autonomous and heterogeneous components, intelligent dissemination of relevant information, and timely delivery of crucial events data.

The means of the publish/subscribe interaction model can efficiently meet the communication requirements of many distributed systems and applications. Specifically, the focus of this research, revolves around exploiting the publish/subscribe interaction paradigm in developing a robust and flexible information communication model for fire emergency response systems. The scope of this research's application area is discussed further in the next section.

1.3 Fire Emergency Response and Situational Awareness

Fire emergency systems are responsive in their nature; dispatching and rescuing operations are performed as response actions upon the occurrence of fire. Delaying the response would cause an increase in the loss of lives and property, where few seconds can separate between fire containment and flashover.

Fire services rely on acquiring accurate information that reflects the emergency environment before and during the occurrence of an event in order to conduct successful

response operations. Geographical data is broadly utilized to comprehensively describe the spatial and descriptive semantics of fire hazards and their surrounding features. The advent of the Geographical Information Systems (GIS) technology has widely supported the operations of planning, preparedness, mitigation, and delivery of fire services by effective storing, retrieving, analyzing, and processing of geospatial information (Maguire 2005). GIS technology has been turned into a key role component in the process of efficient management of crisis incidents (Zerger and Smith 2003). Moreover, leveraging such technology along with decision support systems results in numerous profits (Keramitsoglou *et al.* 2004, Wybo 1998). However, utilizing such systems comes in a stage following the collection of the current and up-to-date information about the emergency. Here, information communication plays a vital role in effectively connecting sources of situational data on one side with emergency response teams and other involved parties on the other side. As time is the critical factor in such situations, real-time awareness of crucial data is also essential. Otherwise, decisions would be delayed which in turn might lead to failure in the emergency response.

The nature of data that assists fire response operations can be classified into two categories: static and dynamic data. Geographical information such as locations of historical incidents, statistics, risk areas, street networks, points of interest and other features are included in the static data class as they are rarely changeable over time. Current GIS systems and applications are capable of handling this class of information efficiently. On the other hand, dynamic data encompasses sudden occurrences of events or rapidly changing information. Locations of emergency vehicles, meteorological sensor observations, 911 emergency calls, and airborne visible and infrared images of current

active fires are some examples of dynamic data. Acquiring such data and monitoring crucial occurrences of events highly requires an advanced information communication framework between dynamic data sources and responsible parties. Usually, fire emergencies involve responders from different teams and jurisdictions, including: firefighting crews, police stations, medical services, fire chiefs and managers. Timely dissemination of dynamic events is necessary to reduce the response time. However, the involved parties may have different interests in dynamic data according to their operational functions in the emergency. In other words, instead of tracking all the changes of dynamic data or events, they would rather prefer to be alarmed about certain happenings or specific changes in state. These requirements add more challenges to the current interaction and information dissemination protocols.

1.4 Problem Definition and Motivation

The information communication and transacting of dynamic observations between distributed components is a key function in fire emergency response and management systems (Turoff *et al.* 2004). However, the traditional interaction models are based on the request/reply communication style between distributed entities. These models may not be appropriate for modern requirements as they lack the following features:

1. *Real-time delivery and pushing of dynamic events data to interested clients:*

Usually, newly generated or updated information is made available in web servers for public use. Using the request/reply interaction, clients have to initiate requests for obtaining the latest data and the servers passively respond to the requests. As clients do not know when the required data will be available in the servers, they

tend to send requests to the servers in constant cycles of time. Periodic requesting, also called pulling, of data opposes the merit of real-time data acquisition as it usually results in either redundant data or delayed awareness of current, may be crucial, data. The former happens because of issuing requests too often, while the latter happens because of issuing requests too infrequently. Also, continuous pulling is a resource-intensive operation, it might overload the interacting components, and it might cause the entire system to break down (Franklin and Zdonik 1998).

2. *Many-to-many and asynchronous interaction style*: with the availability of numerous information sources over wide-area of networks, request/reply interaction makes it difficult for clients to track their needs of information from all different data sources. In the request/reply interaction, a client synchronously requests information from a server then waits until the server responds with a reply (Muhl 2002). The request operation is transacted between one client and one server at a time. In case the required information is available on multiple servers, the client needs to request the information one by one from the servers.
3. *Scalability and loose coupling of interacting clients*: part of this problem is introduced from the previous item. Interacting components are tightly coupled as a consequence of the synchronization procedure of interaction in the request/reply model; while waiting for the reply from the server, the client component stays quiescent until the request results come back to the client side (Muhl 2002). Any failure of either side breaks down the communication completely. Additionally, clients and servers should have previous knowledge about the identity (*i.e.*, IP

address) of each other in order to communicate by means of the request/reply model. At a time, a request is issued to a single server whose identity must be defined in the request operation. Then, the server replies to the client by a call back function that addresses this specific client. These issues significantly impede the scalability of distributed systems in terms of supporting growing number of clients (Pietzuch 2004). Also, they limit the ability of the system to grow and integrate new members in the interaction flow.

4. *Supporting heterogeneous and dynamic behaviours of clients*: in many cases, employing the request/reply style leads to static and rigid types of applications (Eugster *et al.* 2003). Thus, it would raise difficulties to accommodate clients who are joining and leaving the system without coordination or pre-configuration. Furthermore, clients require expressive ways to define their variety and precise interests in data, so that they can flexibly set roles that suit their demands and prevent irrelevant data to be delivered. This is cumbersome to achieve using the request/reply model.

Emergency response and management systems necessitate transacting information between components in a timely and flexible manner. This cannot be realized efficiently by using the request/reply model. There is a need of an alternative paradigm, namely event-based publish/subscribe, to alleviate the deficiencies of the traditional style as mentioned above. Achieving this can significantly support current emergency systems in fire crises, which in turns, improves situational awareness, decision making, and recovery

processes, hence preventing loss of lives and property. This is the main motivation of this research work.

1.5 Research Objectives

The primary goal addressed in this research is to outline an event-based interaction framework for real-time notification and dissemination of dynamic geospatial events to support situational awareness in fire hazards. To serve this primary goal, the following objectives are to be met:

1. Investigating the current theory and implementation aspects of event-based and publish/subscribe systems.
2. Incorporating geospatial type of events in the interaction scheme by designing suitable data models.
3. Designing a geospatial event-based publish/subscribe framework and improving the performance of the dissemination flow.
4. Developing a prototype system for real-time fire emergency response.
5. Testing and evaluating the performance of the interaction by simulating geospatial events in several emergency scenarios.

1.6 Research Methodology

The methodology of this research consists of four major components: literature review, design and data modeling, development and implementation, and finally testing and evaluation.

The research starts by a review of the literature relating to this research's objectives (see Section 1.5). One can notice that the research problem (see Section 1.4) is multidisciplinary; it combines the research field of event-based and publish/subscribe paradigm which is one recent concern of many computer science communities, the GIS and spatial databases research, and in addition to the information technology and systems development. As the publish/subscribe paradigm is the main focus, a large part of the literature review has been oriented to explore terminologies, interaction components, development tools, and techniques used in current publish/subscribe and event-based systems.

The second stage focuses on designing an extended publish/subscribe interaction model particularly suitable to incorporate spatial semantics of operations. The extended model is called *Geospatial-based Publish/Subscribe*. In the extended model, two data structures, *geospatial event/notification* and *geospatial subscription*, are proposed for the interaction operations. Based on these two data structures, the flow process between the publish/subscribe components, namely the publisher, the subscriber, and the event-notification service middleware, is defined. Moreover, an efficient approach for matching geospatial notifications against geospatial subscriptions is proposed.

The third stage is concerned with developing and implementing a system prototype named *Real-Time Fire Emergency Response System*. The above proposed model, *Geospatial-based Publish/Subscribe*, is realized in the system implementation. Two main software components were developed for the prototype: 1) the subscriber component, which is a GIS software intended for users to perform geospatial subscriptions and visualize received geospatial notifications interactively in GIS mapping

environment; and 2) the event-notification service, which is the mediator component residing between the system clients. Simulation programs were developed to publish several themes of geospatial events throughout the system network. Four types of geospatial event data structures, *aka* topics, were defined: emergency assets locations, wireless sensors observations, fire incidents reports, and wildfire remote sensing thermal images. The system was implemented using C# programming language, ArcGIS 9.3 products, and TIBCO Enterprise Messaging Service (EMS) 4.4.

Finally, to test the implemented system prototype, several scenarios are envisioned using simulated geospatial events to evaluate the proposed Geospatial-based Publish/Subscribe model. The performance of the system prototype was evaluated by measuring the time consumed to match and disseminate geospatial events among different numbers of registered subscribers.

1.7 Contribution

In this thesis, an extended framework called Geospatial-based Publish/Subscribe is developed which resembles a generic middleware for supporting information dissemination of dynamic geospatial events in fire emergencies. To date, our investigations have shown that no similar framework has been developed. The literature has presented a variety of applications to which publish/subscribe paradigm has been employed, including stock markets, news dissemination, control systems, network monitoring and many others. These applications incorporate transacting primitive types of events data, mainly textual data. Whereas, this research tackles more complex data structures, namely spatial events data, which encapsulates spatial representation of the 2D

geographic space. Recent research efforts have investigated the publish/subscribe interaction in geographical nature of applications, such as Location Based Services (LBS) and Wireless Sensor Networks (WSN). However, the publish/subscribe interaction was limited to include point type of events and spatial range query type of subscriptions. This research differs from the previous works in the sense that a wider range of spatial types of events as well as more expressiveness of spatial subscriptions are incorporated.

The major contribution of this research is the exploitation of events-based and publish/subscribe model in emergency and hazards applications. Furthermore, it is expected that the direction addressed in this research can be a motive to establish a standard geospatial-based publish/subscribe model recognizing the transaction of spatially-related events worldwide.

1.8 Thesis Organization

Chapter 2 gives the conceptual and theoretical background of this research. Chapter 3 introduces the proposed Geospatial-based Publish/Subscribe model including definitions of geospatial events and geospatial subscriptions, and also explains the proposed event-subscription matching approach by means of clustering and spatial indexing. Chapter 4 presents the Real-Time Fire Emergency Response System (RFERS) prototype designed and developed as part of this research. Chapter 5 discusses the testing of the system implementation with several simulation scenarios, and also provides an evaluation of the system prototype and the proposed matching approach performance in disseminating geospatial events. Finally, Chapter 6 concludes this research with a summary, limitations, and future works.

1.9 Summary

This chapter gave a brief presentation of the research topic by first introducing the publish/subscribe paradigm and its potential application in fire emergency and situational awareness. Next, it defined the research problem, and stated the research objectives. It then introduced the methodology through which the research problem has been tackled as well as the contribution of this research. Lastly, the thesis chapters were outlined.

Chapter Two: Event-based and Publish/Subscribe Systems

2.1 Introduction

Event-based systems are increasingly gaining attention and growing over the recent years. They manifest a powerful information-driven middleware for efficient interaction between clients in large-scale and distributed applications. *Publish/subscribe* is widely common interaction model used in event-based computing (Muhl *et al.* 2006). The key concept of this interaction paradigm is introducing the *middleware* component which facilitates the interaction between distributed clients. The middleware component takes the responsibility of conveying the information messages from *producer* clients (publishers), who generate and publish events information, to *consumer* clients (subscribers), who are interested in receiving the events' information message. In this form of interaction, clients are loosely coupled from each other. In other words, they interact without direct knowledge of each other. The only aspect that relates clients is the content data and values of the events information messages. Producers publish event messages, usually called notifications, throughout the system as they might be of interest to other clients. Consumers receive events that are only of interest as they previously have registered their need of events by specifying filters, called subscriptions. The middleware, called the *event-notification service*, handles published events, matches them with the registered subscriptions, and pushes the events to matched subscriber clients asynchronously and in a timely manner. This interaction procedure is what enables heterogeneous, autonomous, and dynamic clients or sensor devices to be integrated in the system and leads to better scalability and communication efficiency. The main components of publish/subscribe system are depicted in Figure 2.1.

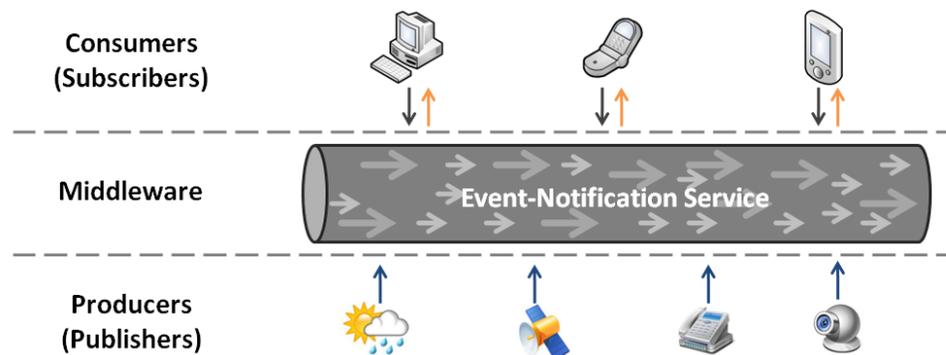


Figure 2.1: Publish/Subscribe system components

This chapter presents the theoretical background of event-based and publish/subscribe interaction model aiming to help the reader to conceptualize the key terminologies, components, system design, and techniques used in this research field. Section 2.2 defines the terms event and notification used in the context of this research. Section 2.3 and Section 2.4 describe the publish/subscribe system components and their roles in the interaction flow. Section 2.5 explains two publish/subscribe communication models by which producers and consumers transact events or notifications. Section 2.6 introduces several algorithms used to improve matching events with registered subscriptions, thus, enhancing the efficiency of events dissemination. Section 2.7 concludes by a review of some research works related to the topic of this research.

2.2 Event and Notification

The definition of the term *event* has been presented in many places in the literature. In this research, however, the definitions adopted by the distributed computing communities will be used. According to Mansouri-Samani & Sloman (1997), an event is defined as

any happening of interest or an instantaneous change of state. Also in Muhl *et al.* (2006), an event is regarded as any detectable change of state that can be observed from within a computer system. Despite the small discrepancies of defining an event, both definitions mentioned above and many others agree on two major characteristics of an event: firstly is the irregular and instantaneous occurrence and secondly that this occurrence is of interest to other clients or parties. The appearance of a person detected by sensors, an increase of a currency quote in stock markets, an observation of a temperature sensor, and a car accident on a highway street are some examples of an event. The concept of events is considered a proper abstraction for observing the dynamic nature of real world entities. Events are generated and broadcasted as data-of-interest packets usually to inform or notify disparate clients about the current state or crucial happening of associated objects. Detection and monitoring systems rely on receiving such events, and accordingly, appropriate actions would be taken when needed at the right time.

Modeling the abstraction of an event might be given different forms and semantics. Some of the critical aspects are considered essentials in defining an event model (Carzaniga 1998), including: (1) events have duration or not, (2) binding events with objects that relate to or else events are modeled irrespectively of their origins, (3) modeling the observation mechanism of events or modeling the occurrence of events regardless of the existence of any object responsible of detecting their occurrence, and (4) the type and the amount of information associated with an event. Usually, conducting a sufficient event model involves analyzing the requirements of the intended application and the functionalities that have to be accomplished.

An *event notification*, a *notification* in short, is defined as the datum that reifies an event (Muhl *et al.* 2006). In other words, a notification contains information describing the occurrence of an event. Usually, the actual representation of a notification is a message that contains data values about the event occurrence. The term notification seems a useful abstraction as in most cases a particular occurrence of events is of interest, for example: “Notify me when the temperature is above 35°C”. In many event-based systems, a notification is considered the primitive element in the interaction process as it involves the physical representation of an event. Thus, designing a notification model is investigated instead of modeling the events themselves. Notifications may describe the plain occurrence of events, but they may contain additional information that describes the circumstances of these occurrences. For instance, a notification of a temperature value may contain additional information such as: Sensor ID, Location, Time, and other attributes describing this particular occurrence. Furthermore, this additional information may be considered for security or authorization reasons or simply for the process of routing the generators of notifications. Various notification data models have been investigated in many research works. A common data model used to structure the content attributes of a notification is the name/value pairs (Carzaniga *et al.* 2001). There are other data models also used in this context, such as: semi-structured records (*e.g.*, XML) (Altinel and Franklin 2000, Muhl and Fiege 2001) and object-oriented records (Hayton *et al.* 1996, Eugster *et al.* 2001).

2.3 Producers and Consumers

Producers and *consumers*, also called publishers and subscribers respectively, are two classes of users or software components, generally they are called clients, that interconnect by means of publish/subscribe systems. Software components in this context refer to applications, processes, threads, web services, or other active entities that take roles in the communication flow of the publish/subscribe system. Each component has an identity and location on the network during the interaction with the system. Those components should have the ability to transact, broadcast and/or receive, events or notifications. It should be mentioned that a client can act as a producer and a consumer simultaneously, but here the two types are distinguished in order to understand the role of each of them.

Producers are clients that generate notifications; they publish notifications via the system as those notifications *might be* of interest to other clients. The focus of each producer is bounded by observing the local happenings, including its own state changes, and generates notifications accordingly. The decision of when and what notifications are needed to be published is left to the producer itself. For instance, a temperature sensor, acts as a publisher component, can publish temperature observations on regular time basis, while an officer would publish a notification of a fire incident report once it happens. Producers publish notifications asynchronously in the sense that neither receivers are addressed in the notifications nor are producers aware of other clients on the network. Rather, all published notifications are addressed to the *event notification service*; this is detailed in Section 2.4.

On the other hand, consumers are those clients who receive notifications of their interests. Consumers subscribe their theme of interests in one or a set of events that might be published in the future. Accordingly, they are notified of events that match their interests (*i.e.*, subscriptions). Similar to the producers, consumers perform subscriptions in asynchronous manner and without any previous knowledge about the actual source of notifications or any other clients incorporated in the system. In other words, a subscription action would be performed in order to register an interest on certain types of notifications rather than subscribing to one or set of destinations and thus receiving notifications that are only generated by those predefined sources. Using the previous example of temperature observations, a monitoring station, acts as a subscriber client, would subscribe its interest in temperature observations that exceed 32°C rather than subscribing to a particular source of the observations. Consequently, whenever a temperature observation is published by any temperature sensor and exceeds 32°C, this particular observation will be delivered to the monitoring station in a timely manner. Regarding the subscription operation, two key questions may arise. The first question is regarding the mechanism through which subscribers can register their interests using the publish/subscribe system. The second question concerns the level of expressiveness that the subscribers can have in order to describe their specific interests. The answers of those questions are detailed in two models of subscriptions that are widely used and investigated in the literature: *Topic-based* and *Content-based* models; both models are discussed later in Section 2.5.

2.4 Event-Notification Service

Event-notification service, notification service in short, is the core component of event-based systems. It interposes between publisher from one side and subscriber from the other side. Notification service is responsible for conveying notifications between clients. All the notifications published by producers as well as all the subscriptions issued by consumers are addressed to and handled by the notification service. The service takes the responsibility for delivering each notification to all consumers having registered subscriptions that matched the published notification. The key function of this component is decoupling the producers and the consumers from being responsible for the communication procedure. In other words, the mediating service handles the notifications delivery process on behalf of producers and the evaluation and matching of subscriptions on behalf of consumers. Clients of publish/subscribe system deal with the notification service as a black box (Muhl, *et al.*, 2006).

Four basic operations are provided by the interface of the publish/subscribe model (Cao 2006). When a producer decides to publish a notification, the required attributes are encapsulated in a form of a notification message and the *Publish(event)* operation is called then. A consumer registers his interest in events by calling *Subscribe(sub)* where the *sub* parameter determines what notifications are of interest to the consumer. Accordingly, the notification service stores this subscription and prepares it for later matching with published notification. Similarly, a consumer can terminate an existing subscription by calling *Unsubscribe(sub)* operation. Upon matching published notifications with registered subscriptions, *Notify(event)* operation is performed by the notification service as a call back function in order to propagate notifications among

consumers whose subscriptions are met. Additionally, a fifth operation, *Advertise(ad)*, might be exhibited by which producers can advertise their notifications data structure that will be published in the future. The advertisement operation would serve the notification service to improve the delivery and the matching processes utilizing the expected flows of notifications. Furthermore, this operation can be used to inform subscriber clients about the data structure and content format of future publications. Figure 2.2 shows a high-level design of the publish/subscribe model.

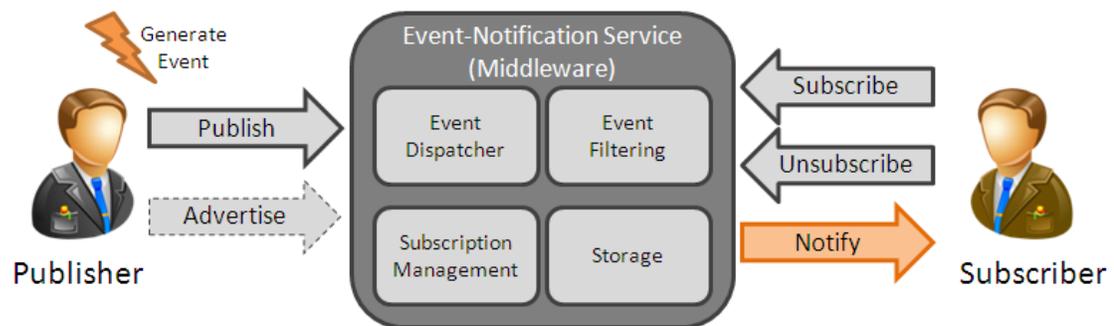


Figure 2.2: Publish/Subscribe Model

The strength of event-based systems relies on the successful operation of the notification service. In fact, most of the computation and the communication processes are happening inside the core of the middleware component. The internal architecture of the notification service is one essential factor that has a large influence on the scalability of the system. In the next sub-sections, two types of notification service architecture are addressed: centralized and distributed architectures, as shown in Figure 2.3.

Thorough investigation and analysis of different architectures of the notification service is beyond the scope of this research. However, the major types of notification service architecture are explained for the sake of providing a general background on

different aspects of this research topic. In the development of the remaining chapters, the centralized architecture of the notification service is considered for simplicity reasons.

This issue is discussed more later on.

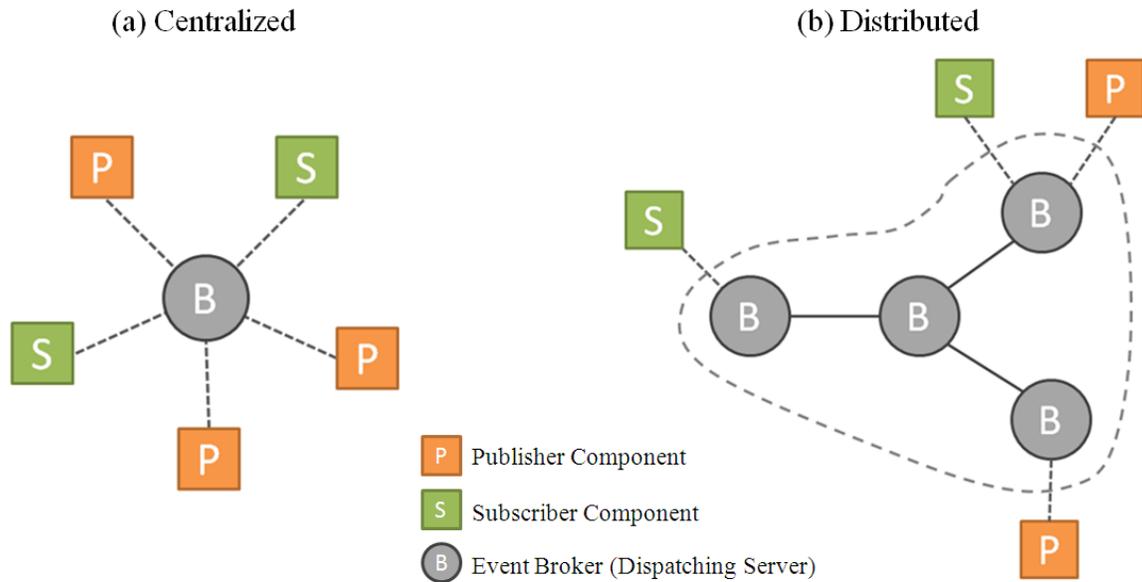


Figure 2.3: (a) Centralized and (b) Distributed architecture of publish/subscribe notification service

2.4.1 Centralized

As shown in Figure 2.3(a), the centralized architecture of the notification service comprises one server component. This central unit is addressed by all the subscription operations as well as all the publications. One principal element of the notification service is the *matching engine*, which is an algorithm used to match the publications (*i.e.*, notifications) with the registered subscriptions. Consequently, it sends the published notifications to the subscribers whose subscriptions are successfully matched. The centralized notification service is introduced as the first generation for the implementation of event-based systems. It is claimed as an easy architecture to deploy

and manage and it is well suited for small-scale event-based applications (Cao 2006).

From a technical perspective, the centralized notification service architecture enables the developers to focus on implementing complex matching algorithms (Fabret *et al.* 2001).

Despite the simplicity of the centralized architecture, it may reduce the scalability of the system and introduce a single point of failure (Cugola and Jacobsen 2002). Moreover, in Cao (2006), several examples were shown where the centralized notification service tends to be an inefficient architecture in cases such as high volume of event publications, high subscriptions diversity and wide users distribution.

2.4.2 Distributed

As shown in Figure 2.3(b), the distributed architecture of the notification service comprises several interconnected server components acting conceptually as a whole as a centralized middleware between users. Usually, these components are called *event brokers* (Pallickara and Fox 2003) or *dispatching servers* (Cugola *et al.* 2001). Each broker serves the local users that are connected to it. Every broker is connected to other brokers within the notification service and thus enabling connectivity to other subnets in the system network. A broker in the notification service acts simultaneously as a publisher and a subscriber on behalf of the publications and the subscriptions that are directed to it from its subnet or the neighbour brokers. The role of a client, publisher or subscriber, ends by accessing the closest brokers in the system network and performing the required publish/subscribe operation. Afterwards, the notification service takes the responsibility of managing this operation throughout its inner brokers. The publish/subscribe operations performed by clients are distributed efficiently among the notification service brokers in a sense to exploit localities in the notification delivery

process (Muhl 2002). Delivering notifications to interested subscribers means routing those notifications starting from publisher components located in certain subnets, throughout the notification service cloud, and reaching potentially numerous subscribers within other distributed subnets. This mechanism is called Event/Notification Routing (Muhl *et al.* 2006, Cao 2006).

The distributed architecture of the notification service seems promising for achieving highly scalable event-based systems over wide-area or large-scale networks (*e.g.*, the Internet). This matter is a primary motivation that led vast research works to realize the distributed architecture of notification services. However, reaching reliable and efficient event-based systems becomes more challenging. Extensive efforts have been spent mainly by the computer science communities in many aspects that concern the distributed notification service architecture, such as: security of publish/subscribe systems (Belokosztolszki *et al.* 2003, Wang *et al.* 2002), fault tolerance in reliable notification delivery and self-stabilizing strategies (Jaeger and Muhl 2005, Pallickara *et al.* 2007), and automatic topology configuration and self-organizing publish/subscribe systems (Jaeger *et al.* 2007, Jaeger 2005). These topics and many others, however, still subjects for future research towards improving the efficiency of distributed notification service in publish/subscribe systems.

2.5 Subscription and Expressiveness

Subscriptions can be seen as boolean-valued filters, which are functions that take a single notification as an input and return *true*, in case of a match found, or *false*, in case of no match found. When a consumer client requires to be notified by certain events, the

consumer would express an interest of notifications by defining a filter function then performing a *Subscribe* operation, as mentioned in Section 2.4, via the publish/subscribe system interface. The operation then is transmitted to the notification service which in turn manages and stores the filter information inside its core. Once a notification is published, the notification service evaluates all the filters of the registered subscriptions and delivers the notification to the consumers whose subscriptions are matched.

Consumers are limited in defining their filters by the provided subscription model or language. How fine-grained filters the consumers can use to express their specific interests is what defines the term *subscription expressiveness* (Carzaniga *et al.* 1999). Increasing the expressiveness of the subscription language means enlarge the domain of the filter model in defining precisely the notifications of interest. In fact, offering rich and more expressive subscription language for consumers is one of the objectives that researchers are seeking to achieve. However, the degree of the expressiveness significantly affects the complexity of the matching algorithm, thus, impacts the efficiency of the delivery of notifications process. Moreover, considering expressive subscriptions would raise a big challenge in terms of the scalability of distributed event-based systems as those features are conflicting (Carzaniga 1998). Therefore, compromising between the aforementioned tradeoffs in deploying event-based systems should be considered.

