EMERGENCY COMMUNICATIONS VIA HANDHELD DEVICES

by

Paul Phat Ngo
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Computer Science

Committee:

____________________	Dr. Duminda Wijesekera, Dissertation Director
____________________	Dr. Daniel Menasce, Committee Member
____________________	Dr. Rao V. Mulpuri, Committee Member
____________________	Dr. Paulo C. G. Costa, Committee Member
____________________	Dr. Sanjeev Setia, Department Chair
____________________	Dr. Kenneth S. Ball, Dean, The Volgenau School of Engineering
Date: ________________	Spring Semester 2013 George Mason University Fairfax, VA

Emergency Communications via Handheld Devices

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Paul Phat Ngo
Master of Computer Science
George Mason University, 2008
Bachelor of Computer Science
Virginia Tech, 1997

Director: Dr. Duminda Wijesekera, Professor
Department of Computer Science

Spring Semester 2013
George Mason University
Fairfax, VA

Dedication

I would like to dedicate this dissertation to my God, my loving father Ngoan Viet Ngo (deceased), my beloved mother Maria Tran and my brother Khiem Ngo, and especially to my Vietnamese Eucharistic Youth Society and the Youth group at the Holy Martyrs of Vietnam Church. You have been inspirational for me to complete my work.

Acknowledgments

I would like to thank many of friends, relatives, and supporters who have made this happen. My former boss Mr. Dale Barr and my co-worker Lew Morrison have helped me edit and comment on my works. Especially, I would like to thank my dissertation director Prof. Duminda Wijesekera for his guidance and support throughout the years. He is a truly amazing director that I could ever ask for. I would also like to thank my God son, Michael Luu, who has been one of my strange motivation to finish this work. I want to prove to him that even though I have a full-time job, a full-time dedication to the church community, and I still have the time to work on my doctorate. So there is really no excuse for him not to get a better education.

Table of Contents

List of Tables

List of Figures

Abstract

EMERGENCY COMMUNICATIONS VIA HANDHELD DEVICES

Paul Phat Ngo, PhD

George Mason University, 2013

Dissertation Director: Dr. Duminda Wijesekera

Ensuring effective communications during emergencies is an important issue for any functional government. One way to address this issue is to ensure the availability of the emergency responders capable of making the appropriate decisions and taking timely actions with sufficient resources. Many XML-based languages such as the Emergency Data Exchange Language (EDXL) and associated Common Alert Protocol (CAP) have been designed to provide a basis for such communications. To ensure that messages are delivered in a timely manner, I propose some role- and task-based ontological enhancements for these languages. I address this availability problem further by proposing a Role-based model Availability Emergency Responder Framework (AERF). This AERF ensures that a list of personnel for a particular role in an organization is always reachable to handle an emergency call. I develop a working prototype of the AERF framework for a local hospital that provides emergency cases. The prototype demonstrates the feasibility and security of the AERF framework and addresses the availability of emergency responders based on their assigned roles. In order to inform the general public of nearby emergencies, the Department of Homeland Security initiated the Commercial Mobile Alert System (CMAS), which utilized existing commercial telecommunication infrastructures to broadcast emergency alert text messages to all mobile users in an area affected by an emergency. One of the limitations of the Cell Broadcast Service (CBS) is that the smallest area that CMAS can broadcast its message is a cell site, which is, in most cases, quite large for small-scale emergencies. I propose an enhancement to CMAS by using CMAS as a transport protocol to distribute small-scale emergency alerts to areas that are smaller than a cell site. I also suggest a proper enhancement to the CAP 1.2 message structure for CMAS emergency alerts. Another limitation of CMAS messages is the maximum message size of 90 characters of clear text. I propose an enhancement to CMAS by using a combination of different encoding techniques and emergency protocol standard including the Common Alerting Protocol (CAP 1.2) to provide alert messages with meaningful and rich content. I show the viability of our solution using a prototype implementation that can generate and broadcast CMAS emergency alerts through Emergency Response Alert System (ERAlert) to Android phones where an Emergency Response Application (ERApp) will intercept, decode, and display meaningful alerts to users. Lastly, I propose a Navigation Assistance Framework (NAF) that allows emergency organizations to provide emergency information that can be filtered through the traffic patterns in order to assist victims navigate out of the emergency and reach their intended destinations in a reasonable amount of time. I develop a ERSimMon to simulate this capability in a small scale to show the effectiveness of my solution.