In notification services, four subscription models (*i.e.*, filtering models) are distinguished in the literature: channel-based, topic-based, type-based, and content-based models (Muhl *et al.* 2006). Particularly, topic-based and content-based are emphasized in

this research as they are the most widely adopted models. The next sub-sections discuss these two models in details.

2.5.1 Topic-based Publish/Subscribe

Topic-based, also called subject-based (Oki *et al.* 1994, TIBCO 1999), is the earliest subscription model adopted in event-based systems. By employing the topic-based mechanism, the notification service predefines a set of subjects or topics by which notifications and subscriptions are classified. Producers are able to publish notifications to any of the predefined topics by annotating each of their notifications with a name string or an ID that refers to a certain topic. From the other side, consumers subscribe their interests in one or a set of topics and thus they receive all the notifications that are published to the topics of interest. A topic can be composed of a set of keywords. Producers and consumers can use those keywords for publishing or subscribing, respectively. The notification service then uses those keywords to classify them into groups.

Topic-based publish/subscribe extends the notion of channels or groups communication, and can leverage the existing group-based multicast communication techniques, such as IP multicast (Floyd *et al.* 1997), in the notifications delivery process. In other words, subscribing to a certain topic can be seen as becoming a member of a group that refers to that topic. Consequently, publishing an event to that topic is viewed as broadcasting this event to the entire topic's members. Topic-based introduces a programming abstraction that maps individual topics to distinct channels in the communication procedure.

The topic-based filtering model is simple to understand and deploy. Industrial solutions, such as Vitria M₃O (Holloway 2008), TIBCO Enterprise Messaging Bus (TIBCO 2000), and USENET News system (Harrison 1995), have adopted the topic-based mechanism. Various enhancements to this mechanism have been proposed in the literature. The use of *hierarchies* in organizing nested topics offers the consumers to perform subscriptions on certain nodes of the topics' trees and thus involving all child topics of those nodes (Eugster *et al.* 2003). The idea of *wildcards* (TIBCO 1999) has been introduced in describing the topics by a set of keywords, thus enabling the consumers to publish or subscribe to several topics that match a given set of keywords. Utilizing an XML model in describing topics schema has been investigated as part of the web notification service standard (Graham *et al.* 2004).

Despite the simplicity and the enhancements added to the topic-based model, the static scheme of a predefined set of topics may restrict the power of subscribers in expressing their specific interests. It should be clear that the actual content information of published notifications has no effect in the matching process and the delivery of those notifications, as it is all about which topics that notifications are published to and subscriptions are registered in. Subscribers have to subscribe to all or none of the notifications published to certain topic. In this case, receiving large amount, irrelevant information would be resulted.

2.5.2 Content-based Publish/Subscribe

Content-based subscription model is the most generic notification selection mechanism (Muhl 2001). This mechanism allows subscribers to express their interest not only in the topic, but also in the actual content information of notifications. In fact, topic-based is

considered a special case of the content-based in which the topic name string of notifications is evaluated against register subscriptions (Cao 2006). Usually using the content-based model, a subscription encapsulates a single or conjunctive predicates (*i.e.*, boolean-valued expressions) that constrain the notifications of interest. Those predicates are evaluated over the content information of notifications and accordingly delivering the matched ones. A simple predicate usually contains an attribute name, a basic comparison operator (*e.g.*, =, >, <, ≥, ≤, LIKE), and a value in the same data type of the attribute name. More complex subscriptions can be formed by combining more than one predicate using logical operators (*e.g.*, AND, OR). For instance, meteorological observations are published by sensors network, while a consumer is interested in those observations where the temperature exceeds 35°C. Thus, the consumer would register a subscription as: TEMPERATURE > 35. Another consumer is interested in certain values of humidity and temperature, thus the consumer would subscribe an interest as: TEMPERATURE > 40 AND HUMIDITY ≤ 0.20.

Enriching the expressiveness of the subscription language by introducing the content-based model adds remarkable value in the paradigm of publish/subscribe systems. Consumers are released from being restricted in set of topics, as in the topic-based model, and allowing them to express their diversified interests at a fine-grained level (Carzaniga and Wolf 2003). Moreover, delivery of irrelevant or uninteresting notifications to consumers is reduced by employing this model, and this is important for parties that have limited processing power devices (Muhl 2002). Nevertheless, realizing a scalable publish/subscribe system as well as conducting an efficient implementation of the matching engine by employing the content-based model is challenging. The content-

based requires more complex notifications matching algorithms because of the need of handling potentially high diversity and large amount of subscriptions; the diversity of registered subscriptions would prevent them from being classified into a finite set of groups, as in the topic-based. This may lead the matching engine to evaluate each single subscription separately against the published notifications, which is an inefficient and time wasting approach. Many researchers have spent their efforts towards developing notification matching algorithms and thus optimizing the matching procedure in the content-based publish/subscribe systems. This is detailed in Section 2.6.

2.6 Notification Matching Algorithms

As mentioned in Section 2.4, the notification service takes the role of connecting producers with consumers by distinguishing the published events or notifications and distributes the information messages to the interested parties. Notification matching is a principal process in any publish/subscribe system. This process is what determines the communication or the information flow between clients. The matching problem can be formulated as identifying the satisfied subscriptions by a given notification. The output from this process is a set of matched subscriptions that refer to one or a set of consumers for which the published notification should be delivered. Usually, the notification service executes the matching process right after a notification is published. Consequently, the consumers interested in this notification are determined then notified by pushing and delivering the published notification to them.

The performance of a publish/subscribe system predominately relies on the efficiency of the matching process or algorithm applied. The faster the matching

algorithm can determine the interested consumers, the minimum the time delay is from publishing the notification until delivering it to interested consumers, hence, the better the performance is. The complexity of the matching process increases relatively with increasing the expressiveness of the subscription language. In the topic-based subscription model, the matching process can be limited by strings matching of the topic names attached with the published notifications against topic names of registered subscriptions. While in the content-based subscription model, the matching process is extended to reach the content-data encapsulated in published notifications. Moreover, the matching becomes more complex as the fine granularity of the filters encapsulated in the subscriptions. Recently, several research works have focused on developing matching algorithms to optimize this process in the content-based model, as it is considered the most generic form in publish/subscribe interaction (Eugster *et al.* 2003, Muhl 2001).

In the next sub-sections, three matching algorithms of the content-based model are presented: the naïve algorithm (also called the *brute force* method), the *counting-based* method, and the *tree-based* method. The last two methods have been extensively recognized in the literature for optimizing the matching process. Several extensions have been developed based on the main idea of these two methods. A thorough investigation of these algorithms and their extensions is not in the main interest of this research. However, the following sub-sections discuss the general concept of these algorithms and how they can optimize the matching process.

2.6.1 Brute Force Method

This is the naïve and the simplest solution of the matching problem. All the subscriptions are evaluated sequentially (*i.e.*, one by one) against a single notification and consequently

the matched subscriptions are determined. Commonly, all the subscriptions are stored in a single table inside the notification service, where each row refers to a single subscription. A matching function takes a single notification and a single subscription row as inputs and associates a boolean output to the subscription; true if the notification matches the subscription and false if the notification does not match the subscription. The matching function iterates all over the subscriptions, which is evaluating one subscription at a time.

Although the brute force matching method is simple to implement, obviously, the performance degrades as the number of subscriptions increases. A single predicate may be evaluated many times as this predicate appears in more than one subscription, and that could be a waste of resources. Furthermore, subscriptions are considered independent elements in this matching method while relations may exist among each other. For instance, if a predicate “TEMPERATURE > 30°C” is evaluated and found as a match, then another predicate “TEMPERATURE > 25°C” should also be a match without even evaluating it. Those dependencies between subscriptions can be exploited to improve the matching process.

2.6.2 Counting-based Algorithm

The focus of the counting-based algorithm (Yan and Garcia-Molina 1994) is to evaluate the predicates contained by the subscriptions rather than evaluating the subscriptions themselves. The counting algorithm separates the process of predicates matching from the process of subscriptions matching. In other words, the matching process of the counting algorithm is divided into two steps: finding the predicates that are matched by a notification, and finding the matched subscriptions whose predicates are satisfied.

The idea of the counting algorithm is to maintain a counter initialized as zero for each subscription. Given a notification, the matching process iterates sequentially over the inner attributes of the notification and matches it with all the predicates. In each iteration, if a predicate is matched with the notification attribute the counter of the associated subscription increases by one. Finally after all the notification attributes are processed, the matched subscriptions are those whose counters are equal to their number of predicates. Figure 2.4 shows an example of the matching procedure using the counting-based algorithm.

All predicates are organized and clustered in one or a set of tables separately from the subscriptions in a way where similarities or covering relationships among predicates together can be exploited (Ashayer *et al.* 2002). In other words, predicates that have same attribute name and the same comparison operator can be clustered in one group. Another technique to manage the predicates is to sort them in a table column where the predicates on a higher level cover other predicates in the lower level. In this way, evaluating similar predicates more than once can be avoided, in contrast to the brute force method, and that would improve the search process for matching subscriptions. Association tables are usually used to maintain the predicates-subscriptions relationships. A more advanced method for organizing the predicates is to maintain the attribute names, the comparison operators, and the values of the predicates in indexing structures and assume those indexes in the predicates matching step in order to look up for matched predicates quickly and efficiently (Carzaniga and Wolf 2003).

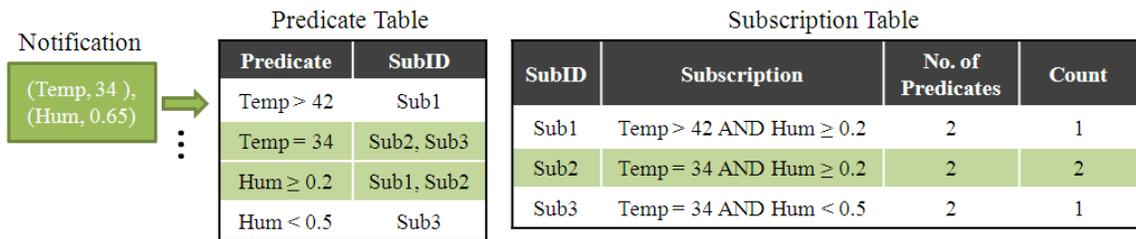


Figure 2.4: An example of content-based matching using Counting algorithm

The counting algorithm assumes subscriptions that are more likely formed by conjunctive predicates. Moreover, predicates have to be pre-processed and organized in a certain scheme before the matching process is actually executed. Thus, insertion and deletion of predicates, upon performing subscribe and unsubscribe operations respectively, should be maintained in a pre-processing step, this would consume some of the processing time. However, the counting algorithm is relatively simple to implement and it is realized in many matching processes of publish/subscribe systems.

2.6.3 Tree-based Algorithm

Similar to the counting-based algorithm discussed in Section 2.6.2, the tree-based algorithm (Aguilera *et al.* 1999) considers each subscription as a conjunction of elementary predicates. In the tree-based algorithm, the subscriptions initially are pre-processed and organized in a form of a multi-level matching tree. The matching tree consists of three elements: non-leaf nodes, leaf nodes and edges. Each non-leaf node contains an attribute test, while the successor edges contain the constants of that test. Leaf nodes, which exist in the bottom level of the tree, represent the subscriptions by which the matching tree is built upon. All the nodes of the matching tree are linked together in the sense of exploiting similarity relationships between the subscriptions' predicates.

Each subscription is explained in a single path starting from the corresponding leaf node in the bottom and ending by the root of the matching tree.

In order to match a given notification, the matching tree is traversed from the root down where the notification attributes are tested against the non-leaf nodes and their corresponding edges. At each level of the matching tree, successfully matched edges are followed until the process hits the leaf nodes. Finally the matched subscriptions are those whose leaf nodes are hit by the matching process. Figure 2.5 shows an example of the matching procedure using the tree-based algorithm.

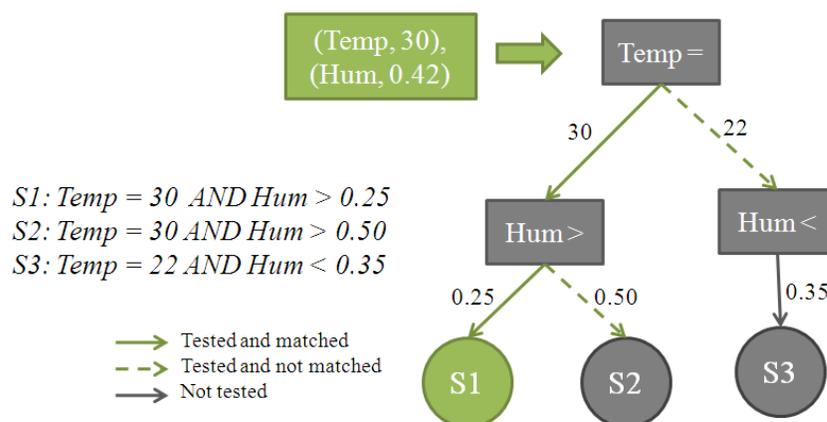


Figure 2.5: An example of content-based matching using Tree-based algorithm

The cost of pre-processing the subscriptions' predicates and maintaining the matching tree is relatively high (Aguilera *et al.* 1999). However, the tree-based algorithm is claimed to be an efficient approach due to the fact that not all the predicates have to be tested, and this would reduce the matching time.

The literature has shown a vast usage of the tree-based mechanism for efficient content-based matching in many research works. The Gryphon (Guruduth Banavar *et al.* 1999, G. Banavar *et al.* 1999), a prototype for event distributed middleware, uses the

tree-based algorithm in content-based matching. In Kale *et al.* (2005), the authors have proven that the tree-based algorithm performs better than the counting-based algorithm according to their model, then they have adopted the tree-based method in developing RAPIDMatch, a content-based matching algorithm. Furthermore, Binary Decision Diagram (BDD) have been exploited for building ordered *binary decision tree* which is used for content-based matching (Campailla *et al.* 2001). It has shown that the BDD method is an efficient matching algorithm even with disjunctive predicates.

2.7 Related Work

The research field of event-based and publish/subscribe systems has been immensely investigated in many directions and aspects especially by the computer science communities. This resulted in many existing systems and research prototypes. Most of the related works that will be mentioned later on in this section have established various advances that can be gained over the traditional request/reply interaction style (see Section 1.2 and Section 1.4). This fact is what caught our attention to realize the publish/subscribe paradigm and serve this research's objectives (see Section 1.5).

This section investigates some of the major research work related to the field of publish/subscribe systems and highlight the scope of this research.

The following research prototypes are considered a major foundation in the world of distributed event-based systems. Scalable Internet Event Notification Architecture (SIENA) (Carzaniga 1998, Carzaniga *et al.* 2001) is one of the early implementations of distributed event-based systems. SIENA targeted the realization of event-based systems at the internet-scale of networks. The main focus of SIENA was on the scalability and the

expressiveness issues and their tradeoffs in the deployment of content-based publish/subscribe over a distributed architecture. HERMES (Pietzuch 2004) is another prototype for large-scale distributed event-based middleware platform. Scalable event dissemination was claimed in this prototype by utilizing peer-to-peer communication techniques for automatic management of its overlay network between event brokers. In events routing, the design of HERMES followed the type-based and the content-based publish/subscribe model. Java Event-based Distributed Infrastructure (JEDI) (Cugola *et al.* 2001), a Java-based object-oriented implementation of a distributed content-based publish/subscribe system. Routing events throughout event dispatchers was based on a hierarchal structure. JEDI prototype has been extended to support mobile computing and dynamic reconfiguration of the network topology as introduced by potentially large number of wireless and non-stationary publishers and subscribers (Cugola and Jacobsen 2002).

Despite the architectural and the functional differences between the aforementioned systems, those pioneers and many other systems that are not mentioned herein have served in conducting a substantial theoretical background for the purposes of this research work. In terms of applicability, however, those systems supported primitive data types of notification, mainly descriptive/textual attributes, while our research tackles the ability of accommodating geospatial type of events and notifications. Handling geospatial events in the publish/subscribe interaction raises the need of dealing with spatial types of interests or subscriptions; this is also not well addressed in the previous publish/subscribe prototypes.

Adapting the spatial semantics of events in publish/subscribe systems has been investigated in few research works. In Bauer & Rothermel (2002), specifications and definitions of the subscription language for handling spatial semantics of events in location-aware applications were proposed. This work is more focused in defining an event and an event composition in the context of spatial locations. Also in Romer & Mattern (2004), an event-based approach for detecting certain states of real world phenomena via Wireless Sensor Networks (WSN) was examined; sensor nodes emit observations in a form of notifications whenever a transition in their local state is happened. Subscribers can issue their interests in a composite of events based on temporal or spatial relationships between the events. Temporal as well as spatial constrains were realized in the subscription language.

In the context of this research, both previous works, however, did not well address the mechanism of handling spatial events in the notification service core. Also, there was no study about the mechanism of how spatial events are matched with subscriptions.

In Chen *et al.* (2003), the authors investigated the issue of accommodating spatially-related events and subscriptions in the context of Location Based Services (LBS) applications. The authors defined a spatial event model as a set of name/value pairs by which mobile users publish their locations using intelligent devices while they are moving. The spatial subscription model was designed to accommodate spatial predicates and allow subscribers to express their spatial interests in events. However, the subscription language only supported two types of spatial predicates: *Within* and *Distance*. Subscribers, who are interested to be notified by certain mobile users, would

use *Within* spatial predicate when they are interested in mobile users once located in predefined zones, and *Distance* spatial predicate when they are interested in mobile users currently located within a predefined distance from the subscribers' origins. The same work was extended to propose the CAMEL project (Ying Chen *et al.* 2003) aiming to develop a spatial publish/subscribe system for LBS. The authors proposed a client-side approach for event matching processing of the subscriptions that have *Within* spatial predicates, the procedure as follows: upon registering the subscribers' interests, the central publish/subscribe server dispatches the *Within* subscriptions to the involved mobile clients. On the client side, the received *Within* subscriptions are spatially evaluated against the acquired location by the client device. In case of a match found, only the matching location event is sent out as a notification to the publish/subscribe server and consequently the client receives messages, such as products promotions, broadcasted for the matched zone area. Handling the spatial matching process in the client sides was claimed to relieve the workload of the publish/subscribe server (*i.e.*, the notification service) as if the location events of the mobile clients are published to the publish/subscribe server regardless and the server in turns takes the charge of matching all the events. However, dispatching the spatial subscriptions to the client sides would cause a burden in the processing load of the clients' mobile devices in case of a large number of subscriptions exist. Further, the system utilizes a predefined and well-known set of zones, rectangles and circles, limiting the subscribers in registering their spatial predicates. The spatial matching engine of the system used a spatial indexing technique, R-tree, to enhance the performance of searching for interested users in certain zones.

The previous work may suit specific applications (e.g. LBS), but it seems that the subscribers are limited in their subscriptions as they have to register their interest only with the predefined zones. Also, the subscribers are still limited by using only two types of spatial constraints, *Within* and *Distance*, which limits the expressiveness of the subscription language to accommodate other spatial relationships (e.g., *Disjoint*, *Overlap*, etc).

In Burcea and Jacobsen (2003), the authors proposed L-ToPSS (Location-aware Toronto Publish/Subscribe System), a publish/subscribe system for LBS applications. Publisher and subscribers were addressed in L-ToPSS as being either stationary (*i.e.*, fixed location) or mobile users (*i.e.*, location is changing over time). Similar to the work presented in Chen *et al.* (2003), publishers publish their locations as events and subscribers subscribe their locations as spatial constraints. The central publish/subscribe server in L-ToPSS is responsible for matching the spatial locations of publishers against the previously registered subscriptions and then notifying the subscribers about publishers who are close by a certain distance. As noticed, all the publications and subscriptions encapsulate spatial coordinates that represent their point location in space, and the geographical distance is the only spatial relationship realized to relate the publications with the subscriptions. The matching engine of L-ToPSS uses the counting-based algorithm for spatial matching processing. This procedure may be considered efficient enough for L-ToPSS as the matching process is limited by testing only the spatial distances constrains. However, accommodating wider range of spatial data types

(*i.e.*, points, lines and polygons) as well as other spatial relationships needs a more effective matching procedure.

Generally, and in spite of the great efforts spent through the research work that was reviewed in this section, there is not enough study found addressing the geospatial semantics of events in the interaction of publish/subscribe systems. Using publish/subscribe interaction and adopting the geospatial semantics of events can be an added-value in a broad range of applications, including: GPS asset tracking, fleet management, natural hazard management, environmental monitoring, and many others. Furthermore, the applications developed using the publish/subscribe model were restricted in minor range, including: LBS and WSN. In the scope or capacity of this research, there has not in the past been enough research implementing the publish/subscribe interaction in the context of emergency management systems. This research aims to address a more generic design of the geospatial publish/subscribe model and employ this design in fire emergency response applications.

2.8 Summary

This chapter defined the major characteristics of an event as an instantaneous occurrence of interest or a change in the state or certain phenomenon. Also, the term notification was defined as a message where its attributes describe the circumstances of an event occurrence. The chapter presented the major components of publish/subscribe systems and role of each component in the interaction framework. Publish/subscribe clients are classified into two types: producers, who publish events, and consumers, who subscribe

their interests in receiving certain events. The notification service is the middleware and the core component of the system. It is responsible of conveying notification messages from producers to consumers in asynchronous and real-time manner. The efficiency of the system largely depends on the performance of notification service in the communication mechanism.

Two commonly used subscription models were presented in the chapter: Topic-based and Content-based. In topic-based, the notification service predefines a set of topics by which producers can publish events and consumers can receive published events. In content-based, consumers can express their specific interests by subscribing filters on the inner (*i.e.*, content) attributes of the future published events. It is considered the most generic form of interaction.

The chapter emphasized the notification matching process, which is a principal function of the notification service to determine the flow of the notification messages. Improving the search mechanism for matched subscriptions leads to accelerate the delivery process of published events thus improving the performance of the whole system. Employing the content-based publish/subscribe needs a more complex matching algorithm. Content attributes should be evaluated against the registered filters (*i.e.*, subscriptions). The chapter explained the brute force method (the naïve matching solution) and two other matching algorithms which are widely used to improve the matching process, namely: counting-based and tree-based. Their main idea is to exploit possible similarities or other relations between the registered subscriptions together in order to avoid unnecessary evaluation of redundant predicates or to reduce the volume of the matching. Clustering of similar or related subscriptions' predicates and using indexes

are some techniques that can be used to amend and speed up the matching process significantly.

Lastly, the chapter reviewed some of the major research works that have been investigated in the fields of event-based and publish/subscribe systems. The chapter presented three research prototypes established for distributed event-based systems development, SIENA, HERMES, and JEDI. The primary scopes of these projects were focused on realizing distributed systems over large-scale of networks. They supported textual or descriptive type of event data models, while this research development attempts to address geospatial semantics of events. The chapter also critiqued some research works that attempted to recognize spatial type of events in LBS and WSN applications. However, the interaction and the spatial event models were limited to their designed applications by incorporating few geometrical types (*e.g.*, points) and a small set of spatial constraints provided in the subscription language. The chapter declared that the aim of this research is to address a more generic design of geospatial publish/subscribe model and employ this design in fire emergency response applications.

Chapter Three: Geospatial-based Publish/Subscribe Interaction

3.1 Introduction

This chapter presents the *Geospatial-based Publish/Subscribe* interaction model, an extended model proposed in this research to accommodate events that are associated with the 2D geographic domain called *geospatial events*.

The main motivation to develop this model is to exploit the publish/subscribe interaction paradigm in transacting geospatial events, therefore, leveraging the situational awareness in fire emergencies.

As shown previously in Section 2.5, the generic subscription model, namely the content-based, has been widely employed as an expressive language allowing consumers of events to specify their detailed interests in the content data of published events. Most of the available implementations of the content-based model support well the attributed (*i.e.*, descriptive or textual values) filters on events by using comparison (*e.g.*, =, >, <) and logical operators (*e.g.*, AND). In the context of geospatial events, these types of filters may not be appropriate for expressing spatial constraints. Spatial filters are needed to offer consumers specifying their interests in a certain geospatial context. Extending the expressiveness of the attributed content-based subscription model is intended by adopting spatial type of constraints. Some of the recent works (see Section 2.7) attempted to study the spatial aspect in the publish/subscribe interaction model. However, they were limited in the representation of spatial events (*i.e.*, only point type) and their work was focused on different applications (*e.g.*, LBS and WSN). Therefore, it is necessary to develop an extended publish/subscribe model to incorporate broader ranges of geospatial event representations and spatial constraints. It was also necessary to utilize this model in the

development of a system prototype, namely a real-time fire emergency response system (see the research objectives in Section 1.5).

The remainder of this chapter explains the proposed geospatial-based publish/subscribe model in details.

3.2 Geospatial Semantics of Events

This section describes how to use publish/subscribe events to represent dynamic geospatial information. It also describes the characteristics of geospatial events conceived in the development of this chapter.

The abstraction of an event in publish/subscribe is defined as an instantaneous occurrence of interest or a state-change of an entity observed and published by producer clients (see Section 2.2). In the context of this research, the concern here is about those events that are associated with geospatial domains; the occurrence is related to a geographical location in space or a real world phenomenon. This type of events is called in this research *geospatial events* (Worboys and Hornsby 2004). Geospatial events can be a useful abstraction to represent sudden occurrences of Earth's phenomena or dynamic state changes of geographic features. GPS locations of vehicles, locations of fire incidents, meteorological sensor observations, and temporal spreading of forest fires are some examples of dynamic geospatial information that can be represented as geospatial events.

The term *geospatial notification* is used in this context to formalize the actual data modeling of geospatial events. In this research, a geospatial notification is defined as a composition of two data components, (1) a spatial component and (2) an attribute

component. The spatial component is used to describe the spatial semantics of an event and considered the predominant data part of a geospatial notification. It contains the geometric shape and location of the event. The spatial component can be represented by one of the following basic spatial feature: a point, a line, or a polygon. The implementation of such data type can be GIS vector or a collection of XY coordinates. The attribute component is a collection of attributes used to assign descriptive information about the circumstances or the properties of an event. Each attribute has a distinct name, a data type, and a data value. Data types include numeric, string, boolean, date/time, and bytes (*e.g.*, Base64 encoding). They can be utilized to attach broad kinds of descriptive information to a geospatial notification. Here, the binary data type supports digital files, including images, documents, media files and other types, to be serialized and encapsulated in the content data of geospatial notifications. For instance, an airborne camera captures thermal images of an active wildfire scene, where in each camera exposure a geospatial notification is initialized. The captured dataset is serialized to a binary format and assigned to a binary type attribute. Finally, the geospatial notification is published to notify other interested clients about the current status of the wildfire event.

Extending the scope of publish/subscribe events to contain geospatial semantics seems promising for numerous web GIS applications. Particularly, applications like emergency response need dynamic geospatial information to flow and be proactively disseminated right away to the right people. However, associating geospatial semantics with events in publish/subscribe systems invokes new challenges to the underlying communication infrastructure. Thus, the proposed model, namely geospatial-based

publish/subscribe, attempts to provide a suitable interaction framework for transacting geospatial events between distributed clients.

3.3 Geospatial Event Clients and the Interaction Flow

After defining the abstraction of geospatial events and geospatial notifications in the previous section, this section explains the interaction workflow between the publish/subscribe components using geospatial events.

Publishers, subscribers, and the notification service middleware are the main publish/subscribe components. In the context of geospatial-based publish/subscribe, publishers are those clients who observe geospatial events, including their geospatial state changes, and consequently publish geospatial notifications as they might be of interest to other clients. Subscribers are those clients who register their interests to be notified about specific geospatial notifications. As geospatial notifications contain attribute and spatial data (see Section 3.2), the subscribers express their interests not only in the content descriptive attribute (using comparison operators discussed in Section 2.5.2), but also they can express their spatial interests in geospatial notifications. This type of subscription is called *geospatial subscription*. For instance, considering emergency assets publish their current GPS locations as geospatial notifications, an officer can subscribe his interest to be notified about any asset within proximity of 1000m from a reported incident location. To define such a subscription, the subscription language should support expressions used to assign spatial constraints on the published geospatial notifications. Using the previous examples, the officer would use the “*Contain*” spatial operator in his subscription expression to receive those emergency assets’ points located within a 1000m

buffer zone centered at the incident point location. This is detailed later on in Section 3.5.2.

The notification service mediates between geospatial event clients. It handles the published geospatial notifications, matches them with the registered geospatial subscriptions, and finally delivers those notifications to the matched subscribers. The geospatial notifications that satisfy the subscribers' interests should be delivered to them in timely manner. The middleware service provides predesigned interfaces for publisher and subscriber clients granting them the performance of publish and subscribe operations, respectively, for geospatial notifications. The interaction operations are previously shown in Figure 2.2.

3.4 Design Considerations

There are some considerations that must be taken into account while designing the geospatial-based publish/subscribe model. The remaining sections of this chapter serve as a detailed analysis of the model based on the following considerations.

Regarding the architecture of the notification service middleware, the centralized architecture (see Section 2.4.1) is adopted in designing the geospatial-based publish/subscribe model. Employing the distributed architecture may offer a more scalable system in terms of integrating clients over a wide-scale of networks (*i.e.*, the Internet). However, it increases the complexity of the design and the implementation of the system. Also, it invokes more research challenges regarding the development of dispatching algorithms of events and maintaining the connectivity between the notification service's brokers. Studying this type of architecture is outside the scope of

this research. In this research, the focus is on designing data models and matching algorithm cope with the geospatial nature of events rather than focusing on the distributed architecture of the system. Realizing the distributed architecture of the notification service is addressed as a part of the future works for this research (see Section 6.3).

As mentioned in Section 2.3, publish/subscribe clients act as subscribers to events, publishers of events, or both roles at the same time. Subscriber clients usually use a software application to perform geospatial subscriptions via the provided notification service interface. The software application provides tools for the subscriber suitable to connect to the notification service unit and perform geospatial subscriptions. Publisher clients may refer to users who utilize a software application to publish geospatial events or to an electronic device (*e.g.*, GPS, sensor, camera, etc) that has the capability to observe and publish geospatial events through the system network. Publishers also should communicate with the notification service unit via the interfaces provided for them. All the clients are unknown to each other; clients interact with the notification service simultaneously and regardless of other clients existing on the network. The only visible component for all the clients is the notification service.

A reliable communication network (*i.e.*, TCP/IP) is assumed underlying between clients and the notification service middleware.

The development of the proposed geospatial-based publish/subscribe model is conducted in three phases. The first is designing the geospatial notification data model by which producer clients publish geospatial events/notifications. The second is designing the geospatial subscription data model which is used by consumer clients to subscribe their interests in geospatial notifications. The final phase is developing the matching

method to evaluate published geospatial notifications with registered geospatial subscriptions and finding the matched subscribers. Section 3.5 introduces the first two phases and Section 3.6 discusses the final phase.

3.5 Geospatial-based Publish/Subscribe Data Models

This section proposes two data models: the Geospatial Notification Data Model and the Geospatial Subscription Data Model. Clients are supposed to utilize these data models in their publications or subscriptions, respectively. Section 3.5.1 introduces the content data model of geospatial notifications by which publishers can perform publish operations, and Section 3.5.2 introduces the geospatial subscription language model by which subscribers can perform subscribe operations for geospatial notifications.

3.5.1 Geospatial Notification Data Model

A geospatial event is described by a geospatial notification which in turns comprises a set of name/value pairs. Each pair specifies a single attribute of the associated geospatial event. The abstraction of name/value pairs is similar to the data structure of records; a record in a table consists of several cells where each cell has a field name and a value in the same data type of the field. Formally, a geospatial notification n_g is formed by a set of nonempty attributes (a_1, a_2, \dots, a_n) , where each a_i is a name/value pair (n_i, v_i) . Each name n_i is assumed unique in the attributes set and has a single data type associated with it. The value v_i should be assigned according to the data type of the name n_i . The data types supported for constructing the name/value pairs are similar to the SQL data types, briefly: *string*, *integer*, *float*, *boolean*, *date/time*, and *byte-array*. In addition to that, *geometry* data types, including: *GeometryPoint*, *GeometryPolyline*, *GeometryPolygon*,

GeometryMPoint, *GeometryMPolyline*, and *GeometryMPolygon*, are added to the collection in order to accommodate the spatial semantics when generating geospatial notifications. This is detailed in the following.

As mentioned in Section 3.2, a geospatial notification has two data components: an attribute component and a spatial component. The attribute component can be formatted by a set of name/value pairs with traditional data types, such as string, integer, and date/time. On the other hand, the spatial component is defined by adding a name/value pair with a geometry data type. In this context, the geometry data type is used to assign the shape and location of the geographic feature that corresponds to the occurred geospatial event. The geometries of simple geographic features, as shown in Figure 3.1, are supported in the geospatial notification model. The value of the geometry data type is assigned as a single or conjunction(s) of XY coordinates which correspond to the actual geometry of the spatial component. In case of a *point* or *multi-point* geometry type, the coordinates correspond to the points' locations, and in cases of *polyline*, *multi-polyline*, *polygon*, and *multi-polygon* geometry types, the coordinates correspond to the vertices' locations that form those geometries. Table 3.1 shows examples of the geometry data format for simple geographic features. The geometries of simple geographic features can cover a large variety of the spatial component representation in generating geospatial notifications. However, complex spatial representations, such as a polygon with a hole inside, need a more complex topological structure in formatting the geometry data value. Using conjunction of coordinates in this case would not be sufficient.

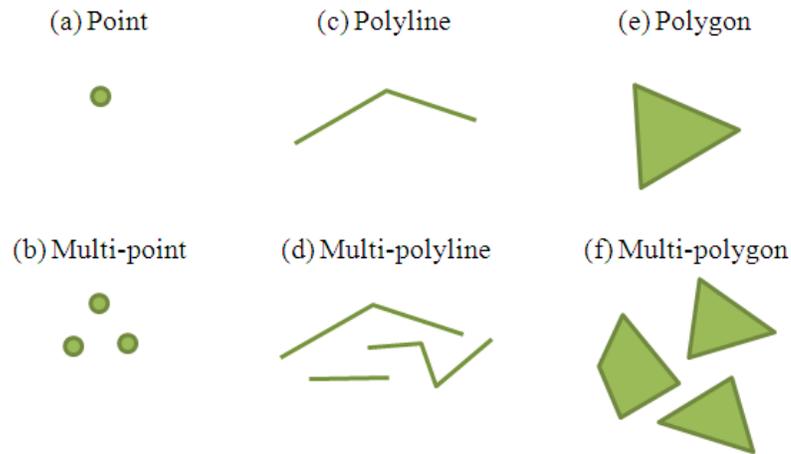


Figure 3.1: Geometrical representations of simple geographic features

Other types of data can be part of the geospatial notifications data contents, specifically computer files, including: images, documents, and media files. Name/value pairs with byte-array data types are employed for this regard. Byte-array data types handle any arbitrary information in binary format. Therefore, the required files are converted first to binary data then assigned to byte-array name/value pairs.

Table 3.1: Examples of name/value pair formatting with a geometry data type

Geometry Type	Name/Value Format
Point, Multi-point	{ <i>GeometryPoint</i> , (10 10)}, { <i>GeometryMPoint</i> , (10 10); (15 10); (13 12)}
Polyline, Multi-polyline	{ <i>GeometryPolyline</i> , (10 10, 12 10, 15 12)}, { <i>GeometryMPolyline</i> , (10 10, 12 10, 15 12); (12 10, 12 15); (11 11, 8 10, 13 12)}
Polygon Multi-polygon	{ <i>GeometryPolygon</i> , (12 14, 10 10, 13 15)}, { <i>GeometryMPolygon</i> , (12 14, 10 10, 13 15); (13 11, 10 15, 16 18); (11 12, 15 12, 12 14, 9 8)}

3.5.2 Geospatial Subscription Data Model

Geospatial subscriptions can be seen as filters or boolean-valued functions that evaluate whether or not published geospatial notifications match the defined constraints. One can

understand geospatial subscriptions as database queries and the evaluation process is nothing more than selecting the rows that match those queries.

Generally, the design of the subscription model should follow the underlying notification data model. Having descriptive and spatial content data in the published geospatial notification, as discussed in Section 3.5.1, imposes incorporating types of predicates capable of expressing specific interests in the descriptive information as well as in the spatial features of geospatial notifications. Therefore, two types of predicates are supported in the geospatial subscription language model: *attribute predicates (AP)* and *spatial predicates (SP)*.

Attribute predicates are utilized to constrain the selection of geospatial notifications according to their content descriptive data. An attribute predicate AP_i specifies an attribute name string, a comparison operator, and a comparison value (e.g., “*HUMIDITY*” > 0.30). The attribute string name entails the process of searching for an attribute (i.e., a name/value pair) within the content data of the evaluated geospatial notification that has the same name string as the predicate’s name string. In case of a name/value pair existing in the geospatial notification content data, the value of this pair is evaluated against the comparison value specified by the predicate using the comparison operator. The result of this evaluation process is either *true* in case of a match or *false* otherwise. If the attribute name string does not exist in the geospatial notification content data, the geospatial notification is considered not matching the subscription. The comparison operators supported herein are =, >, <, ≥, and ≤ for numeric values, and = and LIKE for string values. The comparison values should be consistent with the assigned comparison operator.

Spatial predicates are introduced in the subscription language model offering the subscribers more expressiveness to define their interests in geospatial notifications that satisfy certain spatial constraints. A spatial predicate SP_i is defined as triple parameters: base geometry G_p , a spatial operator SOp , and a buffer value $buff$, i.e. $SP_i = (G_{pi}, SOp_i, buff_i)$. The following is a list of the supported values for each parameter.

- Base geometry G_b : simple features of *Point*, *Polyline*, *Polygon*, *Multi-point*, *Multi-polyline*, and *Multi-polygon*.
- Spatial operator SOp : *Contain*, *Disjoint*, *Cross*, *Touch*, *Overlap*, and *Within*.
- Buffer value $buff$ (optional): any numeric value of type float.

The base geometry G_{pi} is one of the simple geometries listed above. The data structure of the base geometry is similar to the data structure in defining the spatial components of geospatial notifications, discussed in Section 3.5.1. The subscriber usually defines type, shape, and location of the base geometry by manual drawing on the screen or selecting existing geographic features using a GIS map; these functionalities should be provided by the software application that the subscriber is using. Spatial operators SOp_i are equivalent to the role of the comparison operators as mentioned previously in the attribute predicates. However, instead of comparing numeric or string values, spatial operators are used for spatial comparison between geometries. The spatial operators listed above have been recognized as standards of topological relationships between simple geographic features (Schneider and Behr 2006). Spatial operators take two geometries as an input to determine if a specific spatial relationship exists between the two geometries. Usually, the first geometry is called *base geometry* (e.g., the geospatial subscription geometry) and the second is called *comparison geometry* (e.g., the geospatial notification

geometry). In this context, the base geometry is the one defined by the spatial predicate and the comparison geometry is the spatial component of the geospatial notification. A complete definitions list of all the possible topological relationships between simple features, i.e. point, polyline and polygon, are shown in Appendix A. The output from this comparison is a boolean value; *true* if the comparison meets the function criteria, and *false* otherwise. For instance, to evaluate if the position of an emergency truck (point object) is located inside a county region (polygon object), the spatial operator *contain* is performed as “ $(Polygon_{county}) \text{ contain } (Point_{truck})$ ”. The output here is either *true* if the emergency truck is inside the county area, or *false* otherwise. The buffer value *buff* is an optional numeric value of type float assigned if a zone area is required around the base geometry to be included in the evaluation process.

Based on the previous definitions of attribute and spatial predicates, the data structure of a single geospatial subscription Sub_i is formulated as,

$$Sub_i = [SP, AP_i(s)]$$

In this definition, a single subscribe operation Sub_i can encapsulate two Boolean functions: a maximum of one spatial predicate SP and one or more attribute predicates $AP_i(s)$ conjugated by logical operators. Assigning the two functions means that any geospatial notification has to satisfy both functions at the same time in order to be considered as a match. Otherwise, the subscriber can assign the key word “*NULL*” to any one of the functions stating that the subscriber is not interested in constraining the required geospatial notifications by this function.

As discussed in the content-based publish/subscribe section of this thesis (see Section 2.5.2), the subscription language supports various forms of attribute predicates to

filter the notifications of interest based on their content attribute data. In the geospatial subscription language proposed herein, the addition of the spatial predicates extends the expressiveness by involving spatial type of constraints in specifying the geospatial notifications of interest. The definition proposed above for geospatial subscriptions allows subscribers to express fine-grained level of interests. Table 3.2 shows some examples of spatial types of interests in geospatial notifications and their respective expressions of spatial predicates.

Table 3.2: Examples of spatial interests' expressions and their respective spatial predicates formation

Spatial Interest	Geospatial Notification		Respective Spatial Predicate		
	Source Desc.	Comparison Geometry	Base Geometry G_b	Spatial Operator SOp	Buffer Value $buff$ (m)
<i>Notify me of any vehicle is within a municipality boundary</i>	Vehicles current positions	Point 	Polygon 	Contain	0
<i>Notify me of any fire incident happens within 5km of my center</i>	Fire incidents reporters	Point 	Point 	Contain	5000
<i>Notify me of any fire spreading exists in the neighborhood area</i>	Temporal fire spreading area	Polygon 	Polygon 	Overlap	0
<i>Notify me of any police car far from the highway road by 2km</i>	Police cars current positions	Point 	Polyline 	Disjoint	2000

3.6 Geospatial Notification Matching

Notification matching is a prominent process executed by the notification service. The results from this process determine the flow of information between the interacting clients. In this section, the matching process in the context of the geospatial-based publish/subscribe is investigated in details.

3.6.1 Matching Problem

The geospatial notification matching problem can be formulated as the following. Let's assume a geospatial notification n_g is published throughout the publish/subscribe system. A set of subscriptions $Sub_1, Sub_2, \dots, Sub_n$ have been previously registered where each Sub_i defines a filter on geospatial notifications that are of interests. The matching process determines a subset of subscriptions where each Sub_i in the subset matches the geospatial notification n_g . According to the geospatial subscription model discussed in Section 3.5.2, each Sub_i comprises two boolean functions, SP and AP_i (i.e., $Sub_i[SP, AP_i]$), of a spatial constraint and an attribute constraint respectively. Both functions take the geospatial notification n_g as input, evaluate n_g according to the assigned conditions, and generate a boolean value as an output. The Sub_i matches n_g if the output boolean values from both functions are *true* (i.e., $Sub_i[true, true]$), whereas the Sub_i does not match n_g otherwise (i.e., $Sub_i[true, false]$, $Sub_i[false, true]$, or $Sub_i[false, false]$). In cases where the key word "NULL" is assigned to any one of the subscription's functions, the output of the associated function is considered *true* without evaluating n_g .

To understand the matching process of geospatial notifications, let's assume the following example of matching a single geospatial notification against a single geospatial subscription. A vehicle mounted by a GPS device publishes geospatial notifications of its current position to the publish/subscribe system on a regular basis. The type of information encapsulated in every geospatial notification includes: ID, X and Y coordinates for the GPS position, current speed, and time of the acquired position. A user is interested in monitoring the movements of vehicles and requires to be notified if any of the vehicles pass through the geographic boundary of a certain municipality region

exceeding 80 km/hr. This theme of interest requires formulating two constraints: a spatial predicate which states that the municipality region (polygon geometry) should contain the vehicle's current position (point geometry), and an attribute predicate which indicates that the vehicle's speed should exceed 80 km/hr. Accordingly, the user issues the following subscription: *Subscribe* $\{ \{ (GeometryPolygon, \dots), Contain, 0 \}, \{ "SPEED" > 80 \} \}$, where the *GeometryPolygon*, *Contain*, and the value *0* are representing the vertices coordinates of the municipality polygon geometry, the required spatial operator, and the buffer zone value, respectively. "*SPEED > 80*" is the attribute constraint required over the vehicles' geospatial notifications. At a certain moment, the following geospatial notification is published by a vehicle: *Publish* $\{ \{ GeometryPoint, \dots \}, \{ ID, 4 \}, \{ SPEED, 87 \}, \{ PosTIME, "12/9/2008 1:00:06 pm" \} \}$. Subsequently, the notification service matches the published geospatial notification by the vehicle against the registered subscription by the user. As the user's subscription contains spatial and attribute constraints, the vehicle's geospatial notification should be evaluated against both constraints. In other words, the matching process should answer the following questions:

1. Does $GeometryPolygon_{sub}$ with a buffer zone of 0 Contain $GeometryPoint_{pub}$?
2. Is the vehicle's speed value $SPEED_{pub}$ larger than ($>$) $SPEED_{sub}$ speed value?

Then, the notification service will notify the user about the vehicle's geospatial notification if the output from both tests mentioned above is *true*. Otherwise, the vehicle's geospatial notification will be discarded and the user will not be notified.

The aforementioned matching procedure seems quite simple when evaluating a single geospatial notification against a single geospatial subscription. However, the matching process becomes more complicated and expensive as potentially thousands or

even millions of geospatial subscriptions will be involved in the matching process. The challenges of the matching process are discussed in the next section.

3.6.2 Matching Challenges

In the publish/subscribe interaction model, hypothetically, published notifications should be pushed to interested subscribers in timely manner. However, involving a large amount of interest (*i.e.*, subscriptions) in the matching process would prevent the real-time delivery of notification as the process would consume more time. Thus, optimizing the notification matching process is essentially needed. The optimal goal here is to reduce time latency consumed by the matching process. Achieving this means rapid streaming of geospatial notifications to interested clients thus improving the situational awareness in time-sensitive situations.

There are two main issues that would potentially reduce the efficiency of the matching process of geospatial notifications: (1) a large number of registered geospatial subscriptions and (2) high diversity of interests. The former issue entails searching for matching subscriptions among a large dataset of registered geospatial subscriptions, which would increase the processing time and delay the notification delivery. The latter issue would cause the matching process to evaluate each subscription separately when the constraints of geospatial subscriptions are different, which also would consume the processing time. The naïve solution of the geospatial notifications matching, the brute force method, is to store all the subscriptions in one table inside the notification service and conduct the matching procedure on a one by one basis (*i.e.*, evaluating the published geospatial notification against one subscription at a time). Then if a match is found, the geospatial notification is delivered to the associated subscriber. Figure 3.2 illustrates the

geospatial notification matching process using the brute force method. This naïve method simply solves the matching problem by evaluating geospatial notification with all the subscriptions sequentially. However, it is an evidently inefficient and time consuming process when the number of registered subscriptions increases.

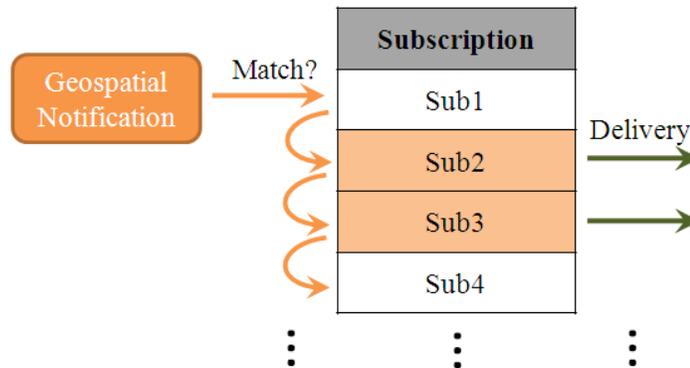


Figure 3.2: Geospatial notification matching using the Brute Force method

As mentioned earlier, to grant a real-time or near real-time delivery of published geospatial notifications, searching for matched subscribers amongst the whole set of issued subscriptions should be performed efficiently and with the minimum processing time possible. To achieve this goal along with the aforementioned challenges, the subscriptions involved in the matching process should be structured inside the notification service unit in a way to speed up the searching process.

The counting-based and tree-based content-based matching algorithms, investigated in Section 2.6.2 and Section 2.6.3 respectively, conceptualize the idea of transforming the registered subscriptions into better data structure and thus enhancing the matching process speed. Data structuring of subscriptions is accomplished by exposing the internal predicates of the subscriptions and exploiting the relationships possible between them to reduce evaluating redundant or unnecessary predicates. Both algorithms

have proven their efficiency in dealing with attributed predicates. In the geospatial publish/subscribe context, however, both algorithms are not generally applicable for evaluating geospatial subscriptions as they contain spatial type queries. Thus, it was necessary to develop an alternative matching approach that can deal with evaluating spatial queries (*i.e.*, constraints). In this research, an efficient matching approach is proposed which is suitable for matching geospatial notifications with geospatial subscriptions efficiently. This approach is detailed in the next section.

3.6.3 Improving the Matching Process

This section describes the proposed approach for improving the geospatial notification matching process.

Before introducing the proposed matching approach, the concept of spatial indexing to enhance spatial query processing is briefly introduced. Then, an explanation is provided on how spatial indexes can be used for indexing the geospatial subscriptions data and thus improving the matching process. Finally, the proposed approach for geospatial notifications matching in the context of the geospatial-based publish/subscribe model is described.

3.6.3.1 Spatial Data Indexing

In general, databases rely on the index data structure for quick access of data requested by a certain query, and that is in contrast with the traditional way of sequentially scanning the data entries which is considered a time-consuming and expensive process. Spatial indexing enhances the processing of spatial queries and speed up retrieving the data of the required spatial objects. The fundamental concept of spatial indexing is the use of approximations (Shekhar and Chawla 2003). Spatial objects are structured inside the

index using a simple approximation of the geometries. The prime geometry approximation used is the smallest bounding rectangle of the objects' geometry, called *minimal bounding box (mbb)* (Rigaux *et al.* 2002). Using a spatial index, the processing of an operation that involves a spatial predicate on a collection of spatial objects is performed in two steps: the *filter step*; selecting all the spatial objects whose *mbb* satisfies the spatial predicate. This step returns a superset of candidates of spatial objects. In the second step, called the *refinement step*, the exact geometries of the spatial objects in the superset are tested against the spatial predicate. This key procedure is behind querying and retrieving spatial data quickly and efficiently. The cost of evaluating complex geometries of the spatial objects is saved by evaluating their approximations, i.e. *mbb*, which is much easier process.

The most commonly used spatial indexing techniques are: Grid-indexing (Rigaux *et al.* 2002), Quad-tree (Samet 2006, Berg *et al.* 2008), and R-tree (Manolopoulos *et al.* 2005). Grid-index and Quad-tree are conceptually similar; both of them divide the 2D space into rectangular partitions. Each partition is a key reference for its fully or partially contained spatial objects. The construction and data structure of both indexes are easy and simple, that entails also the simplicity in reconstructing those indexes while insertion or deletion of spatial objects. On the other hand, the performance of both techniques is relatively less than the R-tree technique with huge and highly-clustered spatial data. R-tree and its extensions have proven their high efficiency for structuring spatial datasets in different operations and under many circumstances. Nevertheless, the clustering time in building this index is relatively expensive. The performance of an index structure may vary under substantial amounts of insertion and deletion operations of spatial objects. In

this regard, Quad-tree and R-tree may perform better as their structures are dynamic and can tolerate updates on existing spatial objects. It is generally known that there is no single indexing method with the capacity to give the best performance under all circumstances. All of them have inherent strengths and weaknesses. The choice of a specific index depends on the nature of the underlying spatial data in terms of the spatial distribution, the size of the dataset, updating the existing spatial data and other factors.

3.6.3.2 Indexing of Geospatial Subscriptions

After introducing the idea of spatial indexing and how it can be used for efficient data retrieval from spatial databases, this section explains the matching procedure of geospatial notifications from a different perspective, and how spatial indexing can enhance the matching process by indexing the registered geospatial subscriptions.

The focus here is on the spatial predicates part encapsulated within the geospatial subscriptions (see Section 3.5.2). The abstraction of geospatial subscriptions is similar to the notion of spatial query processing; a spatial query defines a geometry and spatial relationship (*e.g.*, *Contain*, *Disjoint*, *Overlap*, etc) and selects a set of spatial features that satisfy the spatial relationship condition. In our case, geospatial subscriptions are previously stored in the notification service database. When a published geospatial notification attains the notification service tier, the central matching engine starts evaluating the notification spatial object (*i.e.*, spatial data) with the previously stored geospatial subscriptions (*i.e.*, spatial queries). Let's assume that all the registered geospatial subscriptions are rectangular polygons and have the same spatial constraint type, for example *Contain*. Thus traditionally, the following spatial query “(the *geometry* of the geospatial subscription) *Contains* (the *geometry* of the geospatial notification)” is

executed many times, equal to the number of the geospatial subscriptions, to select the matched geospatial subscriptions. Instead, the matched geospatial subscriptions set can be found reversely by searching for all the geospatial subscriptions, at once, where the geospatial notification geometry is located within. Notice that both procedures give the same results, namely the matched geospatial subscriptions set. However, the latter procedure can be conducted by processing only *one* spatial query, namely “(the *geometry* of the geospatial notification) *Within* (the *geometries* of the geospatial subscriptions)”. Here the roles of geospatial notification and geospatial subscriptions are reversed; instead of taking geospatial notifications as spatial data and geospatial subscriptions as spatial queries, geospatial subscriptions are considered as spatial data and geospatial notifications as spatial queries. This idea raises the need for enhancing the data storage structure of geospatial subscriptions in order to efficiently retrieve the required subscriptions from the database, which can be achieved by using *spatial indexes*.

The aforementioned idea of reversing the roles of notifications and subscriptions has been proposed by several works in the context of LBS applications (Kalashnikov *et al.* 2002, Wu *et al.* 2004). The notifications represented the current positions of moving objects and the subscriptions represented rectangular spatial queries requesting all moving objects located inside the queries boundaries (*i.e.*, *Contain* constraint). They argued that it is more efficient to index the continual range queries (*i.e.*, subscriptions) rather than indexing moving objects data (*i.e.*, events or notifications). That is because objects may continuously move irregularly and in unpredictable ways, which makes it difficult to maintain effective indexes. Rather, their attempts were to build spatial indexes

on range queries to quickly select the range queries that contain a given object. This approach is called *query indexing*.

For improving the matching process in the context of this research work, the idea of indexing the geospatial subscriptions is adopted similarly to the work presented earlier. However, several types of spatial queries, other than *Contain* constraint, are supported in the geospatial subscription language (see Section 3.5.2) and should be accommodated in the matching process. Thus, further considerations have to be taken into account while designing the matching approach. The matching approach proposed in this research is discussed in the next section.

3.6.3.3 The Proposed Geospatial Notification Matching Approach

The proposed approach for matching published geospatial notifications with registered geospatial subscriptions is performed in two phases: (1) a pre-processing phase and (2) a matching phase. The matching engine initially pre-processes geospatial subscriptions into a data structure that allows fast matching. This phase is conducted prior to the actual matching with published geospatial notifications or after geospatial subscriptions are issues by subscribers. At a later stage and when a geospatial notification is published, the matching engine uses the prepared data structure of geospatial subscriptions and conducts the matching process searching for the matched ones. The following explains each phase in more details.

A geospatial subscription contains a base geometry, a spatial operator, and optionally a buffer value (see Section 3.5.2). The spatial operator and the buffer value parameters vary from one subscription to another. In the pre-processing phase, the subscriptions are transformed into homogenous groups of subscriptions, where all

geospatial subscriptions within each group have the similar spatial operator and a *zero* buffer value (*i.e.*, no further buffering required on the base geometry). Geospatial subscriptions are clustered into feature classes according to their assigned spatial operators. For instance, all geospatial subscriptions assigned the *Contain* spatial operator are clustered in a single feature class, where as all geospatial subscriptions assigned the *Overlap* spatial operator are clustered in another feature class. As the geospatial subscription language supports six types of spatial operators (see Section 3.5.2), six feature classes are pre-created the output from the clustering process described above will be at a maximum of six feature classes of geospatial subscriptions. In each feature class, afterwards, the base geometry of each geospatial subscription is buffered according to the buffer value assigned. The new geometry resulted from the buffer processing is reassigned to the associated geospatial subscription as the new base geometry. Two examples of the buffering process are depicted in Figure 3.3. Notice that the buffering process can be skipped if the buffer value is originally assigned zero. The final step of the pre-processing phase is structuring the subscriptions' feature classes into spatial indexes. One spatial index is built for each of the subscriptions' feature classes.