Chapter 1: Introduction

1.1 Background

Multiple mega-scale emergencies highlight the need for better global emergency response. The September 11th, 2001 terrorist attacks in New York, Indonesian Tsunami in 2004, Hurricane Katrina in 2005, Sichaun earthquake in 2008, and the Haiti earthquake and Pakistani floods in 2010 are examples of a few. During the course of the September 11 terrorist attacks, communications were identified as a major bottleneck for all emergency activities. The telecommunications infrastructure experienced an extremely high overload of calls in and out of the damaged areas, which caused congestion at the access and the core networks. As a result, many calls were blocked and rejected. However, mobile users were still able to text message each other [1]. Because text messaging was expensive at the time, many people didn't use it. Consequently, the only effective means of communication was non-electronic.

The US Federal Government has established a Government Emergency Telecommunications Service (GETS) program [7], which ensures a high probability of call establishment during a crisis when the PSTN is congested. This program provides a specific and recognizable phone number to obtain a higher priority for establishing a call. In recent years, with the increased prevalence of wireless phones, the Federal Government established a Wireless Priority Service (WPS) [8] program, where subscription information is used to identify high-priority callers. However, both GETS and WPS services do not guarantee call establishment but rather provide best effort due to the network bandwidth availability.

In 2003, the Organization for the Advancement of Structured Information Standards (OASIS) sponsored the Common Alerting Protocol (CAP) initiative with the objective of providing fundamental messaging protocols to facilitate inter-agency emergency communications. The OASIS technical committee on emergency management has developed a couple of standards [2, 5] on the emergency protocol, which consist of a set of XML tags to exchange the information needed to handle an emergency. With support from the Department of Homeland Security Disaster Management eGov Initiative, the OASIS technical committee on emergency management developed a set of standards for the inter-agency exchange of emergency management data and messaging. Standards [2] and [3] developed the Emergency Data Exchange Language (EDXL) provides a set of XML based tags to exchange the information needed to handle an emergency.

In 2006, the Federal Government established a Worker Adjustment and Retraining Notification (WARN) Act that supported the research and development of the Common Mobile Alert System (CMAS). The proposed CMAS system utilizes existing commercial telecommunication infrastructures to broadcast emergency alerts and warnings to a specified geographic area.

1.2 Thesis Statements

In this dissertation, I make the following thesis statements:

It is possible to:

1. Enhance the usability of CMAS to provide local emergency alerts to a specific area that is smaller than a broadcast cell site,

2. Encode CMAS emergency alerts including local emergencies without violating CMAS length limitation,

3. Enhance the government emergency standards to ensure the availability of emergency responders by providing a language and protocol to automatically reach all relevant responders,

4. And provide navigation assistance and emergency advice to mobile users during an

ongoing emergency.

1.3 Communication Problems and Solutions

This section discusses several existing problems that prevent coherent and smooth emergency communications between different parties. I describe the problems I have identified and proposed enhancements to better facilitate seamless communications.

1.3.1 Existing Standards

In any emergency, urgent task-related communications must reach key officials in a timely manner. Emergency responders must know how to contact the person in charge of a specific task, which is sometimes difficult due to not being able to locate a telephone number, or when reached using directory information, the person may not be available or may have been reassigned to a different job/task. There is no automated method of redirecting the call to the current person who should be attending to that task and is on duty at the time of the call. It's preferable to have a subject, task specific, 911-like calling number for each task, time and locality. The objective is to reach such a capability for the real-time needs of emergency responders. I argue that the first step towards achieving this objective is to develop an ontology/lexicon to share the critical information that fills in the existing gap. I then show how that ontological information can be used in current communication infrastructures designed to facilitate emergency communications.