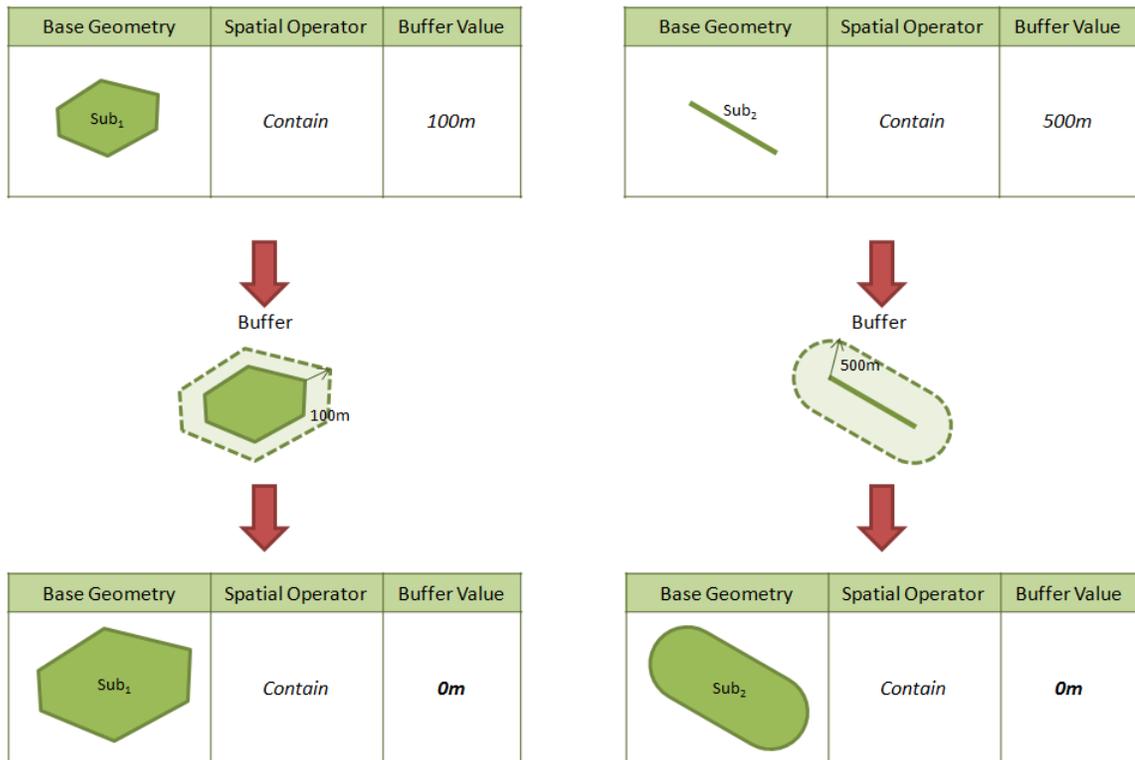


Figure 3.3: An example of geospatial subscriptions clustering process

At the pre-processing stage, geospatial subscriptions are pre-processed into homogeneous feature classes and structured in spatial indexes. In the matching phase, as soon as a geospatial notification n_g reaches the notification service, the matching engine uses the prepared spatial indexes of registered geospatial subscriptions and executes the matching process. The process takes the geometry (*i.e.*, the spatial component) of the geospatial notification and uses it as a spatial query to retrieve the matched set of geospatial subscriptions from the stored feature classes. As there are potentially six subscription feature classes, six spatial queries are executed. The formulations of these queries are as follows:

- (the *geometry* of n_g) **Within** (the *geometries* of *Contain* feature class).

- (the *geometry* of n_g) **Contains** (the *geometries* of *Within* feature class).
- (the *geometry* of n_g) **Disjoints** (the *geometries* of *Disjoint* feature class).
- (the *geometry* of n_g) **Crosses** (the *geometries* of *Cross* feature class).
- (the *geometry* of n_g) **Touches** (the *geometries* of *Touch* feature class).
- (the *geometry* of n_g) **Overlaps** (the *geometries* of *Overlap* feature class).

Notice that in the first two feature classes, *Contain* and *Within* feature classes, the spatial operators *Within* and *Contains* are used respectively as the spatial relationship is reversed by using the geospatial notification n_g as a query. Whereas, in the remaining four feature classes, *Disjoint*, *Cross*, *Touch*, and *Overlap*, similar spatial relationship, *Disjoints*, *Crosses*, *Touches*, and *Overlaps* are used in their respective queries as the function criteria does not change with reversing the base geometry and the comparison geometry. Lastly, the geospatial subscription features selected by executing the above spatial queries are considered matches to the geospatial notification n_g .

As mentioned in the proposed geospatial subscription language (see Section 3.5.2), geospatial subscriptions may also encapsulate attribute filters. The aforementioned matching procedure only evaluates the spatial predicates of geospatial subscriptions. Thus, to conduct a complete matching process, the spatially matched subscriptions set resulted from the above process are evaluated with the content-attributes of the geospatial notification. Those subscriptions that pass through the attribute evaluation process completely match the geospatial subscription.

The proposed matching approach described above is intended to be implemented by the matching engine of the notification service component to accelerate the

dispatching process of published geospatial notifications. This will be explained later on in this document (Section 4.2.3).

3.7 Delivery of Geospatial Notifications

So far, the previous section discussed how the middleware service matches published geospatial notifications with registered geospatial subscriptions and consequently finds the interested subscriber clients whose subscriptions are satisfied. The next stage is to deliver or push the information to the interested subscribers' applications. Technically, the notification service executes the output operation *Notify(n_g)* (see Figure 2.2) encapsulating the published geospatial notification within this operation and addressing the matched subscribers on the delivery process. This section underlines some issues the subscribers should be aware of in receiving geospatial notifications.

In the proposed geospatial-based publish/subscribe interaction, subscriber clients may use mapping applications, where a map containing basic geographic data layers is a primary part of those applications. As the received geospatial notifications contain geospatial data, subscribers generally need to interactively visualize the received notifications on a GIS map and perform various spatial analyses based on the received data, such as overlaying, proximity, and network analyses, or potentially conducting responsive processes and actions upon receiving particular types of geospatial notifications. To this end, there are important issues that the interacting components should be aware of regarding effective handling of delivered geospatial notifications. First and foremost, the content-data schema of potentially published geospatial notifications should be well-known by the interested subscribers (i.e. receivers'

applications). In this manner, the core processing of the subscribers applications can be customized in a way to handle, parse, and manipulate the received notifications data according to the requirements. For instance, a user requires overlying the received points' notifications of assets' current positions automatically in a GIS map and with a certain symbology style. Another user needs an automatic storage of the received notifications of sensors' temperature observations in a certain format inside a database. Second, the coordinate referencing system by which the spatial component of geospatial notifications is created should be known, too. The subscribers then can georeference the received geospatial notifications and perform an appropriate coordinate transformation processing required to be consistent with the coordinate system of their GIS datasets.

There are two methods that can be applied to enhance adopting the above issues in receiving geospatial notifications. First, establishing standards for the content data structure of geospatial notifications, therefore those standards will be recognized and followed in performing publications and subscriptions operations. This can be achieved by predefining well-structured name/value pairs, including the required attribute names, their data types, and other necessary descriptions according to the topic or theme of geospatial notifications. For instance, geospatial notifications for meteorological observations should contain the name/value pairs specified in Table 3.3. The second method is by employing the advertisement technique, using *Advertise(ad)* operation as discussed in Section 2.4, where publishers advertise their notifications data structure and thus subscribers can expect the content data schema of future publications. The latter method seems to be more sophisticated and effective as publishers are not restricted to limited standards of data structures. It needs more development to adopt this technique in

the system. In our research work, however, the first method, namely establishing well-defined content-data structures, is assumed in the implementation of the Real-Time Fire Emergency Response System (RFERS) prototype (detailed in Chapter 4) for simplicity reasons. Adopting the advertisement operations in the geospatial-based publish/subscribe model is addressed as a future work for this research.

Table 3.3: An example of content data structure for geospatial notifications

Attribute Name	Data Type	Description	Example
SensorID	String	ID of the sensor	{SensorID, "A1"}
X	Double	X-axis coordinate (UTM Zone 11N)	{X, -1211956}
Y	Double	Y-axis coordinate (UTM Zone 11N)	{Y, 1535061}
Temp	Double	Current temperature observation (°C)	{Temp, 12.8}
Humidity	Double	Current temperature observation (%)	{Humidity, 0.12}
ObsTime	String	Current observation time	{ObsTime, "24/11/2008 15:12:00"}

3.8 Summary

This chapter presented the Geospatial-based Publish/Subscribe model proposed in this research for handling events and subscriptions that are spatially-related to the geographic space in the publish/subscribe interaction style.

The chapter introduced geospatial events as a useful abstraction for representing dynamic phenomena in the real world. Geospatial events can be greatly utilized in distributing geospatial information about crucial happenings to interested parties and clients and thus being situational aware of time-sensitive events.

Section 3.5 proposed geospatial notification and geospatial subscriptions data models. Publisher clients can utilize the geospatial notification model to encapsulate simple spatial geometries and attributes data to represent events. Subscribers can utilize

the geospatial subscriptions model to define spatial as well as attribute constraints or filters over the geospatial notifications of interest. The geospatial subscription language supports various spatial operators to offer the subscribers more expressive filters.

Efficient matching of geospatial notifications against geospatial subscriptions is challenging. Section 3.6 investigated several problems in the matching and described the brute force method for the matching process. An efficient approach for geospatial notification matching is proposed in this research. Spatial indexing of registered geospatial subscriptions is utilized to enhance the speed and the efficiency of the matching engine, thus, disseminating geospatial notifications quickly to interested clients.

Lastly, the chapter discussed some issues regarding the delivery process of geospatial notifications to subscribers' applications. Subscribers are assumed to have previous knowledge about the content-data structures of delivered geospatial notifications. Their applications can be customized to interactively visualize and process received geospatial notification in GIS applications.

Chapter Four: Development of Real-Time Fire Emergency Response System

4.1 Introduction

This chapter presents Real-Time Fire Emergency Response System (RFERS), a system prototype developed in this research. The design of RFERS targets the transaction of spatio-temporal events crucial in the context of fire incidents and emergencies including: GPS location of emergency field assets (*e.g.*, fire trucks and police cars), temperature and humidity observations of wireless sensors, 911 reports of urban fire incidents, and thermal-infrared airborne imageries of active wildfires. Potential users of RFERS may include first responders, officers, fire chiefs, managers, and responsible agencies as those parties interested in being notified about the occurrence of specific events and accordingly respond and take the appropriate actions. Even the public can be part of the system interaction in broadcasting emergencies and informing interested parties. As time is an essential element in fire emergencies, real-time dissemination of instant happenings of geospatial events to interested clients is the key matter underlined in the system development.

Geospatial-based publish/subscribe framework, developed throughout Chapter 3, is realized in the development of RFERS. RFERS is not intended to be a complete software solution for emergency management applications. Rather, RFERS is developed as a proof of concept to evaluate the adequacy of the geospatial-based publish/subscribe framework and provides an indication of its potential use. The RFERS prototype is attempted to demonstrate the following features:

- Integrating publisher clients that publish heterogeneous, dynamic, and unexpected geospatial events essential for fire emergency services.

- Providing a real-time notification mechanism of essential information for many users simultaneously required for taking quick responses and conducting early warning scenarios.
- Offering an asynchronous style of communication and geospatial events dissemination needed for multi-incident emergency management.
- Enabling a “plug and play” approach for integrating new parties in the system workflow.
- Using GIS technology to map and analyze geospatial notifications leading to better realization of emergency situations.

The design and architecture of RFERS is described in the following section. This section also describes four types of information (*i.e., topics*) that can be contained in geospatial events with RFERS—Emergency Asset Locations, Wireless Sensor Observations, Fire Incident Reports, and Wildfire Thermal-Infrared Images. The data structure of each topic is described in details. Section 4.3 then describes the prototype implementation of RFERS.

4.2 System Design and Architecture

Geospatial publish/subscribe interaction model (see Chapter 3) is implemented in the design of RFERS. The main three components of RFERS are publishers, subscribers, and the notification service middleware. The architecture of RFERS is divided into a three-tier system model: the client tier, the business logic tier (also called application server), and the database tier. Publishers and subscribers fall inside the client tier. They interact

with the middleware tier by sending and receiving messages (*i.e.*, geospatial events/notifications). The notification service component resides in the business logic tier, where most of the application processing work occurs. The business logic tier, from one side, communicates with the client tier by handling all the incoming publications and subscriptions and consequently sending out notification messages. From the other side, the processes running inside the business logic tier are permitted access to the database tier for data storing and retrieving. A portion of the business logic functions resides in the users' applications in the client tier for processing and visualizing the messages data. APIs and TCP/IP protocols facilitate the underlying communication among the tiers. High-level architecture of RFERS tiers is shown in Figure 4.1.

The rest of this section describes each tier of the RFERS architecture in more detail. Before that, Section 4.2.1 introduces four topics of geospatial notification designed for RFERS. The data structure of each topic is described in detail.

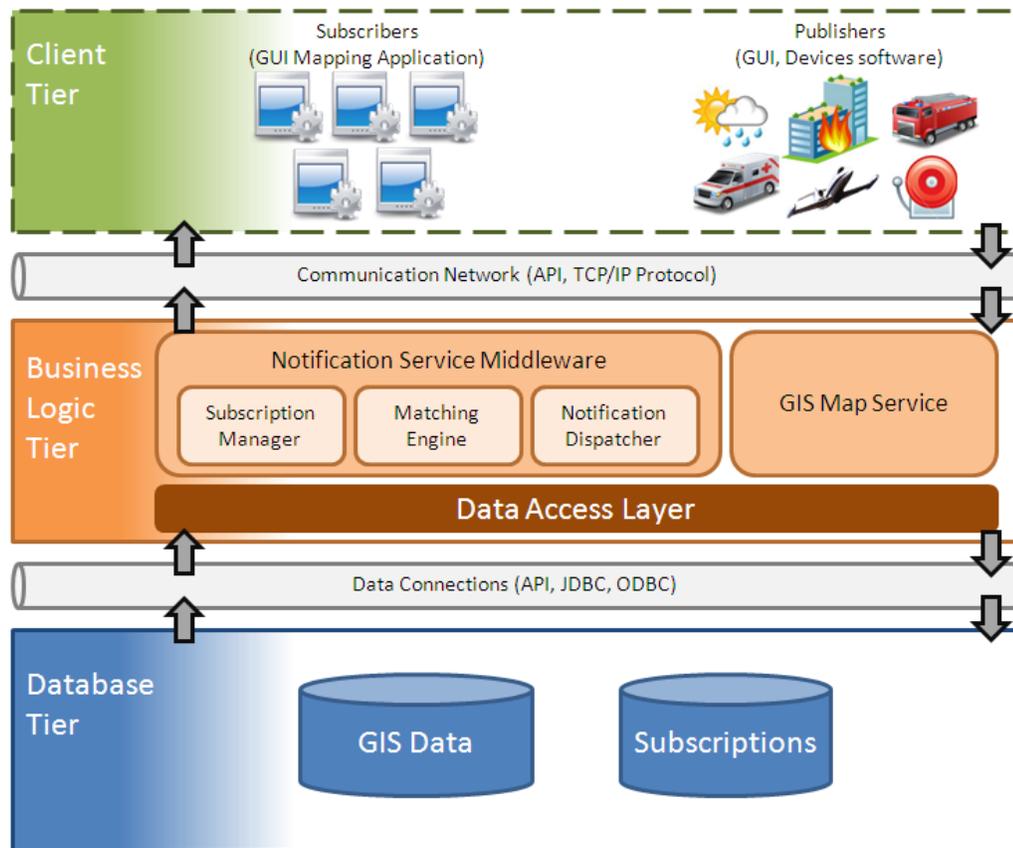


Figure 4.1: High-level architecture of RFERS

4.2.1 RFERS Topics and Data Models

In the development of RFERS, several *topics* of geospatial notifications are designed by which producers and consumers should structure their publications and subscriptions, respectively. The data model of each topic is predefined as a set of name/value pairs.

Those topics then are assumed as built-in standards recognized by the notification service unit. Thus, RFERS clients should realize the topics data models and follow their structures in order to successfully compile their issued operations. Otherwise, the operations will be discarded and not processed by the notification service.

For the purpose of this research, four geospatial notification topics are devised as potential applications of RFERS—Emergency Asset Locations, Wireless Sensor Observations, Fire Incident Reports, and Wildfire Thermal-Infrared Images. The aforementioned topics are selected in the sense that those scenarios would add great values in the operational response for fire emergencies. The following explains each topic and the associated data model in details.

4.2.1.1 Emergency Asset Locations

Being able to pinpoint the instant locations of the first responders' assets, such as fire trucks and ambulances, in the emergency field is of a great importance for effective crew allocation and efficient rescuing operations. Emergency commanders can be part of this scenario as they can be updated in real-time by the locations of their assets in the incident field, thereby adequately dispatching the available resources and decreasing the response time. In some cases, tracking step-by-step movements of all emergency assets is not in the interest of commanders, they would rather demand to be notified of certain instants where the location or the state of emergency assets is crucial. For instance, the commander requests to be notified if any emergency vehicles are within a 1000m radius of the incident location, or if the speed of certain vehicles exceeds 80 km/hr. This way, clients of RFERS are saved from receiving large amount of locations data that might be irrelevant or useless for them.

The data structure of this RFERS topic, namely Emergency Asset Locations, is detailed in Table 4.1. Emergency assets are assumed to be capable of publishing this type of geospatial notifications by potentially utilizing GPS devices integrated with computer

software. Notice that the mandatory type of name/value pairs in the content-data model must be assigned in publishing geospatial notifications to this RFERS topic.

Table 4.1: Geospatial notification content-data model in Emergency Asset Locations topic

Attribute Name	Data Type	Description	Mandatory / Optional	Example
Topic	String	Emergency Asset Locations topic (fixed string: "EALocation")	Mandatory	{Topic, "EALocation"}
GeometryPoint	Point ¹	Point feature represents the current position of emergency asset (X-axis Y-axis) (UTM Zone 11N)	Mandatory	{GeometryPoint, (-1351656.9 1735269.3)}
AssetID	String	Unique ID of emergency asset	Mandatory	{AssetID, "A1"}
AssetName	String	Name of emergency asset or driver	Optional	{AssetName, "Ala"}
Speed	Integer	The current speed of the asset (km/hr)	Mandatory	{Speed, 63}
PosDate	Date/Time	The current date of the acquired asset position (dd/mm/yyyy)	Mandatory	{PosDate, 12/4/2008}
PosTime	Date/Time	The current time of the acquired asset position (hh:mm:ss.sss)	Mandatory	{PosTime, 14:15:21.245}

4.2.1.2 Wireless Sensor Observations

Wireless sensor networks have been broadly utilized in distributed sensing of environmental phenomena. Those small devices (also called nodes) are capable of observing, processing, and broadcasting data for use by emergency responders. The technology of wireless sensors has proven its efficiency in monitoring physical or

¹ Point data type of geospatial notifications is discussed in Section 3.5.1

environmental conditions of open spaces, such as forest area (Yanjun *et al.* 2006), and even in closed areas, such as buildings (Wilson *et al.* 2007) and tunnels (Costa *et al.* 2007).

Table 4.2: Geospatial notification content-data model in Wireless Sensor Observations topic

Attribute Name	Data Type	Description	Mandatory / Optional	Example
Topic	String	Wireless Sensor Observations Topic (fixed string: "WSObservation")	Mandatory	{Topic, "WSObservation"}
GeometryPoint	Point	Point feature represents the current position of wireless sensor (X-axis Y-axis) (UTM Zone 11N)	Mandatory	{GeometryPoint, (-1532686.3 1234245.1)}
SensorID	String	Unique ID of wireless sensor	Mandatory	{ SensorID, "S1"}
SensorName	String	Name of wireless sensor (<i>e.g.</i> , serial number)	Optional	{ SensorName, "SENS001"}
Temperature	Float	The temperature observation value (°C)	Mandatory	{ Temperature, 12.5}
Humidity	Float	The relative humidity observation value (%)	Mandatory	{ Humidity, 0.24}
ObsDate	Date/Time	The date of the acquired wireless sensor observation (dd/mm/yyyy)	Mandatory	{ObsDate, 05/08/2008}
ObsTime	Date/Time	The time of the acquired wireless sensor observation (hh:mm:ss.sss)	Mandatory	{ObsTime, 14:15:21.245}

The design of the RFERS prototype assumes that wireless sensor nodes are deployed and distributed over an area to capture meteorological data, particularly temperature and relative humidity observations. The sensors publish their current locations and observations in forms of geospatial notifications. The data model of this

type of geospatial notifications is shown in Table 4.2. There are no restrictions on when geospatial notifications should be published by wireless sensors. In this topic, irregular patterns of geospatial notifications can be broadcasted throughout RFERS. RFERS subscribers usually register their interests in receiving certain readings in specific geographic locations. They are subsequently notified once these interests are met.

4.2.1.3 Fire Incident Reports

One of the fundamental responsibilities that any fire call center has is to process emergency calls or reports and dispatch the appropriate safety resources to the location of the incidents. Usually, emergency incidents are reported by the public via 911 phone calls or through internet reports issued to the responsible fire departments. The location or the address of the incident is vital and must be identified from the emergency report.

Immediately, first responders, such as police cars and medical services, are dispatched to the emergency locations for rescuing operations. This procedure may involve also notifying other agencies and resources for cooperation and effective mitigation of the emergency. Furthermore, the notification procedure may be extended to reach the neighbourhood and the surrounding community to the emergency location for warning and evacuation purposes. As noticed, many parties are potentially involved in fire emergency situations, and time delay is the essential condition here in terms of saving lives and property. One of the key elements necessitated for conducting successful response is to share incidents information among the involved parties in timely manner.

This geospatial notifications topic of RFERS involves reporting of present fire incidents by publisher clients, including: emergency call operators, dispatchers, officers, and even individual civilians or residents. They potentially can utilize any platform to

publish incident notifications, such as: applications GUI, the internet, web services, PDA, and phone SMS. The address information or the location coordinates of the incidents must be attached in the geospatial notifications content-data. In addition, photos that are captured or associated to the incident scene (*e.g.*, indoor photos, floor plan, and incident exterior scenes) can be encapsulated in the content-data.

Table 4.3 shows the data model of fire incident geospatial notification.

The notification service processes geospatial notifications that are published to this topic. It also performs address geocoding processing in case of unidentified location coordinates, and delivers the data to the interested subscribers in real-time. The following are some examples of scenarios where real-time delivery of incident information for RFERS subscribers is important and effective. Police cars get immediate notification about emergency incidents within a proximity of 3500m to their locations, emergency centers asynchronously receive incident notifications located within their administrative regions, and civilians get notified if emergency incidents occur within their neighbouring areas.

Table 4.3: Geospatial notification content-data model in Fire Incident Reports topic

Attribute Name	Data Type	Description	Mandatory / Optional	Example
Topic	String	Fire Incident Reports topic (fixed string: “FireIncReports”)	Mandatory	{Topic, “FireIncReports”}
GeometryPoint	Point	Point feature represents the location of the reported fire incident (X-axis Y-axis) (UTM Zone 11N)	Mandatory / Optional	{GeometryPoint, (-1522785.1 1248242.6)}
Address	String	The addressing information of the fire incident (Street, City, Province)	Optional / Mandatory	{ Address, “2454 17th AVE NE, Calgary, AB”}
IncDate	Date/Time	Date of the fire incident (dd/mm/yyyy)	Optional	{ IncDate, 14/11/2008}
IncTime	Date/Time	Time of the fire incident (hh:mm:ss.sss)	Optional	{ IncTime, 08:45:00.000}
ReportID	String	Unique ID of the fire incident report	Optional	{ReportID, “I198”}
Reporter	String	Reporter name	Optional	{Reporter, “Kassab”}
IncCause	String	Initial cause of the fire incident (Electricity, Human, Lightning, Gas...)	Optional	{IncCause, “Human”}
IncPhoto	Binary (serialized raster)	Photo of the fire incident scene	Optional	{IncPhoto, (binary_raster)}
Description	String	More description about the fire incident (phone number, loss, injuries, floor information...)	Optional	{Description, “Tel.: 403-224-5421”}

4.2.1.4 Wildfire Thermal-Infrared Images

Remote sensing technology has been widely recognized as an effective tool used for emergency planning, response, recovery, and mitigation efforts. Processing and visualizing remote sensing images give emergency managers the ability to have

comprehensive understanding of the emergency phenomenon, hence, leading to efficient execution of relief and evacuation operations. In wildfire, airborne sensing has been exploited in monitoring active wildfire spreading by capturing temporal images, visible and infrared, data of such disasters (Riggan *et al.* 2003). Moreover, the advent of Unmanned Aerial Vehicle (UAV) technology (see Figure 4.2) has made the acquisition of real-time images more feasible and efficient (NASA 2007, WRAP 2004). Recent research efforts have been targeting the evolution of integrating UAV systems with web-based 3D geoinformation services to provide real-time and directly georeferenced spatial information (Eugster and Nebiker 2008). Disseminating real-time alerting of such essential data in disaster scenarios can be an added value to the existing systems of emergency agencies.



Figure 4.2: UAV in monitoring wildfire disasters¹

¹ NASA, 2003, “Top Story - NASA Develops New Technology to Reduce Wildfire Response Time - August 21, 2003”, Accessed January 19, 2009
<http://www.nasa.gov/centers/goddard/news/topstory/2003/firesames.html>

Wildfire Thermal-Infrared Images topic is developed in RFERS to accommodate remote sensing images of wildfire in the system interaction by means of geospatial notifications. The content-data model geospatial notifications designed for this topic is shown in Table 4.4. An instance from this class of geospatial notifications encapsulates an image file which is a spatial dataset captured at the scene of the fire hazard. Usually, each image dataset is associated with a time stamp (*i.e.*, temporal images) and RFERS publishers of this topic are concerned in broadcasting consecutive images of the fire through a period of time. The image datasets, encapsulated in geospatial notifications, are supposed to be georeferenced with a predefined coordinate system, processed, and ready for visualization and mapping. RFERS subscribers can register their interests in this topic and consequently receive and visualize those datasets once published. Besides, they can constrain the delivery of those geospatial notifications by a region boundary, thus, they receive those images which are located in the predefined boundary region (*i.e.*, using *Contain* or *Cross* spatial predicates as discussed in Section 3.5.2).

Table 4.4: Geospatial notification content-data model in Wildfire Remote Sensing Images topic

Attribute Name	Data Type	Description	Mandatory / Optional	Example
Topic	String	Wildfire Remote Sensing Images topic (fixed string: "WFireRSImage")	Mandatory	{Topic, "WFireRSImage"}
RasterFile	Binary (serialized raster) / GeometryPolygon	Remote sensing raster file (Georeferenced, e.g., GeoTIFF) (UTM Zone 11N)	Mandatory	{RasterFile, (binary_raster)}
RasterDate	Date/Time	Date of remote sensing image acquisition	Mandatory	{RasterDate, 02/03/2008}
RasterTime	Date/Time	Time of remote sensing image acquisition	Mandatory	{RasterTime, 11:25:35.100}
RasterDesc	String	Description of the remote sensing image (wind speed, temperature, location info, type of trees...)	Optional	{RasterDesc, "average temperature of 25oC, relative humidity of 8%, medium-high wind speed"}

4.2.2 Client Tier

Clients of the RFERS application reside in the topmost level of the system architecture, as depicted in Figure 4.1. They are either publishers or subscribers of geospatial notifications. The main function of RFERS is the mediation of geospatial notifications conveyed from publishers to subscribers asynchronously and in a timely manner. RFERS clients are distributed over the network. They are unknown to each other as they only communicate with notification service middleware regardless of other clients existing on the network. In order for clients to communicate with the RFERS notification service unit, they use Application Programming Interface (API) libraries which abstract the complexity of the communication protocols to higher software level and facilitate performing publish and subscribe operations.