The basic 911 services provided in the USA serve as a pseudo name that is available to the general public at every time and every location, but is mapped to a collection of numbers belonging to an emergency call center based on the call originator's location. Although we take it for granted, the public switched telephone network (PSTN) has been designed to translate the pseudo name 911 to a location-specific telephone number. Thus this address translation depends only on a single parameter, the caller's location. Our objective is to extend this capability in order to facilitate the communication beyond the first call from the public. The issue of extending this paradigm for emergency responders to contact each other depends on a plethora of parameters, nature of the emergency, priority of immediate needs and resources to fulfill them. We agree that if a person is not available to receive the request, the communication breaks down. But often, locating this person takes multiple calls/SMS and email messages before the responder can be reached. It is this gap that we propose to fill by developing an ontology (hence the lexicons) as the need to parameterize the basic 911 service.

The EDXL is the language of choice to facilitate the emergency communications between organizations and agencies. However, to route, receive and respond to these messages, the responder anticipating an emergency duty-related request must be identifiable by other collaborators that will need his services. Consequently, our development enhances the EDXL entities to ensure that the calling party is able to reach the best called party based on the latter's availability. To do so, we propose that all potential responders expose their capabilities and fallback options in case they cannot be reached during emergencies. These capabilities of responders include the followings: 1) the role played in an organization, 2) the tasks the actor can execute, 3) the estimated time to respond to a request (perhaps due to many emergency calls) or execute these tasks, 4) available resources that can be used in the execution of these tasks, 5) direct contact information, and an alternative contact chain in case of unavailability of the best contact, 6) the sensitivity of the information authorized to receive, and 7) a contact to report complaints about the quality of service, including contacting difficulties.

1.3.2 Commercial Mobile Alert System - CMAS

The proposed CMAS system utilizes existing commercial telecommunication infrastructures to broadcast emergency alerts and warnings to a specified geographic area. However, the coverage area that CMAS broadcasts its alerts is a *cell*, quite large for small-scale emergencies (e.g., major car accidents on main highways and intersections or a burning building), where the affect is confined to that area. While CMAS is still in the design and development phases, we have identified two shortfalls in the CMAS Cell Broadcast Service specification

[9]: (1) sending an alert to a confined area for small-scale emergencies is not possible, (2) the limitation to only three types of alerts: *Presidential Alerts*, *Imminent Threat Alerts*, and *AMBER Alerts*, (3) the current protocol standard does not provide enough information to facilitate this communication effectively, and (4) the size limitation of CMAS alert message is to 90 characters.

We address the first shortfall by building the Android mobile application to filter the CMAS alerts based on Global Positioning System (GPS) location and the specified affected area and display the alerts only if a user's location is inside the affected area. We also built a CMAS prototype to illustrate the working solution to address the shortfall. We address the second shortfall by adding the capability of sending local emergency alerts to small-scale emergencies. We address the third shortfall by examining the Common Alerting Protocol (CAP) version 1.2 [6] and suggest an enhancement to the CAP 1.2 message structure by adding necessary XML tags to facilitate this communication effectively. We address the fourth shortfall by creating the encoding scheme specifically for the purpose of sending more emergency information in a 90-character message and providing the flexibility to broadcast multiple of 90-character messages for one emergency alert.

1.3.3 The Availability of Emergency Responders

In order to respond to and triage emergency needs that prevent loss of life and property, call handlers and first responders need to have the authority and contact information to call for specialized tasks. But due to schedule changes, unanticipated personal needs, or environmental differences, specialized personnel may be unreachable at the time of a call. Therefore, establishing a mechanism to reach specialized emergency responders is the primary objective of this paper.

We identified common operations in which police, fire fighters, and medical care-givers submit work schedules and contact information for emergency responders on a periodic basis. These operations have two primary issues: (1) frequent and manual updates if there are any changes to the staff, and (2) not having an automated way to reach a backup handler in case the primary handler is unreachable. As a solution, we propose the Availability Emergency Response Framework (AERF). We address the first issue by providing emergency handlers with an Emergency Responder Application (ERApp) that can be installed on the person's smart phone. ERApp allows her to send a time-out message (unavailability to accept emergency calls) to the AERF framework. In response, the framework will find a backup person, who has similar expertise as the requesting personnel and coordinate the replacement. We address the second issue by equipping organizations with an Organizational Monitor Control (OMC) system that monitors primary handlers' availability and autonomously coordinates replacements if one has to be absent from work due to unexpected circumstances. The OMC system also allows the organization-wide schedulers to provide the list of backups (up to three personnel for the primary handler) in case he/she is unreachable.