In RFERS, the implementation of clients' applications is independent of the system behaviour. In other terms, clients can utilize any software platform or application to communicate with the notification service and be part of the system interaction. However, clients' applications have to implement the APIs provided by the notification service in order to perform publish/subscribe operations. In the RFERS prototype implementation (see Section 4.3), a desktop GIS application is implemented to be utilized by subscriber clients. Simulation programs are implemented to act as publishers of geospatial notifications. Those applications are developed for the purpose of this research to demonstrate the interaction mechanism and evaluate the underlined geospatial-based publish/subscribe model. The following describes the major functionalities of RFERS clients' applications.

Subscribers' applications interact with the RFERS middleware unit via the network and visualize the received geospatial notifications over a GIS map. RFERS subscriber clients register their specific interests in receiving geospatial notifications published to one or more of the RFERS topics, discussed in Section 4.2.1. They are assumed to know about these topics as well as the specifications of their data models. A single subscription should be addressed to one topic, and the encapsulated filter should be formulated strictly over the attributes of the data model associated to that topic. RFERS subscribers can utilize their applications' GUI to formulate their subscription filters then execute the required subscribe operations. The work flow of the subscriber client application is shown in Figure 4.3. A subscription (Sub_g) data is processed and prepared according to the content-data format of the required topic and finally transmitted through the RFERS web network. The middleware notification service receives, processes, and

stores the subscription inside the database tier for later matching. Finally, an acknowledgment message (*Ack_Msg*) is sent back to the subscriber confirming that the subscribe operation is compiled and registered successfully. As soon as a published geospatial notification (n_g) matches the registered subscription, the geospatial notification will be sent to the subscriber who owns the matched subscription. In the subscriber client side, a message listener asynchronously triggers the core computation of the subscriber application once a message (*i.e.*, geospatial notification) arrives from the RFERS web network. Automatically, the message is parsed to expose the content data of the message. The local business logic of the client application transforms the message data into a spatial feature thus to be overlaid with the GIS base map layers already contained in the map control. The subscriber then can store the new received data inside a local database repository and conduct further GIS or mapping analysis using the tools provided in the GUI.

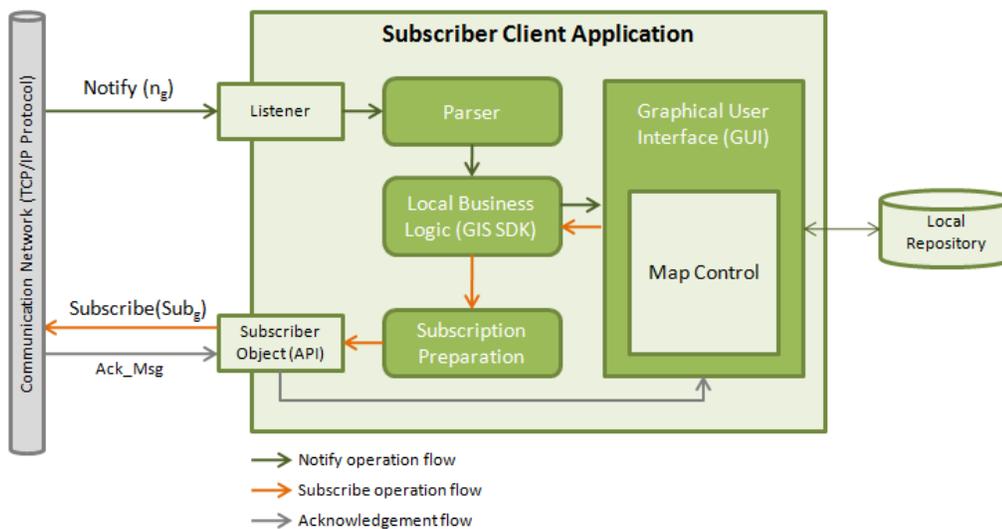


Figure 4.3: RFERS subscriber client application

Publisher clients are the sources of geospatial notifications data. They generate and broadcast geospatial notification messages throughout the RFERS web network. The data encapsulated in those messages should be structured according to the data models of the RFERS topics specified in Section 4.2.1. Each geospatial notification message should be tagged (*i.e.*, defining the “*Topic*” attribute predicate in the message content-data) by the topic string name. Thus, the notification service can distinguish the geospatial notification topic and the content-data format. Any failure in structuring and defining the geospatial notification predicates leads the notification service middleware to discard the data and not reach the subscriber clients.

4.2.3 Business Logic Tier

There are two main components that reside in this tier: the notification service middleware and the GIS map service. RFERS notification service acts as the intermediary component and supplies the communication between RFERS clients. There is no direct connection between clients together as mentioned in Section 4.2.2. Rather, the notification service takes the responsibility of channelizing the information from publishers to subscribers.

The notification service interacts closely with the database tier through the data access layer. Upon receiving geospatial subscriptions issued from subscribers, the notification service manages and stores the subscriptions information inside the predesigned subscriptions database. Also, when receiving geospatial notifications issued from publishers, the notification service retrieves the registered subscriptions and conducts the matching process to find the interested subscribers in such notifications. As depicted in Figure 4.1, the notification service consists of three key processes: the

subscription manager, the matching engine, and the notification dispatcher. To clarify the middleware tier work flow and the function of each of the notifications service's processes, the remaining of this section details the process flow when a subscriber client registers a geospatial subscription and a publisher client broadcasts a geospatial notification.

As soon as a client subscribes a geospatial subscription Sub_i , the operation message is transmitted to the notification service unit. The subscription manager takes the role of processing the subscription's message and storing the encapsulated information inside the RFERS subscriptions' database. The process starts by parsing the subscription's message to expose the internal information which usually includes a spatial predicate, an attribute predicate, and other attributes related to the identity of the subscriber client, such as: ID and Name attributes. The notification service uses the ID attribute to uniquely identify the client and send him published geospatial notifications that match his subscription(s). The spatial predicate, as defined in Section 3.5.2, comprises a base geometry G_b , a spatial operator SOp , and an optional buffer value $buff$. After parsing the subscription's message, the next step is applying the buffer zone on G_b with the value of $buff$ and accordingly modifies the original G_b . Afterwards, the geospatial subscription is clustered according to the assigned SOp type and stored in the associated table of the subscriptions database. More details about the process of preparing issued geospatial subscriptions are described in Section 3.6.3.3. The spatial indexes created for the subscriptions tables are maintained during the insertion of the new base geometry of the geospatial subscription. Finally, the notification service sends back an acknowledgement message to the client confirming that the geospatial subscription is

processed and registered successfully. Figure 4.4 illustrates the aforementioned process flow.

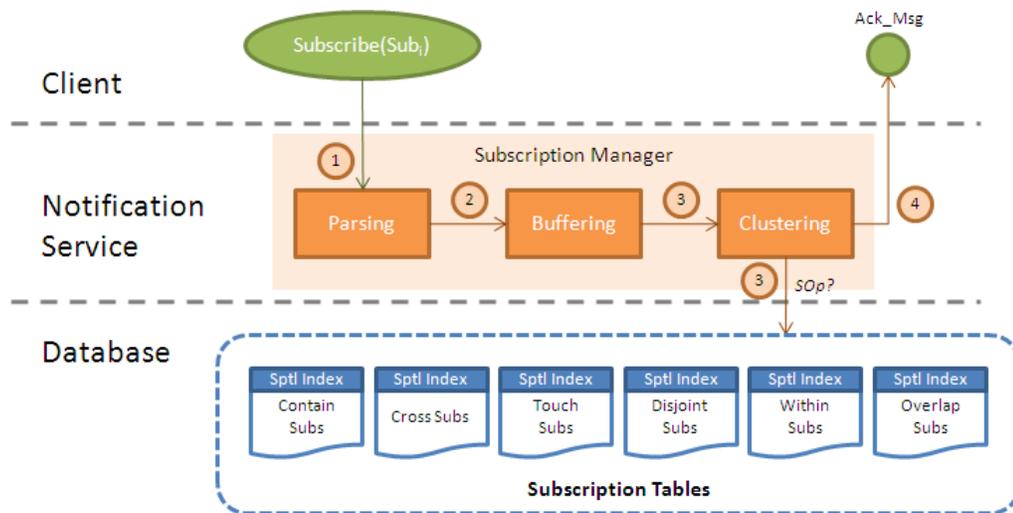


Figure 4.4: The notification service process flow upon issuing geospatial subscription

When a publisher client issues a geospatial notification n_g , similarly the operation message is transmitted to the RFERS middleware. Figure 4.5 illustrates the process procedure inside the notification service upon publishing a geospatial notification. This time, the function of the notification service is to deliver n_g to RFERS subscribers who have previously registered geospatial subscriptions that match the n_g features. The matching engine is the primary and the most consuming process at this stage; it takes charge of filtering the stored geospatial subscriptions and finding the matched set. The RFERS matching process is developed similar to the proposed matching approach discussed in Section 3.6.3.3. Here the matching procedure is clarified again for the sake of completing this context.

After parsing the n_g message (Step 1 in Figure 4.5), the matching engine takes the spatial feature of n_g (*i.e.*, the geometry of n_g) as an input and executes a maximum of six spatial queries to retrieve the geospatial subscriptions whose spatial geometries satisfy the queries criteria (Step 2 in Figure 4.5). Those spatial queries are shown in Table 4.5. The spatial indexes created from the subscriptions tables are used in this step to accelerate the searching process. This is the key improvement of the matching engine (more details are discussed in Section 3.6.3.3). The result from this step is a subset of geospatial subscriptions whose spatial constraints (*i.e.*, spatial predicates) are satisfied by the geometry of the published n_g . Subsequently, the other attributes of the n_g are evaluated against the attribute predicates, if available, assigned in the geospatial subscriptions subset resulted from the previous spatial matching step (Step 3 in Figure 4.5). The attribute evaluation here is conducted in a one-by-one basis. The geospatial subscriptions that pass this step are considered to completely match the published n_g . Thus, the associated subscribers will be notified about the n_g . The notification dispatcher process takes the matched geospatial subscriptions along with the published n_g and pushes the n_g message data to the associated subscribers' applications (Steps 4 and 5 in Figure 4.5). The notification dispatcher identifies each subscriber by his unique ID throughout the RFERS network, thus uses the subscriber's ID to deliver the n_g message for the subscriber application through a *Notify()* operation.

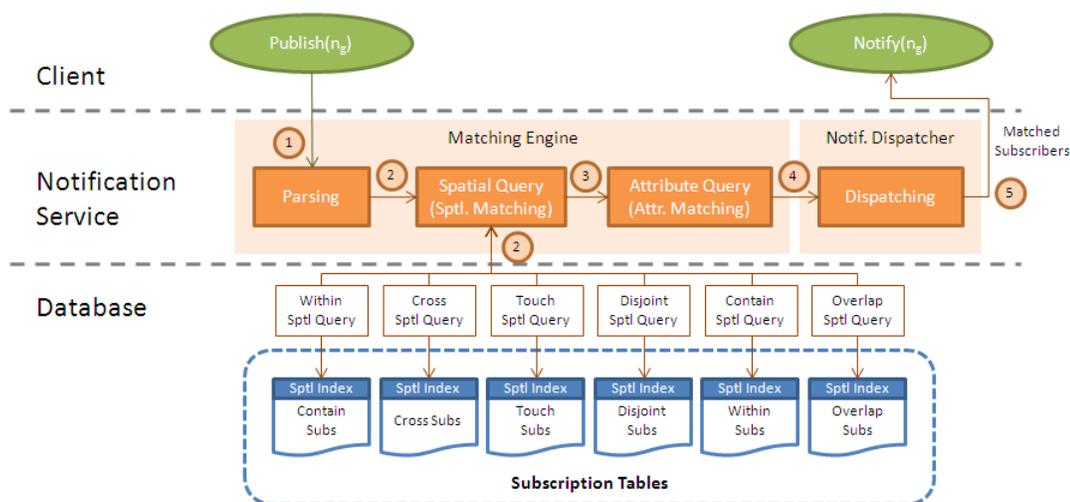


Figure 4.5: The notification service process flow upon publishing a geospatial notification

Table 4.5: The spatial queries executed in the spatial matching process

Base Geometry	Target Geometry Table	Spatial Query
The geometry of the geospatial notification (n_g .Geometry)	Contain_Subs	SELECT * FROM Contain_Subs WHERE Within (n_g .Geometry, <i>Contain_Subs</i> .Geometry) = true;
	Cross_Subs	SELECT * FROM Cross_Subs WHERE Cross (n_g .Geometry, <i>Cross_Subs</i> .Geometry) = true;
	Touch_Subs	SELECT * FROM Touch_Subs WHERE Touch (n_g .Geometry, <i>Touch_Subs</i> .Geometry) = true;
	Overlap_Subs	SELECT * FROM Overlap_Subs WHERE Overlap (n_g .Geometry, <i>Overlap_Subs</i> .Geometry) = true;
	Disjoint_Subs	SELECT * FROM Disjoint_Subs WHERE Disjoint (n_g .Geometry, <i>Disjoint_Subs</i> .Geometry) = true;
	Within_Subs	SELECT * FROM Within_Subs WHERE Contain (n_g .Geometry, <i>Within_Subs</i> .Geometry) = true;

The second component of the business logic tier is the GIS map service. RFERS clients are provided this service to access and view georeferenced maps and GIS data in various scales. Subscribers can utilize this service to overlay received geospatial notifications and conduct further modeling and geoprocessing analysis.

4.2.4 Database Tier

RFERS database tier holds two databases: the subscriptions database and the GIS database. Only the middleware tier has a direct access to both databases through a data access layer.

The subscriptions database stores all the geospatial subscriptions issued by subscribers that are successfully compiled and processed by the notification service. As mentioned in the previous section, the registered geospatial subscriptions are clustered in six tables: *Contain_Subs*, *Cross_Subs*, *Touch_Subs*, *Disjoint_Subs*, *Within_Subs*, and *Overlap_Subs*, where each of them records the attributes of the geospatial subscriptions whose spatial predicates are assigned *Contain*, *Cross*, *Touch*, *Disjoint*, *Within*, and *Overlap* spatial operators, respectively. The subscriptions tables have the same fields schema shown in Table 4.6. A geometry data type field, called the *spatial_shape*, is created in each of the aforementioned tables. The values of the shape field are assigned the base geometries of the spatial predicates encapsulated in the geospatial subscriptions, and those geometries refer to the spatial features of the geospatial subscriptions. Moreover, a spatial index is created on the *spatial_shape* field and used in the matching procedure while searching for match geometries with the published geospatial notification geometry.

Table 4.6: Data fields structure of the geospatial subscription tables

Field Name	Data Type	Description
Topic	String	The topic name string of the geospatial subscription ('EALocation', 'WSObservation', 'FireIncReports', or 'WFireRSImage')
spatial_shape	Geometry	The base geometry of the geospatial subscription
subID	Long integer	Unique ID of the geospatial subscription
subscriberID	Long Integer	Unique ID of the subscriber client (application)
att_filter	String	the attribute filter/predicate assigned in the geospatial subscription
subDate	Date/Time	Date of the geospatial subscription
subTime	Date/Time	Time of the geospatial subscription

The GIS database is designed as a repository of spatial data needed for developing a regional base map. Through the GIS map service component in the middleware tier, the application's GUI provided for subscriber clients can be used to visualize the base map data layers. Subscribers would use the base map mainly for overlaying with the received geospatial notifications. Also, the base map can be utilized for performing various GIS analysis, such as: attribute and spatial query, proximity, and network analysis. The base map data layers include provinces, regional municipalities, capitals and major cities, highways, water areas, and topography features created at the national scale of Canada. Other large-scale and detailed data layers, including local streets, healthcare facilities, schools, police stations, buildings footprints, parks and recreations, land use, and other features, are collected particularly for the Alberta and British Columbia provinces. In addition to that, multi-scale satellite image services are included to enrich the base map with more comprehensive information. The GIS data is collected mainly from DMTI

Spatial¹, ESRI² and GeoGratis³, then processed, prepared, and migrated in the GIS database.

4.3 Prototype Implementation

This section describes the prototype implementation of RFERS developed throughout this chapter. The RFERS prototype is not meant to be a commercial product for the market, where further technical and development issues should be maintained in considering this matter. Rather, the focus here is to test the adequacy of the RFERS architecture and evaluate the interaction and dissemination of data by means of the proposed model, geospatial publish/subscribe, for a potential market product in the future.

The prototype implementation of RFERS is conducted in developing three main components to model the interaction mechanism: (1) simulation programs for publishing geospatial notifications into the four RFERS topics, (2) a desktop GIS application for subscribers to receive geospatial notifications and visualize the data in a GIS environment, and (3) the notification service middleware for managing subscriptions, matching geospatial notifications against registered subscriptions, and dispatching geospatial notifications to subscriber clients. The next sections explain the development tools used for the prototype implementation then describe each of the prototype components in details.

¹ DMTI Spatial, 2008, "Location Intelligence Solutions. Mapping Software, Enterprise Mapping Solutions, Address Data, Geo Data, DMTI Spatial", Accessed March 25, 2009, <http://www.dmtispatial.com/>

² Environmental Systems Research Institute (ESRI), 2009, "ESRI - The GIS Software Leader", Accessed March 25, 2009, <http://www.esri.com/>

³ GeoGratis, 2009, "GeoGratis – Home", Accessed March 25, 2009, <http://geogratias.cgdi.gc.ca/geogratias/en/index.html>

4.3.1 Development Tools and Software Packages

In general, there are plenty of programming libraries and commercial software packages that can be utilized to develop the RFERS prototype. However, the choice of the required development tools has been made considering the following factors: user-friendly programming environment, availability of the software packages for the research, availability of documentations and user tutorials, and the author's previous experience. TIBCO Enterprise Messaging Service (EMS) v4.4 and ESRI ArcGIS v9.3 products are the main utilized packages from the RFERS prototype implementation. TIBCO EMS is a standard-based messaging software that serves as a communications backbone between distributed and a wide range of applications and platforms (TIBCO 2008). ESRI ArcGIS family is widely known, easy to use GIS software utilized in many GIS applications and development solutions (ESRI 2004). Both of the software packages provide .NET Framework APIs for developers to customize extended and advance applications that meet their requirements. In the implementation of the RFERS prototype, C# is the programming language used, in addition to EMS and ArcGIS Engine .NET SDK for implementing the system prototype components. For building the RFERS database, ArcSDE v9.3 Workgroup Geodatabase and Microsoft SQL Server 2005 Express Edition are used for the storage and access management of spatial data required for the purposes of the prototype.

4.3.2 Simulation of Geospatial Notifications

For experiment and evaluation purposes, four versions of a simulation program, one for each RFERS topic, have been developed to publish geospatial notifications throughout the system prototype in regular or irregular timing basis. The simulation programs are

Windows based desktop interfaces implemented in C# and using EMS API for creating messages sender object. They facilitate the creation of geospatial notifications content-data and the execution of publish operations. The geospatial notification data models described in Section 4.2.1 are realized in the simulation programs; the simulation program interface is used to fill out the attribute values according to the RFERS topic specifications. Figure 4.6 shows the simulation programs for publishing geospatial notifications into (a) the Emergency Asset Locations, (b) the Wireless Sensor Observations, (c) Fire Incident Reports, and (d) Wildfire Remote Sensing Images topics.

As discussed previously in Section 4.2.2, clients applications are independent of the system behaviour. They only use the appropriate APIs provided by the notification service to perform publish/subscribe operations. As the simulation programs act as RFERS publishers, they implement the notification service APIs for publishing geospatial notifications. One can create many instances of the simulation programs to publish geospatial notifications to one or more RFERS topics simultaneously. These instances also can be run on different computers and publish geospatial notifications without any synchronization with other instances. Furthermore, the simulation instances can join and leave the RFERS network irregularly and without pre-configurations needed. Therefore, the irregular behaviour of publisher clients can be demonstrated using the simulation programs.

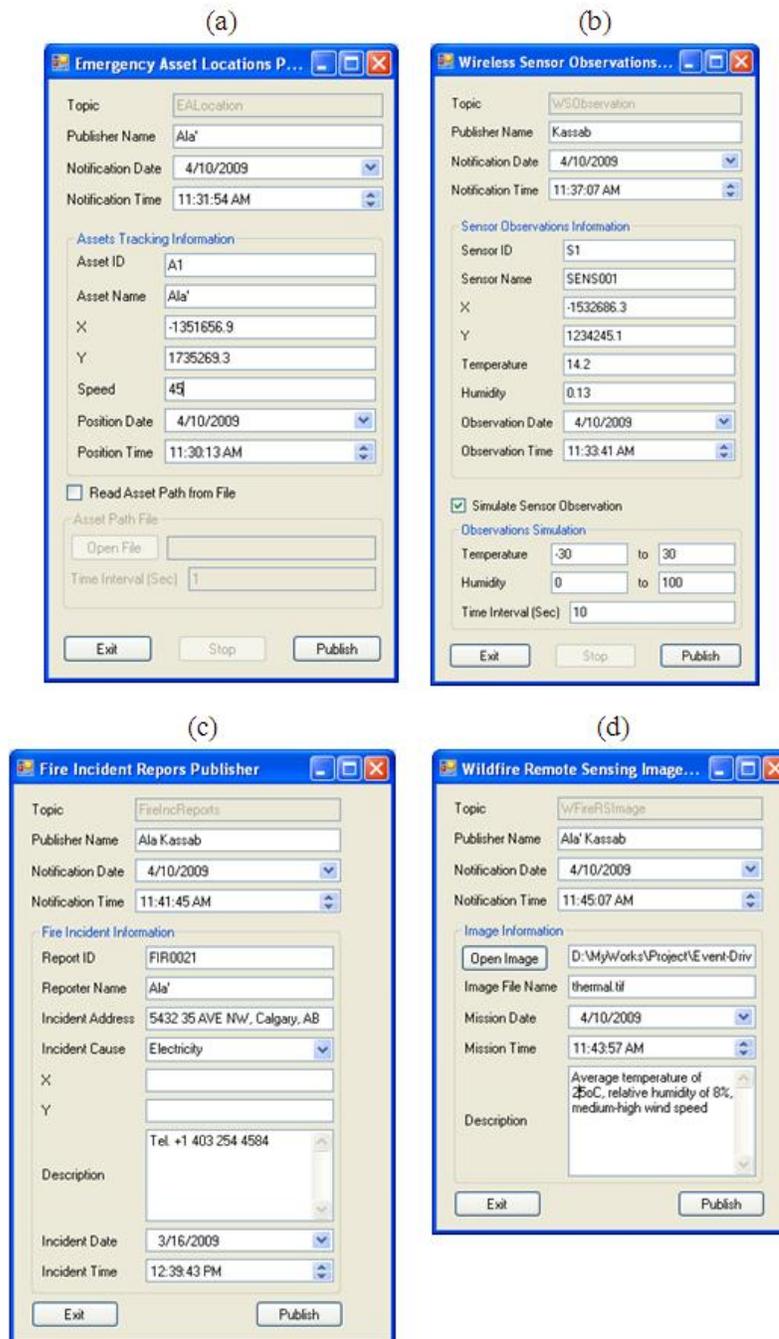


Figure 4.6: The simulation programs for publishing geospatial notifications into RFERS topics: (a) Emergency Asset Locations topic, (b) Wireless Sensor Observations topic, (c) Fire Incident Reports topic, and (d) Wildfire Thermal-Infrared Images topic

4.3.3 Subscriber GIS Application

RFERS subscriber clients are provided a front-end software application capable of interacting with the RFERS framework and managing the received geospatial notifications in a GIS environment. The application is implemented in C# and uses ArcGIS Engine .NET APIs to build and customize GIS mapping functions. The application also uses EMS .NET API to create an instance of a messages receiver object.

When a user runs the application, a login screen, depicted in Figure 4.7, turns up asking the user to enter a user name needed to identify the user location on the RFERS network as well as maintain previous subscriptions registered by the same username. As the user logs in to the main application, the notification service opens a unique communication channel with the subscriber's application for potentially exchanging messages. In addition, the user is granted an access to a map service for viewing base map data prepared inside the RFERS geodatabase.



Figure 4.7: Subscriber's application login screen

The application user interface, depicted in Figure 4.8, contains two main tabs, namely Map and Subscriptions/Notifications. In the Map tab, a base map is viewed inside

a mapping control. Several mapping and navigation tools are provided in the main toolbar, including: zoom in, zoom out, pan, full extent, search, identify, measure distances and areas, go to XY, and others. These tools help the user to explore the overall region data. While receiving geospatial notifications (*i.e.*, dynamic spatial features), those features will be overlaid with the base map data and visualized automatically inside the mapping control. The user then is updated by the geospatial notifications visually and uses the received data for conducting further GIS analysis which would help him to take the appropriate actions if needed.

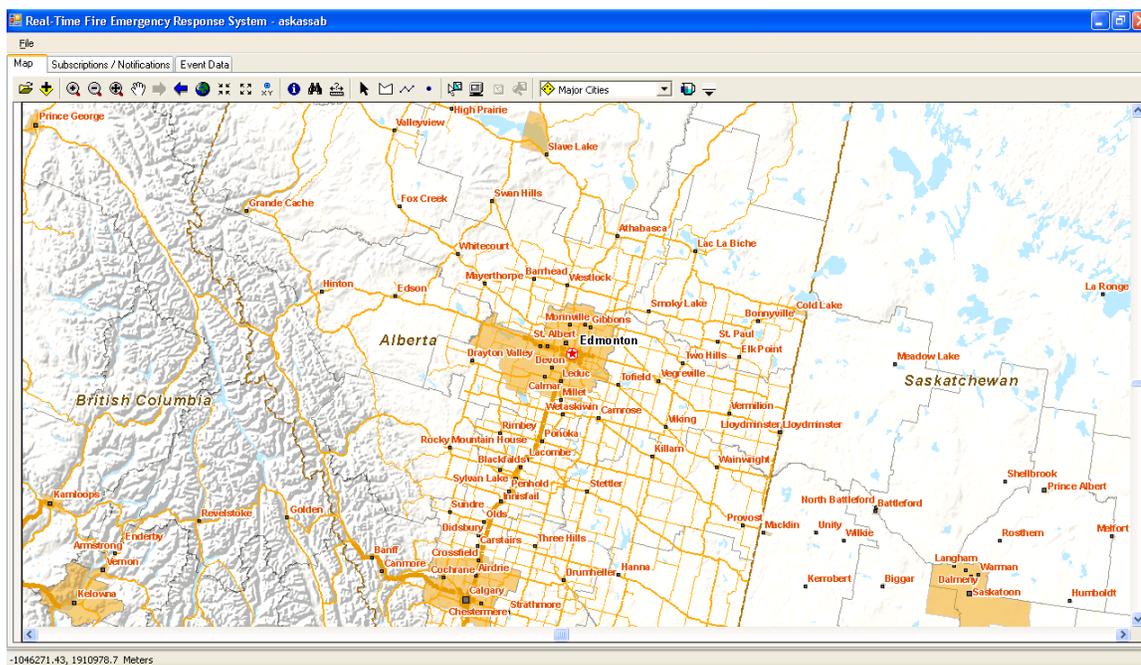


Figure 4.8: Subscriber's application Map tab

The Subscriptions/Notifications tab is used to perform geospatial subscriptions or explore the information of the received geospatial notifications. The tab area, depicted in Figure 4.9, is divided horizontally into two main horizontal panels, namely Subscriptions and Notifications. The user utilizes the Subscriptions panel to subscribe or unsubscribe

for geospatial notifications. The Notification panel shows several attributes of the received geospatial subscriptions in a table form, the attributes mainly include: the matched subscription ID, the publisher name, date and time of sending and receiving the notification, and the content-data attributes.

By clicking the subscribe button, a dialog box appears requesting the user to formulate the parameters values of the geospatial subscription. According to the geospatial subscription model described in Section 3.5.2, three main parameters are needed to define a geospatial subscription: a topic string name, an attribute filter/predicate, and a spatial filter/predicate. As shown in Figure 4.9, the user selects one of the four RFERS topics via the topics combo box specifying the topic string name that the user is interested in. Then, the user goes further to assign an attribute and spatial filters as required. If the user does not assign any of those filters, then the user is interested in receiving all the geospatial notifications published to the assigned topic name.

Defining the attribute filter is conducted by a manual typing of the required query in the appropriate text box for this feature. As mentioned previously, previous knowledge about the geospatial notifications data models in each topic is assumed. Thus, the user should be aware of the correct syntax as well as the exact string names of the topic's geospatial notification attributes in defining the attribute query.

For defining the required spatial filter, three parameters should be assigned by the user, namely the base geometry, the spatial operator, and the buffer value. The user defines the type of the geometry (*i.e.*, point, polyline, or polygon) according to the required spatial filter. Further, the user utilizes the Map tab to define the shape and the

spatial location of the subscription's geometry; this is conducted by either manual drawing of the geometry on the mapping control or selecting an existing spatial feature(s) whose spatial shapes and locations will be taken to define the geometry. The user then uses the appropriate dialog box controls to assign the other two parameters.

Finally, the user confirms the subscription parameters and closes the dialog box, thus the operation is sent out to the RFERS notification service for registration. After a successful registration, the geospatial subscription will be added as a table record in the Subscription panel entailing that any geospatial notification that matches the registered subscription will be delivered to the user's application. Unsubscribe operation can be simply conducted by selecting one of the registered geospatial subscriptions then clicking unsubscribe button, thus unregistering the selected subscription from the notification service.

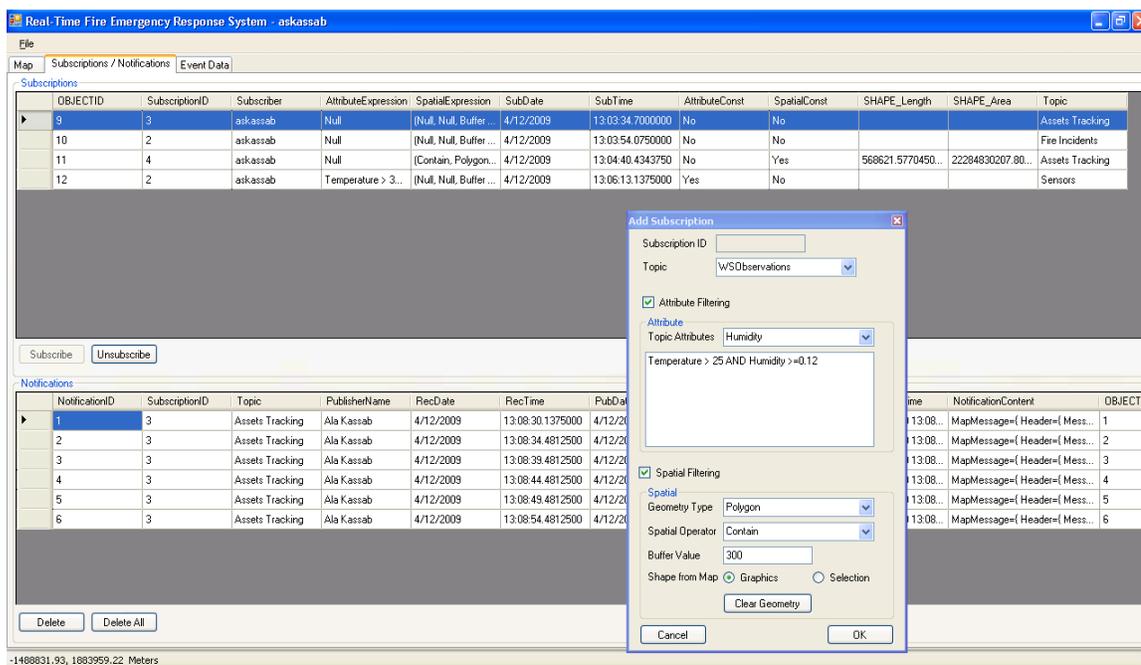


Figure 4.9: Subscriber's application Subscription/Notifications tab

Likely to the publisher simulation programs discussed in the previous section, the independent and the irregular behaviour of RFERS subscriber clients is assumed. The subscriber GIS application can be used by many users and deployed in distributed computers over the network. A user can also join and leave RFERS network, by running or closing the application respectively, irregularly and without pre-knowledge of other clients on the network. Although the number of active clients on the RFERS network would affect the performance of the middleware component, there are no restrictions on how many clients should interact simultaneously in the RFERS prototype.

4.3.4 Middleware Server

The notification service component is implemented in C#. ArcGIS Engine .NET SDK is used for handling and managing the geospatial subscriptions inside a spatial database as well as for conducting the matching process when geospatial notifications are published (see Section 4.2.3). In addition, EMS API is used for receiving published geospatial notifications as well as for dispatching them to matched subscribers' applications. The RFERS database is created and prepared using ArcSDE 9.3 and SQL Server 2005 Express to support spatial type of data. Feature classes are prepared in the RFERS database to store the geometry features of the geospatial subscriptions that are issued by subscribers. Multi-level grid, a maximum of three grid levels, spatial index is the only indexing technique supported in SQL Server Express database (ESRI 2008). For the purpose of this research and the prototype implementation, however, that the grid spatial indexing is considered sufficient and can be used to improve the matching process.

The GIS map service component (see Figure 4.1) hosts a map service which is consumed by subscribers' applications to view a base map for Canada. The map service

is created using ArcGIS Server v9.3 and the base map spatial layers are managed in the RFERS database. ArcGIS Server v9.3 is an easy-to-use software for creating several GIS services and resources for sharing information and GIS functionalities by making them available to clients' applications. Different tools provided by the software are utilized to improve the map service for faster performance and high-quality in viewing the base map.

4.4 Summary

This chapter presented Real-Time Fire Emergency Response System (RFERS), a system prototype developed in this research. Geospatial-based publish/subscribe framework proposed in Chapter 3 is realized in the development of RFERS. Clients of this system interact by means of geospatial events/notifications. RFERS is a proof of concept prototype proposed in this research to demonstrate how dynamic geospatial information in the context of fire emergency can be disseminated in real-time to interested parties thus potentially improving response actions and situational awareness. The system is also developed to evaluate the adequacy and the performance of the proposed geospatial-based publish/subscribe model.

The chapter proposed four geospatial event topics: Emergency Asset Locations, Wireless Sensor Observations, Fire Incident Reports, and Wildfire Thermal-Infrared Images. In the Emergency Asset Locations topic, geospatial notifications represent the current locations of emergency assets moving in the geographical space. Subscriber clients can be notified in real-time about those assets whose properties satisfy certain spatial and attributes constraints, such as assets that are within proximity of 1000m from

an emergency location. In Wireless Sensor Observation topic, meteorological data observed by distributed sensors are published in RFERS. Subscribers, for instance, can receive a subset of those observations that are acquired in certain geographic region and their temperature readings when they exceed a certain value which can potentially cause an ignition of fire. In Fire Incident Reports, addresses of present fire incidents are published as geospatial notifications. Thus, interested clients can be notified in timely manner and dispatch the appropriate response resources. In Wildfire Thermal-Infrared topic, airborne temporal images captured at the scene of active forest fires can be disseminated to monitoring agencies, thus, they would allocate firefighters and available emergency personnel for efficient emergency mitigation.

The architecture of RFERS is divided into three tiers: clients tier, business logic tier, and database tier. Clients communicate with the notification service unit that resides in the business logic tier (*i.e.*, the middleware component) by performing publish/subscribe operations. The middleware component communicates with the database tier to manage subscriptions inside the RFERS database and match geospatial notifications with registered subscriptions. The middleware component also offers a GIS map service and provides RFERS clients with GIS base map data useful to visualize geospatial notifications and conduct GIS mapping analysis.

The chapter also described a prototype implementation of RFERS. C# programming language and commercial software packages, EMS v4.4 and ArcGIS v9.3, are used for the implementation. A desktop GIS application is implemented for the use of subscriber clients, and simulation programs are developed to simulate publishing geospatial notification to RFERS topics. The irregular behaviour of RFERS clients is

assumed in the prototype implementation; many instances of those applications can be run on different computers simultaneously without any synchronization of the interaction processes.

Chapter Five: Testing Results and Performance

5.1 Introduction

This chapter discusses the attempts made to test the RFERS prototype developed in Chapter 4. Further, the chapter evaluates the performance of the proposed interaction model, namely geospatial-based publish/subscribe, through using the RFERS prototype with simulated data.

In this chapter, several simulation scenarios are demonstrated to show the mechanism of disseminating geospatial events in real-time to interested clients. The scenarios also give an indication on the usefulness of geospatial-based publish/subscribe interaction in the context of fire emergency and situational awareness. The efficiency of the RFERS prototype can be evaluated by how quick dynamic geospatial information is reflected and delivered to the interested clients. This largely depends on the performance of the applied matching engine in searching for matched interests (*i.e.*, geospatial subscriptions) within a potentially large number of clients' subscriptions. Thus, this chapter evaluates the performance of the RFERS prototype in matching geospatial notifications with different sets of numbers of geospatial subscriptions.

The chapter is divided into two main parts. The first part, in Section 5.2, investigates the efforts made to test the RFERS prototype implementation and whether the system's components meet their requirements. The second part, in Section 5.3, discusses the evaluation process of the implemented geospatial-based publish/subscribe interaction by testing the RFERS matching engine with simulated datasets of geospatial subscriptions.

5.2 RFERS Prototype Testing

This section attempts to assess the implemented prototype of RFERS developed throughout Chapter 4; the actual implementation of the prototype components is described in Section 4.3. The prototype testing herein can be stated as the process of validating and verifying whether the functions provided by the prototype software meet their requirements and the interaction between the system's components work properly. The major testing process has been conducted by simulating the geospatial publish/subscribe interaction in different scenarios. Three interaction scenarios are envisioned between the RFERS publishers and subscribers utilizing the topics' data models proposed in Section 4.3.1. The next three sub-sections explain the outcomes from each scenario in details. In addition to that, the prototype has been demonstrated for other members of the Positioning & Mobile Information System group at the University of Calgary with the intent of obtaining feedbacks regarding the usability and the adequacy of the software. This has contributed on enhancing the prototype implementation by fixing some software bugs as well as in highlighting some techniques that can be considered to improve the software functionalities.

For testing purposes, the RFERS prototype components have been set up as follows: the RFERS middleware as well as the RFERS database (see Section 4.3.4) has been installed on a server workstation with 2.4GHz Intel Core 2 Quad-Q6600 CPU and 4GB RAM running Windows XP Professional. The RFERS GIS subscriber application and the publisher simulation programs (see Section 4.3.3 and Section 4.3.2 respectively) have been set up on a local computer with 1.86GHz Intel Core 2 Duo E6320 CPU and 2GB RAM also running Windows XP Professional. Both machines, the server and the

local, are connected through a local network, thus ease the communication of information between both computers and without taking care of other network and accessibility configurations which is not the concern on the testing process. The testing results, including the major limitations and problems, are presented in the last subsection.

5.2.1 Emergency Assets Close by a Fire Incident

To illustrate this scenario, let us consider the following example. An officer in a fire station is concerned about immediate notification in any fire incident reported in the designated region area the fire station is responsible for. The fire station service area is bounded by the geographical region of the Forward Sortation Area (FSA) code number “T2N”, Calgary, Alberta—located in this region are the University of Calgary and the Foothills Hospital. To implement this type of interest utilizing the RFERS software, the officer would subscribe his interest in the geospatial notifications that will be published to Fire Incident Reports topic (“FireIncReports” topic string name) and their locations are contained by the “T2N” FSA geographical boundary. Thus, as the RFERS subscriber application is running on the officer’s local computer, the officer would locate the FSA polygon that has the code “T2N” utilizing the base map data and the mapping tools provided in the application. After that, the officer initiates a new subscription and assigns the parameters with values that fit the respective interest. The parameters and their values are defined as the following: Topic Name = *FireIncReports*, Attribute Filter = *Null*, Subscription Geometry Type = *Polygon*, Spatial Operator = *Contain*, and Buffer Value = *zero*. For the subscription geometry, the officer selects the “T2N” FSA polygon and uses it to define the subscription geometry required. Finally, the officer submits his

subscription then he gets an acknowledgement message that his subscription has been registered successfully by the RFERS notification service.

The steps mentioned above to initiate new subscription have been performed by the author using the RFERS subscriber application. Meanwhile, random geospatial notifications of type “Fire Incidents Reports” have been generated and published to the *FireIncReports* topic using the simulation program designed for this, where the main piece of information encapsulated in every generated geospatial notification is the address text of the incident. The addresses that stated point locations outside the previous registered subscription’s geometry (*i.e.*, “T2N” polygon boundary) have not been delivered to the RFERS subscribe application. When “2500 University Drive NW”, Calgary, AB” address text published, the RFERS subscriber application received that geospatial notification and the address is located instantly over the base map. Figure 5.1 depicts the received geospatial notification of type “Fire Incident Reports” that encapsulated the address “2500 University Drive NW, Calgary, AB”. The officer then would explore the neighbourhood area and features to come up with the appropriate response action to the reported incident.

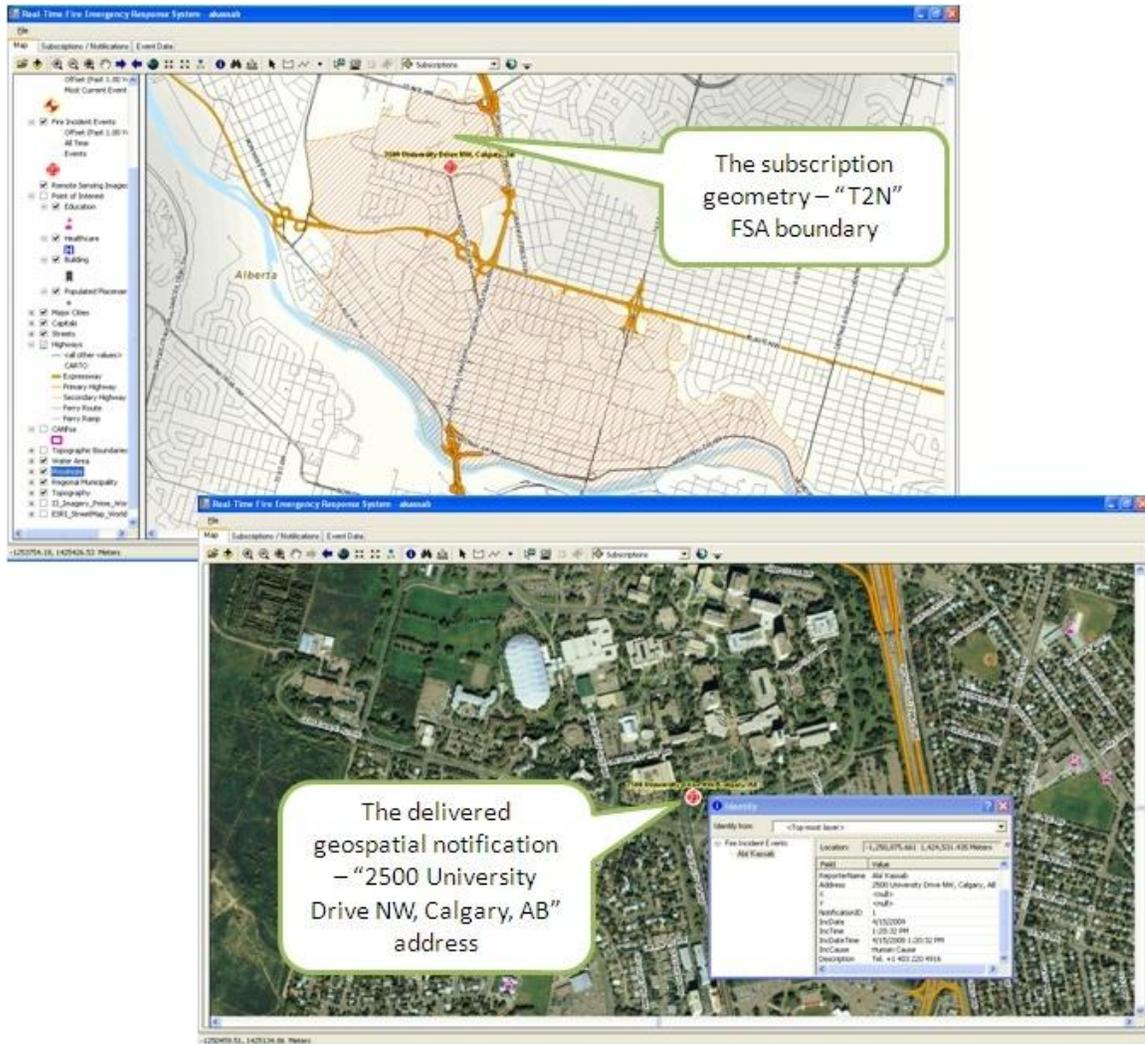


Figure 5.1: Subscription and publishing scenario utilizing the “FireIncReports” topic of the RFERS prototype

Let’s assume that in the example above the officer further requires to be notified about the positions of emergency vehicles within proximity of 3000m from the incident location. Accordingly, the officer initiates another new subscription and defines its parameters as the following: Topic Name = *EALocations* (the string name of “Emergency Asset Location” topic), Attribute Filter = *Null*, Subscription Geometry Type = *Point* (the fire incident location), Spatial Operator = *Contain*, and Buffer Value = *3000*. The

subscription geometry is defined as the point location of the reported fire incident. Figure 5.2 shows the area of this subscription for the “Emergency Asset Location” topic after submitting the operation to the RFERS notification service. During this time, it is assumed that several emergency assets are moving all over the area in different directions and they are constantly publishing their current positions throughout the system. For this matter, four instances of the publisher simulation program have been run simultaneously to broadcast fabricated coordinates of four moving assets, E1, E2, E3, and E4. The time interval has been set to 5 seconds in the simulation program instances, where in each time interval tick a geospatial notification of the current position of the asset is published to the “Emergency Asset Location” topic. While publishing the assets current positions, two of them, E2 and E4, entered the subscription region and the subscriber application started to receive geospatial notifications of the two assets positions. The rest of the emergency assets were driving away from the region of interest by the subscriber, thus their positions were not delivered to the subscriber application. The received geospatial notifications were shown instantly on the base map, where the consecutive positions of one asset are symbolised as a driving path and the latest position as a car symbol labelled by the asset name. Figure 5.2 shows received geospatial notifications of the two aforementioned assets.

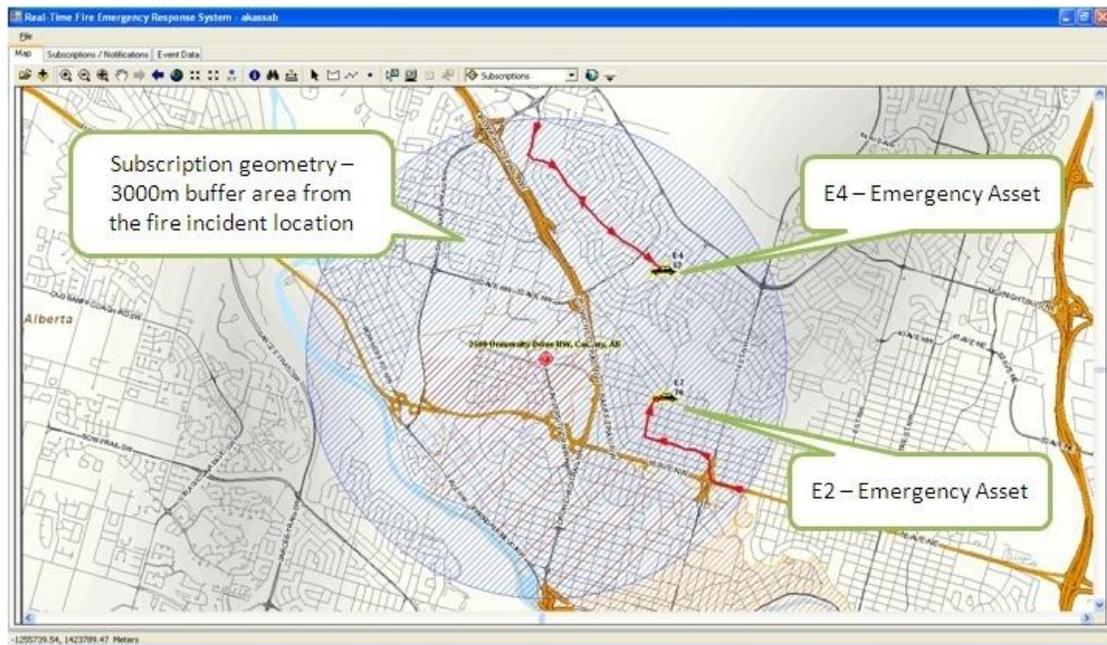


Figure 5.2: Subscription and publishing scenario utilizing “Emergency Asset Location” topic of RFERS prototype

5.2.2 High Temperature and Low Relative Humidity Observations of Wireless Sensors

In the context of this scenario, the correlation between wildfire hazards or ignition potential and meteorological conditions is significant and well known in the history. Wildfire danger tends to increase as a consequence of increasing temperature and decreasing relative humidity observations. Thus, it is of importance to get notified about the happening of this critical condition once observed in wildlife environment. This simulation scenario is planned to trial the RFERS prototype interaction mechanism in producing and consuming geospatial notifications of type “Wireless Sensor Observations” topic.

The simulation program was utilized to create 10 instances of “Wireless Sensor Observation” topic publishers. Each one of the instances referred to one simulated wireless sensor. The geospatial notifications encapsulated random values of temperature

and relative humidity observations ranging from 20°C to 30°C and from 5% to 20%, respectively. The time interval for publishing a single geospatial notification by one sensor is set to 10 seconds. The spatial locations of the simulated wireless sensors were distributed all over Jasper National Park area located in Western Alberta, Canada.

In the subscriber application side, a subscription operation was performed to consume wireless sensor observations whose temperature and humidity values are greater than 27°C and less than 8%, respectively. At the same time, the observations of interest were only those located within Jasper National Park boundary. Accordingly, the subscription's parameters were assigned as the following: Topic Name = *WSObservation* (the string name of "Wireless Sensor Observations" topic), Attribute Filter = *Temperature > 27 AND Humidity < 0.08*, Subscription Geometry Type = *Polygon* (Jasper National Park boundary), Spatial Operator = *Contain*, and Buffer Value = *zero*. The Jasper National Park polygon was selected from the base map data and used as the spatial geometry of this subscription. The subscription then was submitted and registered successfully (see Figure 5.3).

A while from registering the subscription, the sensors' observations started to show on the base map. The observation values kept updating on the map as long as the simulated sensors publish geospatial notification matching with the previously registered subscription. To ensure that the observations received are only the matched ones with the registered subscription, the content-data of the received geospatial notification were checked out by the author and found they correctly matched the registered subscription. Figure 5.3 illustrates the subscriber application resulting from applying this scenario.

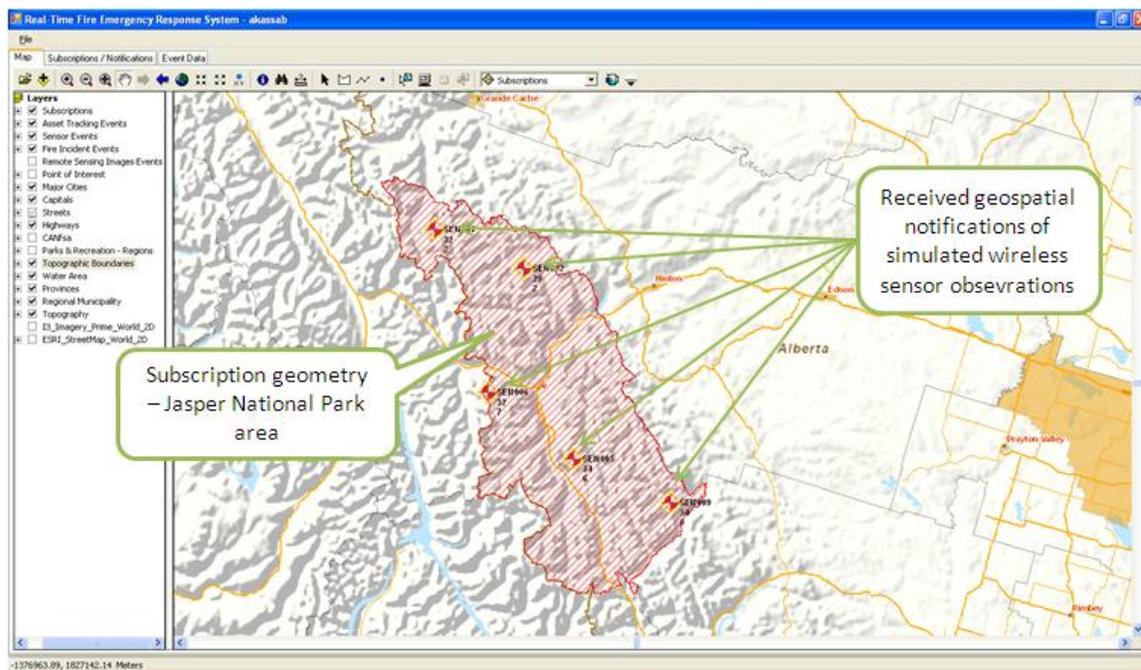


Figure 5.3: Subscription and publishing scenario utilizing “Wireless Sensor Observations” topic of RFERS prototype

5.2.3 Wildfire Monitoring Using Remote Sensing Thermal-Imaging

The aim of this scenario is to ensure that transacting geospatial notifications by means of the “Wildfire Remote Sensing Images” topic functions and meets the requirements of RFERS prototype. The geospatial notifications publisher this time is assumed publishing imaging datasets captured by an airborne camera at the scene of a wildfire. The imaging datasets should be in their final form, in other words, previously processed and ready to be visualized and mapped in a GIS before publishing them throughout RFERS. Temporal flight lines or over passes can be conducted by the aircraft to monitor the spread and the growth of the fire. The subscriber in this scenario is concerned to be notified about the imaging datasets in real-time providing sufficient information for responding and preventing more loss and damage of natural resources.

The USDA Forest Service, Pacific Southwest Research Station (PSW) in California, USA, developed an airborne thermal-infrared imaging system, FireMapper, to monitor the progress and the intensity of wildfires and support tactical fire suppression by providing temporal thermal-infrared imageries of current wildfire events (Riggan *et al.* 2003). The captured imageries data is processed and georeferenced, the final products then are made widely available on their website (2009) for downloading in different data formats. For this scenario, nine datasets of temporal thermal-imageries were downloaded. The datasets were captured and made available on the Internet while monitoring the Corral Fire event that took place in Los Angeles County, CA, USA during November 2007. The data format of those imageries is GeoTIFF, where georeferencing information of the imageries is embedded within the data files. The images were encapsulated in the content-data of geospatial notifications and published to the “Wildfire Remote Sensing Images” topic using the publisher simulation program. On the other side, a subscription was previously performed to receive all published geospatial notifications to the “Wildfire Remote Sensing Images” topic. In other words, there are no constraints or filters applied on the content-data. Consequently after executing the publications, the images are visualized directly on the base map showing the active fire spreading and the temperature intensity classes. As further geospatial notifications are published, the images would overlap each other and the most recent image would appear on the top level. Figure 5.4 shows the subscriber application resulting from applying this scenario.

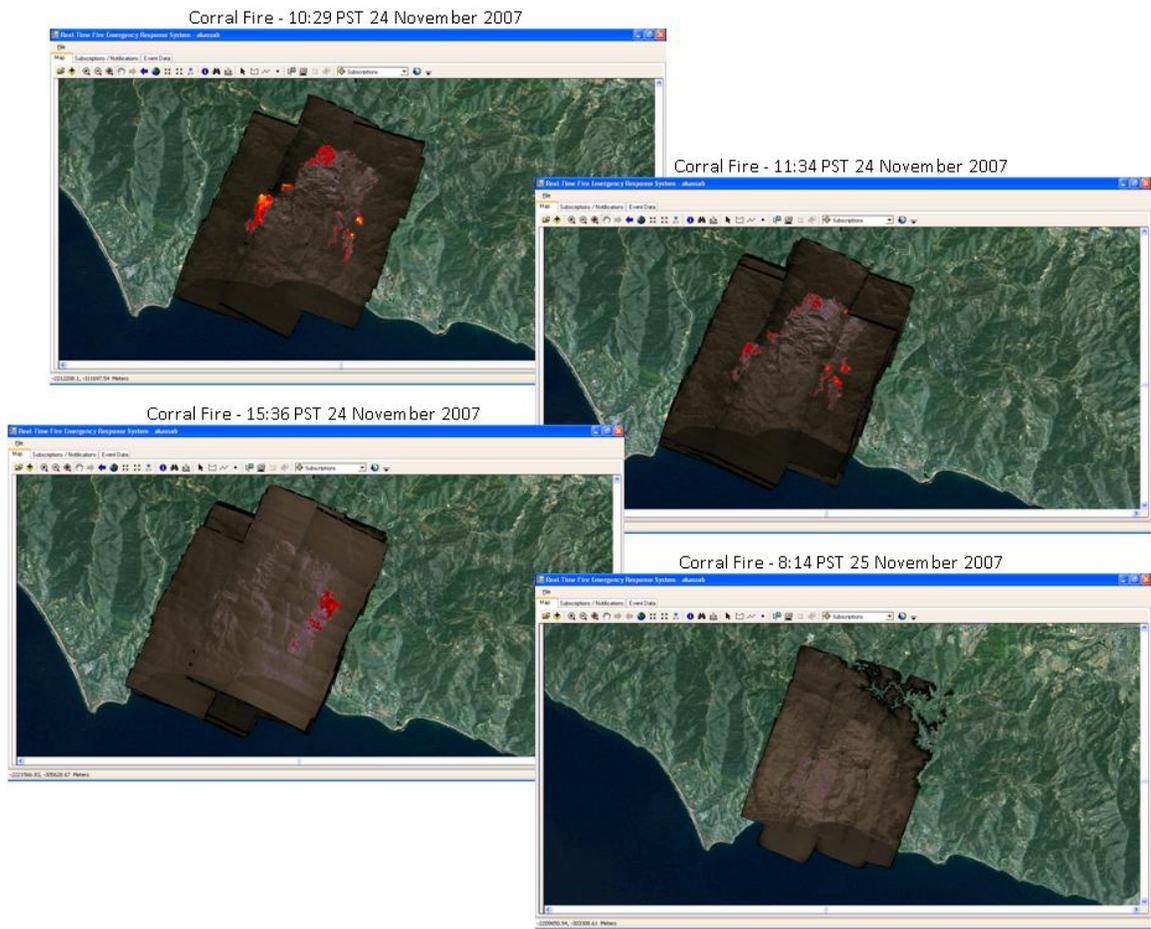


Figure 5.4: Publishing and subscription scenario utilizing “Wildfire Remote Sensing Images” topic of RFERS topic

5.2.4 Discussion

Throughout the aforementioned scenarios and other software demonstrations and experiments conducted, the results of the RFERS prototype testing process established that the RFERS software meets its requirements and the provided functions are working properly. However, the results also have shown major limitations and problems associated with the prototype implementation and the publish/subscribe interaction methodology applied. The rest of this section emphasizes these issues.

Heavy server load and single point of failure: the centralized architecture of notification service (see Section 2.4.1) is assumed in the proposed geospatial publish/subscribe model, as mentioned in Section 3.4, and realized in the implementation of the RFERS prototype. This introduces a major limitation in terms of heavy load expected on the server unit since all the publications and subscriptions are handled by the centralized matching engine. Further, the interaction between clients is all dependent on the correct functioning of the server unit, in particular the matching engine; if the server fails, the whole system breaks down. On the other hand, this architecture is simple to implement and can be efficient and well suited for relatively small-scale businesses.

The solution of this problem would be by realizing the distributed architecture of the notification service (see Section 2.4.2). The distributed architecture increases the scalability of the system in accommodating globally distributed clients. Nevertheless, it also introduces new difficulties in implementing the matching engine and brings in new algorithms that should be recognized in routing the operations between the notification service's nodes.

The centralized architecture fits the objectives of this research. The comparison between different notification service architectures is beyond this research scope. Realizing the distributed architecture with the proposed geospatial publish/subscribe model is one of the intended future works for this research (see Section 6.3).

Availability status of subscribers: the application provided for RFERS subscribers receives the geospatial notifications delivered from the notification service and visualize the data on the base map automatically, as shown in the testing scenarios of Section 5.2. However, what would happen to the geospatial notifications data in case a subscriber is

not available on the RFERS network at the time of dispatching? This question was investigated and taken into considerations in the RFERS prototype implementation. When a subscriber is not running the front-end application, the notification data will be stored temporally inside the notifications service. Therefore, as soon as the subscriber logs in to the RFERS network, the original notifications data will be dispatched to the subscriber. Technically, this issue is addressed in the RFERS prototype by implementing a method called *createDurableSubscriber()* available in the EMS API. Hence, the geospatial notifications will not be discarded even if the subscriber is inactive. Furthermore, the messages of the geospatial notifications can be assigned an additional attribute, message expiration, indicating a period of time to keep the message alive. Thereby, the message will be destroyed and not delivered if the current passes the expiration value of the message. Using this property would maintain the resources of the notification service unit in case many interested subscribers happen to be inactive for a long time.

Visualization of received geospatial notifications: the spatial locations of geospatial notifications are highly dynamic and unpredictable. Visualization of newly added spatial features of the geospatial notifications requires continuously refreshing the base map in order to screen the new features and symbols. The traditional technique of refreshing the map screen after receiving new data is either to refresh the whole base map, which means all the existing GIS layers, or refreshing only the GIS layer where the new spatial features are stored. The former method is inefficient as all the data have to be redrawn every refreshing time thus wasting the computation resources. The latter can be considered in cases where the update frequency on the dynamic layer (*i.e.*, the geospatial

notifications data layer) is relatively low. However, both methods potentially burden the visualization of the spatial data when high volume of geospatial notifications is received and affect the overall performance of the subscribers' applications.

To refine the visualization performance, an advanced drawing method of dynamic spatial data is applied to avoid refreshing unchanged background areas of the base map. This functionality is provided by the programming libraries of ArcGIS Engine 9.3, the method is called *PartialRefresh* (a detailed documentation is available online¹). This method allows the map view to redraw partial areas of spatial data and work with the display caches in order to make the drawing process quicker and more efficient. This mechanism is implemented in the RFERS subscriber application. The internal computation of the application is developed in a way to redraw only the local area surrounding the new incoming geospatial notification. This has contributed significantly in minimizing the drawing time of the map data and thus enhancing the application performance. Although the visualization performance has been greatly enhanced utilizing the above method, this problem still exists to a certain extent in the RFERS subscriber application. This issue is addressed as part of future work to be further investigated and developed for this research.

Windows-based desktop subscriber application: the subscriber application developed for RFERS prototype is Windows-based desktop software. This implies that the user has to setup and configure the subscriber software on a local computer with Windows operating system and also meet or install other software requirements in order

¹ ESRI Developer Network (EDN), 2009, "IActiveView.PartialRefresh Method", Accessed April 10, 2009, http://resources.esri.com/help/9.3/ArcGISEngine/ArcObjects/esriCarto/IActiveView_PartialRefresh.htm

to run the application. With the advent technology of web-based applications, it is highly recommended to replace the subscriber desktop software with a web-based application. This entails many advantages for the users of RFERS, which mainly include: (1) the system becomes platform-independent; users do not need certain operating systems to interact within the RFERS, the only thing they need is a browser, which is available for everybody, and an internet connection, (2) there is no special configuration required to run the application, (3) accessibility for users is available anytime and anywhere, and (4) it is easier to add new functionalities and update or improve the system performance. Nevertheless, the desktop application also has its advantages over the web-based application in terms of security, performance, dependency on the internet connection, easy development, and others. The comparison between desktop- and web-based applications is still a subject of debate beyond this research scope.

In the context of this research, developing a web-based application for RFERS subscribers would be a great asset to the research. On the other hand, meeting the requirements of the interaction between web application clients by means of geospatial publish/subscribe model is highly challenging. The interaction style in most of the web applications relies on “pulling” the data from a provider or a source of information by a request/reply operation. Thus, if a client needs to be updated in near real-time of the new information, the client has to pull the provider periodically which is an inefficient mechanism for the overall system performance. In contrast, the “push” style can be considered to overcome the problems originated from the “pull” style. Instead of clients originating requests to the data provider, the data provider server pushes the data to the clients upon receiving new or updated information. Adopting the “push” style in web

applications is promising and potentially pertinent to the mean of the publish/subscribe model. However, our investigations have shown that applying this style takes time and effort considering the time constraints set to this research. Integrating web applications in the RFERS publish/subscribe interaction is intended in the future work of this research.

User-interface: although the user interface of the subscriber application is designed in a way regular computer users would be familiar with, it requires more improvement to make it easier and simpler. Training prior to using the software or creating a user manual can be considered to assure the proper usage of the RFERS prototype.

Dependency on commercial software packages: ArcGIS Engine 9.3, ArcGIS Server 9.3, and TIBCO EMS software products and their .NET framework APIs are required to deploy and run the RFERS prototype components. This places some constraints on the developed system from being an independent product. The prototype is meant to be a proof of concept and not a commercial product, as mentioned in Section 4.3. The aforementioned commercial products have been utilized to facilitate the implementation of the prototype as they provide high-level functions that abstract the complexity of dealing with low-level objects and controls. The proposed model, geospatial-based publish/subscribe, and the design of the RFERS can be implemented regardless the development framework or the software used. In this regard, open source software and data format can be potentially employed to reduce the deployment cost of the system. This issue needs more investigation and it is addressed in the future work of this research.

5.3 Performance Evaluation

This section introduces the second part of this chapter, namely evaluating the performance of the geospatial-based publish/subscribe interaction using the implemented RFERS prototype. As mentioned early in this chapter, the efficiency of the RFERS prototype in integrating distributed clients and disseminating geospatial events, so do publish/subscribe systems in general, mostly rely on the performance of the middleware (*i.e.*, the notification service). This is not surprising due to the fact that most of the communication processes, particularly the matching engine, are executed inside the middleware core. In geospatial-based publish/subscribe, the process of matching geospatial notifications against registered geospatial subscriptions is the most resources consuming. The speed of delivering geospatial events to interested clients largely depends on the matching speed. Achieving high matching speed, thus high performance, means quick awareness of dynamic geospatial events, potentially crucial, which in turn accelerates decision-making and response actions in emergency situations.

This section attempts to evaluate the performance of the proposed geospatial notifications matching approach detailed in Section 3.6.3 and implemented in the RFERS middleware (see Section 4.2.3). The next sub-sections describe the evaluation process in details.

5.3.1 Evaluation Metric

The metric selected to evaluate the performance of the RFERS middleware is the *matching time*, which is the processing time that the matching engine takes to find matched subscriptions (or subscribers) upon publishing geospatial events. The matching time gives an indication on how fast published geospatial notifications can be

disseminated to subscribers. The delivery time, which is the time consumed from the moment of publishing a geospatial notification to the moment the geospatial notification triggers the subscriber application, can be also considered as an evaluation metric.

However, it is hard to measure the delivery time accurately as it is dependent on the network technology and connection speed used in transferring information, the network distance between the middleware server and the clients' computers, the size of the transferred data, and other factors which are difficult to control. Thus, it does not give a realistic measure of the matching engine performance.

In evaluating the performance of the RFERS prototype, the matching time is measured in different scenarios in an attempt to see how the performance of the matching engine changes with changing the number of registered subscriptions and spatial indexing used to structure the subscriptions data. The experiments are shown in Section 5.3.3.

5.3.2 Simulation Environment

In this evaluation process, the same computer setup applied in testing the functionalities of RFERS prototype (see Section 5.2) is used here. A server workstation with 2.4GHz Intel Core 2 Quad-Q6600 CPU and 4GB RAM running Windows XP is used to deploy the RFERS middleware and the database, and a local computer with 1.86GHz Intel Core 2 Duo E6320 CPU and 2GB RAM also running Windows XP is employed to run the GIS subscriber application and publisher simulation programs.

For simulating geospatial subscriptions, the FSA 6-digits postal code boundaries (*i.e.*, polygon features) for Alberta, Canada are used to represent the base geometries of registered geospatial subscriptions. The postal code polygons can be good representation of potential geospatial subscriptions that would be performed in real cases. Because they

give realistic spatial distribution of populated areas, the interests of subscriber clients may be more focused to monitor geospatial events in those regions. Moreover, the irregular shape of these polygons gives more practical measures of the matching performance rather than using rectangular or regular shapes of all the subscriptions, which eases the matching process. The polygon feature class of Alberta postal codes originally contains almost 100,000 features. In real cases, the base geometries geospatial subscriptions can overlap with each other as multiple subscribers may register their interests in same regions, thus, their subscriptions tend to overlap. To reflect this nature, the postal code polygons are replicated and slightly shifted to create dense, larger numbers and overlapped polygons which are used as geospatial subscriptions. Figure 5.5 shows a portion of the simulated polygons feature class. The final geospatial subscriptions dataset contains around 1,000,000 polygon features.

Satisfying the spatial constraints contained by geospatial subscriptions is the main concern in the proposed geospatial-based publish/subscribe interaction. Although the geospatial subscription language and the matching process proposed in this research (see Section 3.5.2 and Section 3.6.3.3 respectively) support attribute constraints, enhancing the matching process in matching attribute type of constraints is not of the interest in this research. In our experiments herein, therefore, geospatial subscriptions that have only spatial constraints are considered. To simplify the evaluation process, all the geospatial subscriptions are considered assigning a spatial filter with a *polygon* base geometry (the created postal code geometries), *Contain* spatial operator, and *zero* buffer value. The spatial indexing technique used in structuring the geospatial subscriptions is the Multi-Grid indexing technique (see Section 4.3.4).

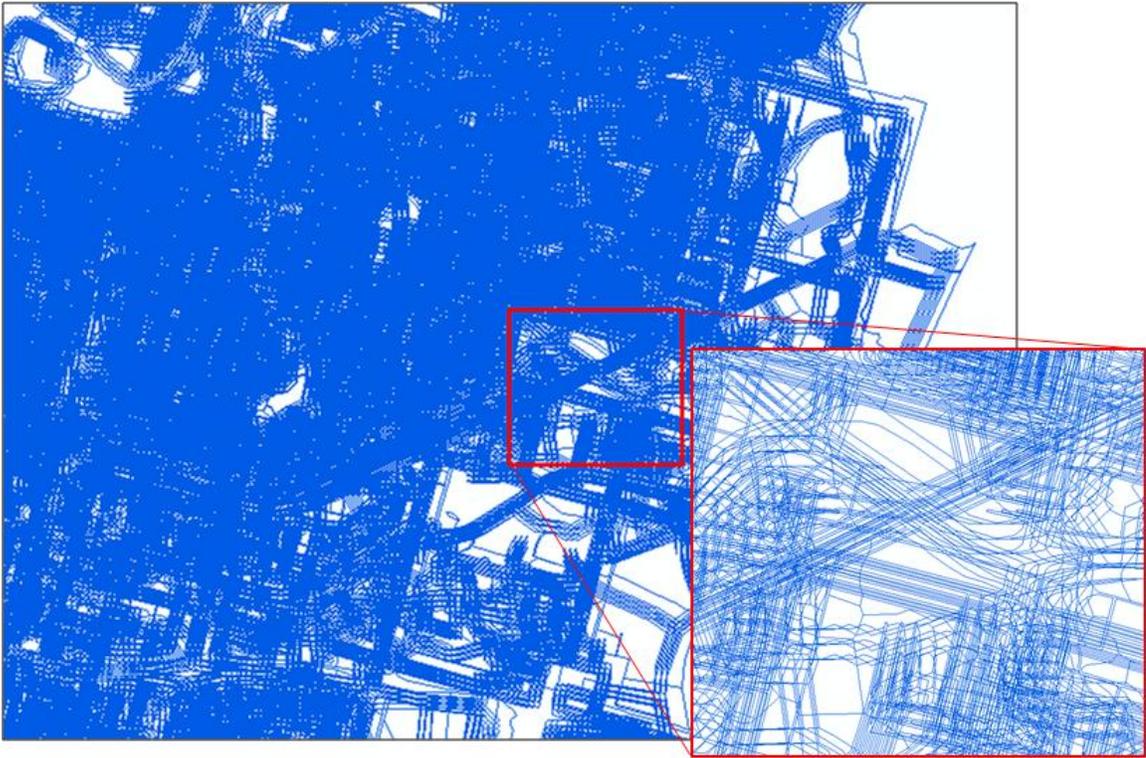


Figure 5.5: Simulated geospatial subscriptions dataset

5.3.3 Experiments

There are three main experiments conducted to evaluate the performance of the proposed matching engine in RFERS middleware. First, the proposed matching approach, implemented in the RFERS notification service, is evaluated by measuring the matching time with increasing the number of geospatial subscriptions. The second experiment is intended to compare the proposed matching approach with the brute force matching approach to understand the efficiency that can be achieved by using the proposed matching approach over the naive solution (*i.e.*, brute force matching). The third experiment is intended to see the effect of the spatial indexing of the matching process. Thus, the matching time with and without indexing the geospatial subscriptions is measured. The next sub-sections describe these experiments in details.

5.3.3.1 Number of Geospatial Subscriptions

This experiment is set up to assess the matching time using the proposed matching approach with increasing the number of registered geospatial subscriptions. The geospatial subscriptions dataset, described previously in Section 5.3.2, is used to extract several subsets of geospatial subscriptions. For instance, to create a subset of 100,000 features, this number of features is selected *randomly* from the geospatial subscriptions dataset and extracted to create a new subset. After extracting each subset, the subset's polygon features (*i.e.*, geospatial subscriptions) are uploaded and prepared in the RFERS subscription database. Afterwards, the spatial indexes created on the geospatial subscriptions tables are updated to maintain the new addition of geospatial subscriptions subset. The subset's geospatial subscriptions then are ready to be matched with published geospatial notifications. The matching time is measured after uploading each of the subsets.

Geospatial subscriptions in this experiment are assumed to be registered in the “EALocations” topic (*i.e.*, Emergency Asset Locations topic). As mentioned before in Section 5.3.2, geospatial subscriptions encapsulate spatial filters (*i.e.*, *polygon* base geometry, *Contain* spatial operator, and *zero* buffer value) and are without attribute constraints. The simulation programs are utilized to generate and publish random emergency assets locations to the “EALocations” topic. Around 1000 geospatial notifications are published to this topic (100 notifications per second) and the matching time is measured for each notification. With each geospatial subscriptions subset, the average matching time to match a single geospatial notification is calculated. Figure 5.6

shows the relation between the average matching time and the number of registered geospatial subscriptions.

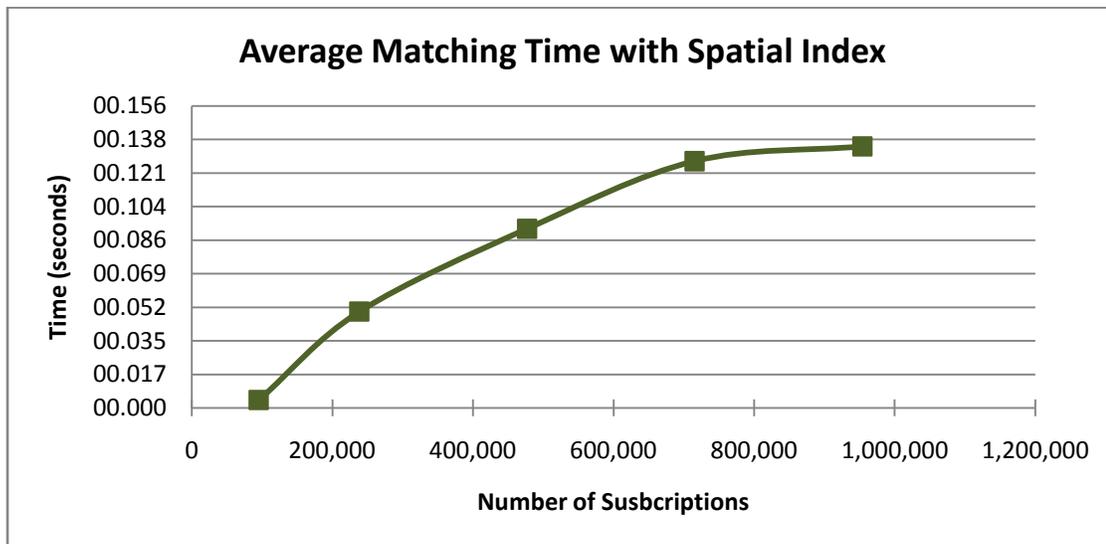


Figure 5.6: Average matching time of a single geospatial notification using the proposed matching approach

As can be seen in the graph, matching the number of simulated geospatial subscriptions ranges from 100,000 to 1,000,000 features. The matching process takes around 4 milliseconds to match one geospatial notification with 100,000 subscriptions. While in matching 100 geospatial notifications with 1,000,000 subscriptions, the matching time is around 135 milliseconds. The relation shown in the graph between the number of subscriptions and the matching time tends to be a logarithmic curve. The increasing ratio of the number of subscriptions is getting larger than the increase ratio of the matching time, which means, the performance is getting higher proportionally with a larger number of subscriptions. This relation is resulted from the effect of the spatial indexing in structuring the subscriptions, which leads to better performance in the matching process.

5.3.3.2 Brute Force versus the Proposed Matching Approach

In this experiment, the brute force method (see Section 3.6.2) is compared with the proposed matching approach.

The matching procedure using the brute force method functions as the following steps. First of all, all registered geospatial subscriptions are previously stored in one table inside the RFERS database, where each record refers to one geospatial subscription. Upon publishing a geospatial notification, the matching process takes the first geospatial subscription in the table and parses its constraints. Second, the matching process buffers the subscription geometry according to the assigned buffer value. Third, the assigned spatial operator is used to evaluate the buffered geometry of the subscription against the geometry of the geospatial notification. Finally, if a match is found, the geospatial notification is delivered to the owner client of this subscription. The matching process continues to match the next subscription in the table until all the subscriptions are evaluated.

Similar to the previous experiment, several subsets of geospatial subscriptions are created and both matching approaches are run to measure the matching time in each subset. The geospatial subscriptions in this experiment are also registered in the “EALocations” topic and the simulation program runs to publish random geospatial notifications to the same topic through the network. Around 500 geospatial notifications are published, 100 notifications per seconds, and the matching time is measured for each notification. Finally, the average matching time is calculated in each geospatial subscriptions subset. Figure 5.7 shows a graph of the average matching against the number of subscriptions registered in each trial.

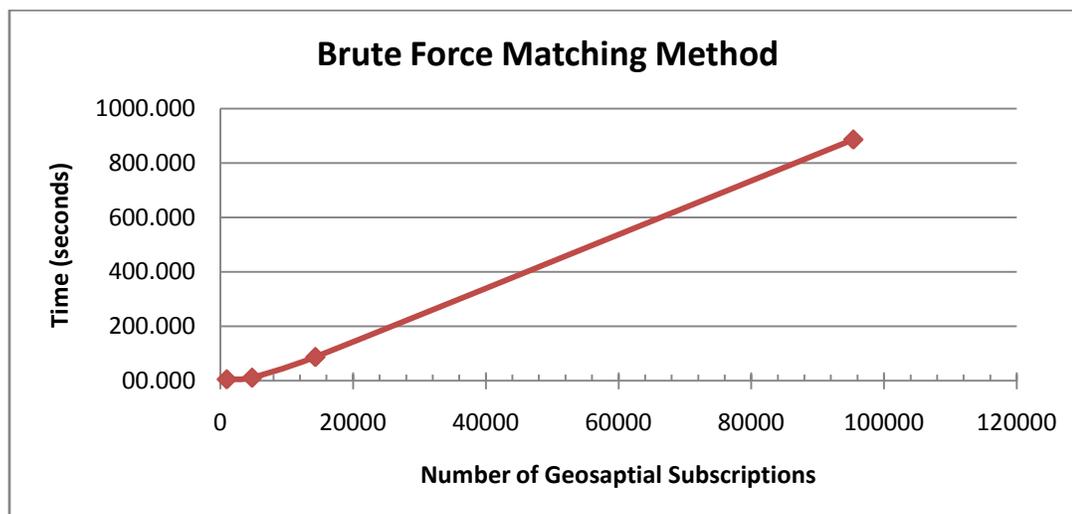


Figure 5.7: Average matching time of a single geospatial notification using the brute force method

Using the brute force method, the average time of matching one geospatial notification with 1,000 subscriptions is around 5 seconds (*i.e.*, 5000 milliseconds). With 100,000 subscriptions, the average matching time is approximately 900 seconds. As can be seen in the previous graph, the matching time keeps increasing with larger numbers of geospatial subscriptions. The results from the previous experiment (see Section 5.3.3.1) have shown clearly that the proposed matching approach outperforms the naïve matching solution. The matching time results that are achieved by using the proposed matching approach in this experiment were close to small fractions of a millisecond. The brute force method consumes relatively large time in the buffering process of the subscriptions' geometries during the matching. This issue is mainly what causes the matching process using the brute force method to take huge time. In this experiment, the naïve solution spends almost two hundred thousand times more to process a geospatial notification with 100,000 subscriptions than the proposed matching approach.

5.3.3.3 Effectiveness of Spatial Indexing

This experiment tests the effect of using spatial index to structure geospatial subscriptions on the matching process. Here the proposed matching approach is examined twice: in matching geospatial notifications with spatially indexed geospatial subscriptions, and the second time is in matching geospatial notifications without indexing geospatial subscriptions.

The simulated geospatial subscriptions dataset (see Section 5.3.2) is used to create several subsets with different number. Each subset is stored in the RFERS subscriptions database. Like the previous experiments, geospatial subscriptions are registered in the “EALocations” topic. Geospatial notifications of the same topic are published by the simulation program; almost 1000 points are published in a speed rate of 100 points per second. The matching process then is executed twice on each of the geospatial subscriptions subsets: one with indexing the geospatial subscriptions and another without using index. Figure 5.8 graphs the relation between the average matching time of a single geospatial notification and the number of geospatial subscriptions and the effect of using spatial indexing in the matching process.

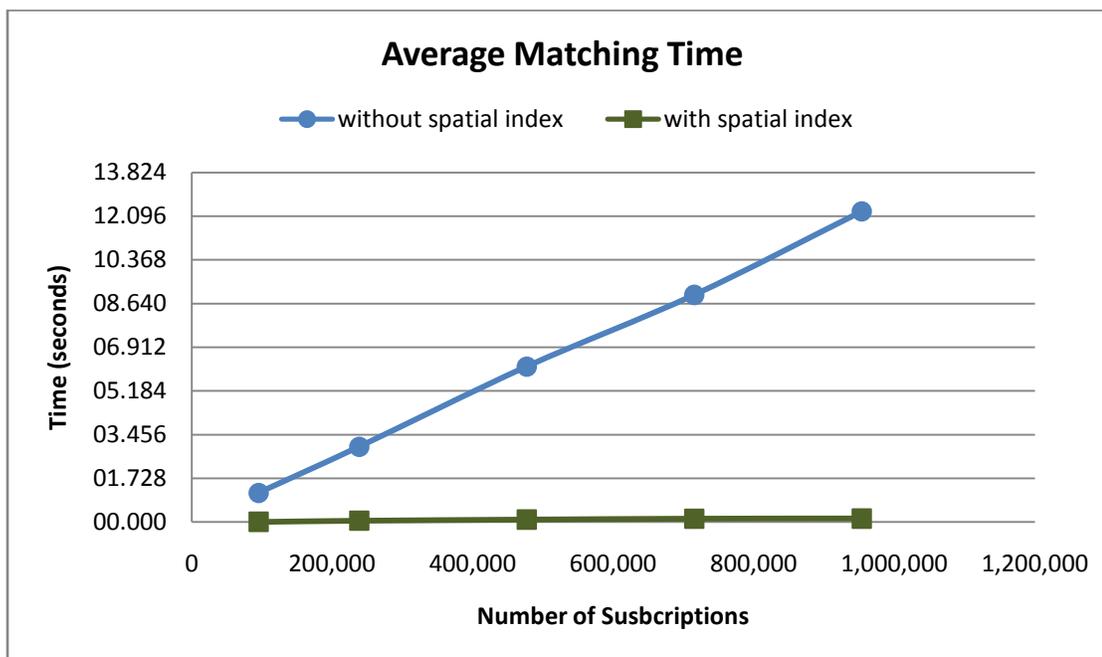


Figure 5.8: The effect of using spatial index on the average matching time

As can be seen from the previous graph, using spatial index to structure geospatial notifications greatly increases the performance of the matching process. Without indexing, the matching time tends to scale linearly with increasing the number of registered geospatial notifications. A spatial index can be built initially on the geospatial subscriptions table that is stored inside the RFERS database. Inserting new features to the geospatial subscriptions table, however, requires maintaining the spatial index to achieve high performance in the matching process. Maintaining the spatial index may consume some processing time. Nevertheless, spatial indexes can significantly leverage the matching process. Furthermore, rebuilding the spatial indexes can be managed cautiously to avoid repeated processing of the indexes, thus, resources can be saved.

5.3.4 Discussion

The performance results conducted throughout the previous experiments established that the matching approach, proposed in the research (see Section 3.6.3.3), performs well in the context of geospatial-based publish/subscribe. Evidently, this approach outperforms the naïve solution in matching geospatial notifications with geospatial subscriptions. The use of spatial indexing in structuring and matching the registered geospatial subscriptions greatly improves the RFERS notification service function in the interaction. In summary, our experiments show that the matching engine implemented in the RFERS prototype will be capable to match 100 geospatial notifications against 1,000,000 geospatial subscriptions in almost 14 seconds. The previous experiments, however, also show a major limitation in the implemented matching process. The following describes this issue and how it can be more improved.

Processing of published geospatial notification: the matching process implemented in the RFERS prototype evaluates a single geospatial notification at a time with the registered geospatial subscriptions. In other words, if a collection of geospatial notifications are published about the same time to the server unit, they will be kept in order inside a data queue. They will then be processed sequentially by the matching engine; the first geospatial notification added to the queue is the first to be processed and removed, this data structure is called First-In-First-Out (FIFO). The sequential processing of the geospatial notifications was a point of criticism. It is suggested that the geospatial notifications should be processed simultaneously regardless of their publishing order, thus accelerating the dispatching time of geospatial notifications. This can be achieved by using a number of threads (*i.e.*, thread pool) that can be created to perform the matching

process for a collection of geospatial notifications in parallel mechanism. The number of threads can be tuned and dynamically determined based on the number of waiting geospatial notifications. Although the thread pool technique can be used to enhance the matching process performance, it introduces additional implementation challenges to the RFERS prototype as it requires a solid background on multi-threading programming and parallel computing. Studying these issues is beyond the scope of this research work and can be considered in future research work and development.

5.4 Summary

This chapter discussed the procedure made to test the actual implementation of the RFERS prototype. Further, the chapter also evaluated the performance of the RFERS prototype, particularly, the efficiency of the developed matching engine of the notification service middleware.

The RFERS prototype was tested through conducting simulated interaction scenarios and several demonstrations of the system. The testing procedure showed that the implementation of the RFERS software meets the desired requirements. The feedback acquired from demonstrating the system helped in enhancing the implementation.

The performance of the matching engine in RFERS was evaluated by simulating the matching process with different numbers of geospatial subscriptions. The matching time was measured and considered as an indication of the efficiency of the matching process; the lower the matching time, the faster the matching process, thus, the more efficient the system would be. The results established that the proposed matching

approach greatly outperforms the naive solution. Moreover, the use of spatial indexes significantly improves the matching process.

Chapter Six: Conclusions and Future Work

6.1 Introduction

This final chapter concludes this research by summarizing the work that has been done and outlining the conclusions drawn out of this research work. It also comments on the limitations and proposes areas for future work.

This research has developed Geospatial-based Publish/Subscribe, an interaction framework based on publish/subscribe style to transact dynamic geospatial events in the context of heightened situational awareness in fire emergencies. Loosely-coupled, heterogeneous, and autonomous distributed clients communicate by means of this interaction framework in producing and/or consuming geospatial events asynchronously and in real-time. Moreover, Real-time Fire Emergency Response System (RFERS) prototype has been implemented in this research as a proof of concept to evaluate the adequacy and the efficiency of the interaction.

To summarize, Chapter 1 provided a brief introduction about the topic of this research and outlined this research problem and the key objectives. Chapter 2 reviewed terminologies, system components, interaction models, and processing algorithms used in the literature of publish/subscribe interaction systems. Chapter 2 also presented some major works that have been done in this field and investigated several related research. Chapter 3 proposed Geospatial-based Publish/Subscribe interaction framework that is developed in this research to transact and disseminate geospatial events between distributed clients. Chapter 3 defined the data models used to encapsulate geospatial semantics in performing publish/subscribe operations and also proposed an efficient matching approach that can be utilized to enhance the efficiency of the notification

service, thus, increasing the performance of the system interaction. Chapter 4 discussed the development of RFERS prototype realizing the proposed geospatial-based publish/subscribe in integrating distributed clients. Chapter 4 also discussed the design of the system and the implementation of the prototype software components. Chapter 5 presented simulation scenarios used to testify the sufficiency of the provided functions by the RFERS prototype. Also, the chapter evaluated the performance of the underlined interaction and the matching process of geospatial events with geospatial subscriptions that potentially scale to large numbers.

6.2 Conclusions and Limitations

Emergency agencies seek to maintain situational awareness and effective decision making by continuous monitoring of and real-time alerting about sources of information regarding current incidents and developing fire hazards. The nature of this goal requires integrating different types, potentially numerous, sources of dynamic geospatial information from one side and large number of clients having heterogeneous and fine-grained interests in data from the other side. The traditional request/reply communication style may function inefficiently in such scenarios as it is based on point-to-point, synchronous and pulling mode interaction between consumer clients and information providers/services. Publish/subscribe interaction style has leveraged many application to alleviate the shortcomings of the traditional request/response by providing many-to-many, asynchronous, real-time, and intelligent interaction between distributed clients.

In publish/subscribe systems, publisher and subscriber clients communicate by producing and consuming events, respectively. An event can be given many semantics to

represent an instantaneous happening of interest. A notification represents the actual message that encapsulates attributes data describing the associated event. The publish/subscribe middleware (*i.e.*, the notification service) acts as a mediator by routing notifications from publisher clients to interested subscriber clients in timely manner. The middleware takes the charge of the information dissemination which makes distributed clients loosely-coupled in the interaction and leads the system to extend efficiently over wide-area of networks. In the context of fire emergency, events can be given geospatial semantics and used to represent sudden occurrences or highly-dynamic changes of features' states that are related to the geographic space. Although the literature has shown an exhaustive understanding of publish/subscribe systems, there were few research tackling the issue of accommodating generic geospatial semantics in the publish/subscribe interaction. This issue has arose the need to develop an extended publish/subscribe interaction model and incorporate geospatial events in the communication framework.

Geospatial-based Publish/Subscribe has been proposed in this research for two main objectives: the ability to encapsulate geospatial representations of events, called geospatial events, in the clients' publications and offering subscribers an extended expressiveness to define their spatial interests in receiving geospatial events. Consequently, a geospatial notification data model has been proposed to construct geospatial notifications. A single geospatial notification is a composition of two data components: a spatial component and an attribute component. The spatial component is a name/value pair that represents the shape and geometry of the associated event. The attribute component composes one or set of name/value pairs with primitive data types,

such as: numeric, textual, binary, date/time and others. The focus here is on the spatial component and it is considered the primary representation of geospatial notifications. The spatial component holds XY coordinates pair(s) as a geographic representation of geospatial events, including: a point, a polyline, a polygon or multi any one of these types. Publisher clients can use this data model to publish geospatial notifications that could be of interest to other subscriber clients. A geospatial subscription data model has been proposed to enable subscribers specifying not only attribute constraints but also spatial type of constraints. A geospatial subscription composes two types of predicates: a spatial predicate and an attribute predicate. Subscribers use spatial predicates to select geospatial notifications that satisfy certain spatial relationship by defining a base geometry (*e.g.*, point, polyline, and polygon), a spatial operator (*e.g.*, *Contain*, *Overlap*, *Cross ...etc*), and a buffer value. This definition of spatial predicate provides a wide variety of spatial constraints to express fine-grained spatial interests. The attribute predicate in this data model supports defining constraints of the content attribute data of geospatial notifications by means of comparison and logical operators (*e.g.*, =, <, ≥, AND, OR...etc). As publishers and subscribers perform geospatial-based publish/subscribe operations, the middleware handles published geospatial notifications and disseminates them among subscribers according to their geospatial subscription criteria. Matching geospatial notifications with geospatial subscriptions is a primary process executed within the notification service core. It determines the flow of information. The efficiency of the interaction largely relies on the performance of the matching process. This research has proposed an efficient matching approach by pre-processing of geospatial

subscriptions into clusters and using spatial indexes to structure the subscriptions and accelerate searching for matched ones with published geospatial notifications.

The proposed geospatial-based publish/subscribe interaction has been realized for developing Real-time Fire Emergency Response System (RFERS), a system prototype for transacting geospatial events essential in the context of fire emergencies. Four data models (*i.e.*, topics) have been designed in the prototype, these topics are: Emergency Asset Locations, Wireless Sensor Observations, Fire Incidents Reports, and Wildfire Thermal-Infrared Images. Publishers and subscribers are assumed to have previous knowledge about the RFERS topics as they perform the required publish/subscribe operations following the data structure of these topics. Three components have been implemented for the RFERS prototype: (1) a desktop GIS application for subscribers to facilitate performing geospatial subscriptions and visualize received geospatial notifications in a mapping environment, (2) simulation programs to simulate publishing geospatial notifications into the RFERS topics, and (3) the notification service middleware where the proposed geospatial notifications matching approach has been realized in implementing this component. TIBCO v4.4 and ArcGIS v9.3 software products have been utilized with C# programming language for the implementation.

The RFERS prototype has been tested in two phases. First, several interaction scenarios using the RFERS topics have been carried out to test the requirements of the implemented prototype functionalities. Second, the performance of the matching engine implemented in the RFERS middleware has been evaluated with simulated data. The matching time has been selected as a performance measure; the lower the matching time, the faster the matching process is, the better the interaction performance would be. The

proposed matching approach has shown satisfactory performance in matching geospatial notifications with geospatial subscriptions; 100 geospatial notifications can be matched against 1,000,000 geospatial subscriptions within 135 seconds. It has proven the use of spatial indexes significantly enhances the performance of the matching process.

The development of this research work is limited in several issues which will be discussed now in this section.

The first is regarding the scope of the proposed geospatial-based publish/subscribe interaction framework. The aim here is to extend the expressiveness of the subscription language by adopting geospatial semantics in the data models of events and subscriptions. The definitions proposed for geospatial events and geospatial subscriptions are limited to accommodate simple geographic representations and simple spatial relationships, respectively. Designing data models to accommodate more complex relationships needs more investigation and is intended for future work. Moreover, the matching process has been improved using spatial indexing considering the spatial data only, while the content attribute data can be further structured aiming to more improvement in the matching process.

The second concerns the RFERS design and implementation. The system prototype is intended to provide a proof of concept for the interaction mechanism by means of geospatial-based publish/subscribe. Geospatial events are structured is limited, built-in number of topics (*i.e.*, the four RFERS topics mentioned early). Realizing a generalized data structure of geospatial events needs further investigation and potentially exploiting the use of the *advertise* operation (see Section 2.4) in the interaction model. Furthermore, deploying the implemented software of the system prototype requires

installing and configuring other commercial products (*i.e.*, ArcGIS v9.3 and TIBCO v4.4).

The third issue is related to the use of different spatial indexing techniques in improving the geospatial notifications matching. In the performance experiments (see Section 5.3), multi-grid index technique is used to structure geospatial subscriptions, thus, enhancing the matching process. Although the evaluation results have shown satisfactory performance, using other indexing techniques (*e.g.*, R-tree and Quad-tree) may result in better system performance as those techniques can maintain the addition and deletion of geospatial subscriptions sufficiently (*i.e.*, dynamic indexing structure). Comparing different spatial indexing techniques is not addressed as it is beyond the scope of this research.

6.3 Future Work

There is a wide range of potential avenues in which this research can be improved and extended. In this section, some of the major research avenues that can be further investigated and guide the future work are highlighted.

The centralized architecture of the middleware is assumed in the design of geospatial-based publish/subscribe. It is highly recommended to pursue further research to realize this interaction in distributed middleware architecture, thus, increasing the scalability of the framework in adopting large numbers of globally distributed clients. Furthermore, the service provided by the middleware can be extended to accommodate composite events detection by which subscribers can specify their interests not only in unity of geospatial events, but also in the advanced correlation, spatial or temporal, that

may happen between geospatial events together. Addressing the composite event detection service would benefit many applications where monitoring complex patterns of geospatial events is essential. For instance, a subscriber is interested to be notified if more than five emergency vehicles are located in the same region. Another subscriber is interested in the temperature observations published by wireless sensors if the readings keep decreasing for over 6 hours.

Another research avenue can be targeted towards extending the RFERS prototype for developing an enterprise service and integrating clients' applications by means of transacting geospatial events in the context of fire emergencies. The service can be offered to potential clients via GIS web application or by interoperable web service supporting a large variety of clients' applications.

More potential future work pertains realizing the geospatial-based publish/subscribe interaction in supporting emergency management systems in crisis situations, including: earthquakes, floods, tsunamis, tornados and other natural hazards. These situations require dynamic and real-time coordination of a large number of disparate groups and management of information gathered from variety of sensor networks and mobile objects. Geospatial-based publish/subscribe middleware can support delivering the right information to the right place at the right time.

6.4 Summary

This chapter concluded this research by summarizing the work, highlighting the major key findings, and commenting on the main limitations. The chapter also discussed recommendations for the future work.

References

- Aguilera, M. K., Strom, R. E., Sturman, D. C., Astley, M. and Chandra, T. D. (1999) *Matching Events in a Content-based Subscription System*, translated by Atlanta, Georgia, United States: ACM New York, NY, USA, 53 - 61.
- Altinel, M. and Franklin, M. (2000) *Efficient Filtering of XML Documents for Selective Dissemination of Information*, translated by Cairo, Egypt.
- Ashayer, G., Leung, H. K. Y. and Jacobsen, H.-A. (2002) *Predicate Matching and Subscription Matching in Publish/Subscribe Systems*, translated by Vienna, Austria: IEEE Computer Society Washington, DC, USA, 539 - 548.
- Banavar, G., Chandra, T., Mukherjee, B., Nagarajarao, J., Strom, R. E. and Sturman, D. C. (1999) *An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems*, translated by Austin, TX, USA: IEEE Computer Society Washington, DC, USA, 262 - 272.
- Banavar, G., Kaplan, M., Shaw, K., Strom, R. E., Sturman, D. C. and Wei, T. (1999) *Information Flow Based Event Distribution Middleware*, translated by Austin, TX, USA: IEEE Computer Society Washington, DC, USA, 114 - 121.
- Bauer, M. and Rothermel, K. (2002) *Towards the Observation of Spatial Events in Distributed Location-Aware Systems*, translated by Vienna, Austria: IEEE Computer Society Washington, DC, USA, 581 - 582.
- Belokosztolszki, A., Eyers, D. M., Pietzuch, P. R., Bacon, J. and Moody, K. (2003) *Role-based access control for publish/subscribe middleware architectures*, translated by San Diego, California: ACM New York, NY, USA, 1 - 8.
- Berg, M. d., Cheong, O., Kreveld, M. v. and Overmars, M. (2008) *Computational Geometry: Algorithms and Applications*, Springer.
- Burcea, I. and Jacobsen, H.-A. (2003) *L-ToPSS - Push-oriented Location-based Services*, translated by Berlin, Germany: Springer-Verlag, London, UK, 131 - 142.
- Campailla, A., Chaki, S., Clarke, E., Jha, S. and Veith, H. (2001) *Efficient Filtering in Publish-Subscribe Systems using Binary Decision Diagrams*, translated by Toronto, Ontario, Canada: IEEE Computer Society Washington, DC, USA, 443 - 452.
- Cao, F. (2006) *Architecture Design for Distributed Content-Based Publish-Subscribe Systems*, unpublished thesis Princeton University.

- Carzaniga, A. (1998) *Architectures for an Event Notification Service Scalable to Wide-area Networks*, unpublished thesis University of Colorado.
- Carzaniga, A., Rosenblum, D. and wolf, A. (2001) 'Design and Evaluation of a Wide-Area Event Notification Service', *ACM Transactions on Computer Systems*, 19(3), 332–383.
- Carzaniga, A., Rosenblum, D. S., Wolf, A. L. and Wolf, E. L. (1999) *Challenges for Distributed Event Services: Scalability vs. Expressiveness*, translated by Los Angeles, CA, USA.
- Carzaniga, A. and Wolf, A. L. (2003) *Forwarding in a content-based network*, translated by Karlsruhe, Germany: ACM New York, NY, USA, 163 - 174.
- Chen, X., Chen, Y. and Rao, F. (2003) *An Efficient Spatial Publish/Subscribe System for Intelligent Location-Based Services*, translated by San Diego, California: ACM New York, NY, USA, 1 - 6.
- Chen, Y., Rao, F., Yu, X. and Liu, D. (2003) *CAMEL: A Moving Object Database Approach for Intelligent Location Aware Services*, translated by Melbourne, Australia: Springer-Verlag, London, UK, 331 - 334.
- Costa, P., Coulson, G., Gold, R., Lad, M., Mascolo, C., Mottola, L., Picco, G. P., Sivaharan, T., Weerasinghe, N. and Zachariadis, S. (2007) *The RUNES Middleware for Networked Embedded Systems and its Application in a Disaster Management Scenario*, translated by White Planes, New York, USA: IEEE Computer Society, 69 - 78.
- Cugola, G. and Jacobsen, H.-A. (2002) 'Using publish/subscribe middleware for mobile systems', *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(4), 25 - 33.
- Cugola, G., Nitto, E. D. and Fuggetta, A. (2001) 'The JEDI Event-Based Infrastructure and Its Application to the Development of the OPSS WFMS', *IEEE Transactions on Software Engineering*, 27(9), 827 - 850.
- ESRI (2004) *ESRI GIS and Mapping Software*, Redlands, CA, USA.
- ESRI (2008) 'ArcGIS Desktop Help 9.3 - Tuning spatial indexes', [online], available: [accessed
- Eugster, H. and Nebiker, S. (2008) *UAV-Based Augmented Monitoring – Real-Time Georeferencing and Integration of Video Imagery with Virtual Globes*, translated by Beijing, China: 1229 - 1235.

- Eugster, P. T., Felber, P. A., Guerraoui, R. and Kermarrec, A.-M. (2003) 'The Many Faces of Publish/Subscribe', *ACM Computing Surveys (CSUR)*, 35(2), 114 - 131.
- Eugster, P. T., Guerraoui, R. and Damm, C. H. (2001) *On Objects and Events*, translated by Tampa Bay, FL, USA: ACM New York, USA, 254 - 269.
- Fabret, F., Jacobsen, H.-A., Lirbat, F., Pereira, J., Ross, K. and Shasha, D. (2001) 'Filtering Algorithms and Implementation for Very Fast Publish/Subscribe Systems', in *ACM SIGMOD*, Santa Barbara, California, USA,
- Floyd, S., Jacobson, V., Liu, C.-G., McCanne, S. and Zhang, L. (1997) 'A reliable multicast framework for light-weight sessions and application level framing', *IEEE/ACM Transactions on Networking (TON)*, 5(6), 784 - 803.
- Franklin, M. J. and Zdonik, S. B. (1998) "*Data In Your Face*": *Push Technology in Perspective*, translated by Seattle, Washington, USA: ACM New York, USA, 516-519.
- Graham, S., Niblett, P., Chappell, D., Lewis, A., Nagaratnam, N., Parikh, J., Patil, S., Samdarshi, S., Sedukhin, I., Snelling, D., Tuecke, S., Vambenepe, W. and Weihl, B. (2004) *Publish-Subscribe Notification for Web Services*.
- Harrison, M. (1995) *The USENET handbook: a user's guide to Netnews*, O'Reilly & Associates, Inc. Sebastopol, CA, USA.
- Hayton, R., Bacon, J., Bates, J. and Moody, K. (1996) *Using Events to Build Large Scale Distributed Applications*, translated by Connemara, Ireland: ACM New York, NY, USA, 9 - 16.
- Holloway, S. (2008) 'Are you ready for M3O - Vitria's new BPM offering?', available: <http://www.vitria.com> [accessed
- Jaeger, M. A. (2005) *Self-organizing publish/subscribe*, translated by Grenoble, France: ACM New York, NY, USA, 1 - 5.
- Jaeger, M. A. and Muhl, G. (2005) 'Stochastic Analysis and Comparison of Self-Stabilizing Routing Algorithms for Publish/Subscribe Systems', in *13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'05)*, Atlanta, Georgia, 471-479.
- Jaeger, M. A., Parzyjegla, H., Mühl, G. and Herrmann, K. (2007) *Self-organizing broker topologies for publish/subscribe systems*, translated by Seoul, Korea: ACM New York, NY, USA, 543 - 550.

- Kalashnikov, D., Prabhakar, S., Hambrusch, S. and Aref, W. (2002) *Efficient Evaluation of Continuous Range Queries on Moving Objects*, translated by Springer-Verlag, 731 - 740.
- Kale, S., Hazan, E., Cao, F. and Singh, J. P. (2005) *Analysis and Algorithms for Content-Based Event Matching*, translated by Columbus, Ohio, USA: IEEE Computer Society Washington, DC, USA, 363 - 369.
- Keramitsoglou, I., Kiranoudis, C. T., Sarimveis, H. and Sifakis, N. (2004) 'A Multidisciplinary Decision Support System for Forest Fire Crisis Management', *Environmental management*, 33(2), 212-225.
- Maguire, D. J. (2005) *GIS, Spatial Analysis, and Modeling*, 1st ed. ed., ESRI Press.
- Manolopoulos, Y., Nanopoulos, A., Papadopoulos, A. N. and Theodoridis, Y. (2005) *Rtrees: Theory and Applications, Advanced Information and Knowledge Processing*, Springer.
- Mansouri-Samani, M. and Sloman, M. (1997) 'GEM: a generalized event monitoring language for distributed systems', *IEE/IOP/BCS Distributed Systems Engineering Journal*, 4, 96-108.
- Muhl, G. (2001) *Generic Constraints for Content-Based Publish/Subscribe*, translated by Trento, Italy: Springer-Verlag, London, UK, 211 - 225.
- Muhl, G. (2002) *Large-Scale Content-Based Publish/Subscribe Systems*, unpublished thesis Technische Universität Darmstadt.
- Muhl, G. and Fiege, L. (2001) 'Supporting covering and merging in content-based publish/subscribe systems: Beyond name/value pairs', *IEEE Distributed Systems Online (DSOnline)*, 2(7).
- Muhl, G., Fiege, L. and Pietzuch, P. (2006) *Distributed Event-Based Systems*, Germany: Springer.
- NASA (2007) 'Wildfire Imaging Flights By NASA's Ikhana UAV Conclude', [online], available: http://www.nasa.gov/centers/dryden/news/Features/2007/wildfire_social_10_07.html [accessed
- Oki, B., Pfluegl, M., Siegel, A. and Skeen, D. (1994) *The Information Bus - An Architecture for Extensible Distributed Systems*, translated by Asheville, North Carolina, United States: ACM New York, USA, 58 - 68.

- Pallickara, S., Bulut, H. and Fox, G. (2007) 'Fault-Tolerant Reliable Delivery of Messages in Distributed Publish/Subscribe Systems', in *Fourth International Conference on Autonomic Computing (ICAC'07)*, Jacksonville, Florida, USA, IEEE Computer Society, 19 - 19.
- Pallickara, S. and Fox, G. (2003) *NaradaBrokering: A Distributed Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids*, translated by Rio Janeiro, Brazil.
- Pietzuch, P. R. (2004) *Hermes: A Scalable Event-Based Middleware*, unpublished thesis University of Cambridge.
- Rigaux, P., Scholl, M. and Voisard, A. (2002) *Spatial Databases: With Application to GIS*, 2nd ed., Morgan Kaufmann Publishers Inc.
- Riggan, P. J., Tissell, R. G. and Hoffman, J. W. (2003) *Application of the Firemapper Thermal-imaging Radiometer for Wildfire Suppression*, translated by 1863 - 1872.
- Romer, K. and Mattern, F. (2004) 'Event-Based Systems for Detecting Real-World States with Sensor Networks: A Critical Analysis', in *International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP'04)*, Melbourne, Australia, IEEE Computer Society, 389 - 395.
- Samet, H. (2006) *Foundations of Multidimensional and Metric Data Structures, Computer Graphics and Geometric Modeling*, San Francisco, CA, USA: Elsevier/Morgan Kaufmann Publishers Inc.
- Schneider, M. and Behr, T. (2006) 'Topological Relationships Between Complex Spatial Objects', *ACM Transactions on Database Systems (TODS)*, 31(1), 39 - 81.
- Shekhar, S. and Chawla, S. (2003) *Spatial Databases: A Tour*, Prentice Hall.
- TIBCO (1999) *TIB/Rendezvous*.
- TIBCO (2000) 'TIBCO - Service Oriented Architecture (SOA) Software, Business Process Management (BPM) Software Leader', [online], available: <http://www.tibco.com> [accessed
- TIBCO (2008) *TIBCO Enterprise Messaging Service*, Palo Alto, CA, USA.
- Turoff, M., Chumer, M., Van de Walle, B. and Yao, X. (2004) 'The Design of A Dynamic Emergency Response Management Information System (DERMIS)', *Information technology Theory and Application*, 5(4), 1-36.

- USDA (2009) 'Freeway Complex Fire, PSW Research Station - USDA Forest Service', [online], available: [accessed]
- Wang, C., Carzaniga, A., Evans, D. and Wolf, A. L. (2002) *Security Issues and Requirements for Internet-Scale Publish-Subscribe Systems*, translated by Island of Hawaii: IEEE Computer Society Washington, DC, USA, 303.
- Wilson, J., Bhargava, V., Redfern, A. and Wright, P. (2007) *A Wireless Sensor Network and Incident Command Interface for Urban Firefighting*, translated by Philadelphia, PA: IEEE Computer Society, 1 - 7.
- Worboys, M. F. and Hornsby, K. (2004) 'From Objects to Events: GEM, the Geospatial Event Model', in *GIScience*, Springer, 327-344.
- WRAP (2004) 'The Wildfire Research and Applications Partnership', [online], available: <http://geo.arc.nasa.gov/sge/WRAP/> [accessed]
- Wu, K.-L., Chen, S.-K. and Yu, P. S. (2004) *Indexing Continual Range Queries for Location-Aware Mobile Services*, translated by IEEE Computer Society, 233 - 240.
- Wybo, J.-L. (1998) 'FMIS: A Decision Support System for Forest Fire Prevention and Fighting', *IEEE Transactions on Engineering Management*, 45(2), 127-131.
- Yan, T. W. and Garcia-Molina, H. (1994) 'Index Structures for Selective Dissemination of Information Under the Boolean Model', *ACM Transactions on Database Systems (TODS)*, 19(2), 332 - 364.
- YanJun, L., Zhi, W. and Yeqiong, S. (2006) *Wireless Sensor Network Design for Wildfire Monitoring*, translated by Dalian, China: IEEE Computer Society, 109 - 113.
- Zerger, A. and Smith, D. I. (2003) 'Impediments to using GIS for real-time disaster decision support', *Computers, Environment and Urban Systems*, 27(2), 123 - 141.

APPENDIX A: TOPOLOGICAL RELATIONSHIPS BETWEEN SIMPLE GEOMETRIES

The following figure shows all the spatial relationship combinations between simple geometries (*i.e.*, point, line and polygon) using *Contain*, *Within*, *Touch*, *Overlap*, *Cross*, and *Disjoint* spatial operators. Only “true” relationships are shown in this figure. The spatial relationships are taken as defined in ESRI ArcObject v9.3¹

Contain				Within			
Base Geo. \ Comp. Geo.	Point	Line	Polygon	Base Geo. \ Comp. Geo.	Point	Line	Polygon
Point				Point			
Line				Line			
Polygon				Polygon			
Point				Point			
Line				Line			
Polygon				Polygon			

Touch				Overlap			
Base Geo. \ Comp. Geo.	Point	Line	Polygon	Base Geo. \ Comp. Geo.	Point	Line	Polygon
Point				Point			
Line				Line			
Polygon				Polygon			
Point				Point			
Line				Line			
Polygon				Polygon			

¹ ESRI Developer Network (EDN), 2009, “IRelationalOperator Interface”, Accessed February, 2009.

		Cross			Disjoint				
Comp. Geo.	Base Geo.				Comp. Geo.	Base Geo.			
	Base Geo.								
	N/A	N/A	N/A	N/A		   	 	   	
	N/A	 	 	N/A		 	 	  	
	N/A	  	N/A	N/A		  	  	  	