The AERF framework can be implemented at the organizational level to ensure the availability of the personnel based on roles assigned within the organization. Each role comes with its own responsibilities, authorities, and permissions. Most emergency organizations, such as fire stations, law enforcement agencies, hospitals, etc., already have a well-defined, established list of organizational roles with identified responsibilities, authorities for the role players, and permissions that can be easily integrated with the AERF framework to maximize the availability of their personnel in handling emergency calls.

We use the eXtensible Access Control Markup Language (XACML) [10] for specifying organizational policies and describing who has access to what resources under what conditions. The AERF framework will access and evaluate personnel's current work schedules and their availability status against the organizational policies to ensure that they are met before a replacement takes place.

1.3.4 The Emergency Navigation Assistance Framework

According to the Out-of-State and Long Commutes 2011 Survey [60], 8.1 percent of U.S. workers had commutes of 60 minutes or longer. In addition, 61.1 percents of the workers drove to work alone. American people spend significant amounts of time, on the average of 25 minutes [61], in their vehicles on the road to go from home to work on an ordinary day.

On top of this average commute time, local emergencies such as car accidents, road construction, inclement weather, etc. may add extra delay into the average commute time. Commuters have to adjust to these unexpected delays on a case-by-case basis. Consequently, they may have to shift their schedule or rearrange appointments and meetings to accommodate for the time lost sitting in traffic. Sometimes, cancellations and frustrations are unavoidable. According to a poll conducted by ABCNews on traffic in the United States [62], the average commute time on a bad day for Americans is 46 minutes.

Clearly, dealing with unexpected delays is a major concern for commuters. We address this concern with two approaches. The first approach is to provide commuters with navigational assistance that offers alternative routes to their destinations in order to avoid an ongoing emergency and its affected area. This may be a great help to commuters who are not familiar with an area or who waste time sitting in traffic. The second approach is to provide commuters with relevant emergency advice based on the type of the emergency.

We convey these local emergencies by sending the GPS location of the emergency and the affected area measured by the radius from the emergency GPS location to mobile users' devices. We also enhanced the original CMAS limitation on the message size of 90 readable characters described in section 3.2. Both of these CMAS enhancements allow local emergency information to be broadcast to mobile users more effectively.

To provide relevant navigation assistance to mobile users in a variety of emergencies, from the most dynamic, like a tornado or hurricane, to the least changing such as construction road blocks, we need the most up-to-date information regarding the emergency. We propose a Navigation Assistance Framework (NAF) to set a foundation for this continual work.

Chapter 2: The Existing Standards

2.1 EDXL

EDXL is a language designed for sharing information and exchanging data among local, state, tribal, national and non-governmental organizations to facilitate emergency response [2]. Figure 2.1, taken from Page 10 of [2], shows the entities used in creating the EDXL syntax in the form of an Entity Relationship (ER) model, where the entity in red is our enhancement that will be described the later subsection.

As Figure 2.1 shows, at the highest level, each EDXL distribution element (i.e. message) has six required attributes and six optional attributes. In addition, every message has a target area identifying a geographical region and a content object describing the incident, confidentiality levels and roles for the originator and consumer of the message.

Required attributes of the distribution element consist of a distribution ID, sender ID, date and time the message was sent, distribution status (consists of one of the four values: Actual, Exercise, System and Test), a distribution type consisting of value such as Report, Update, Request, Sensor Status, etc., and Combined Confidentiality having the most restrictive level of confidentiality sought for the combined payload.

The optional attributes consist of the language used in the message and (possibly multiple instances of) the sender's role, recipient role, keywords, distribution references (indicating distribution constraints) and possibly an explicit address for delivery. The explicit address is an XML schema.

EDXL messages can have four kinds of optional roles. They are sender's role, recipient's role, originator's role and consumer's role. These roles are supposed to be used for two purposes: (1) identifying potential recipients and (2) message distribution. In addition, explicit addresses can also be used for the latter task. The recommended usage syntax for

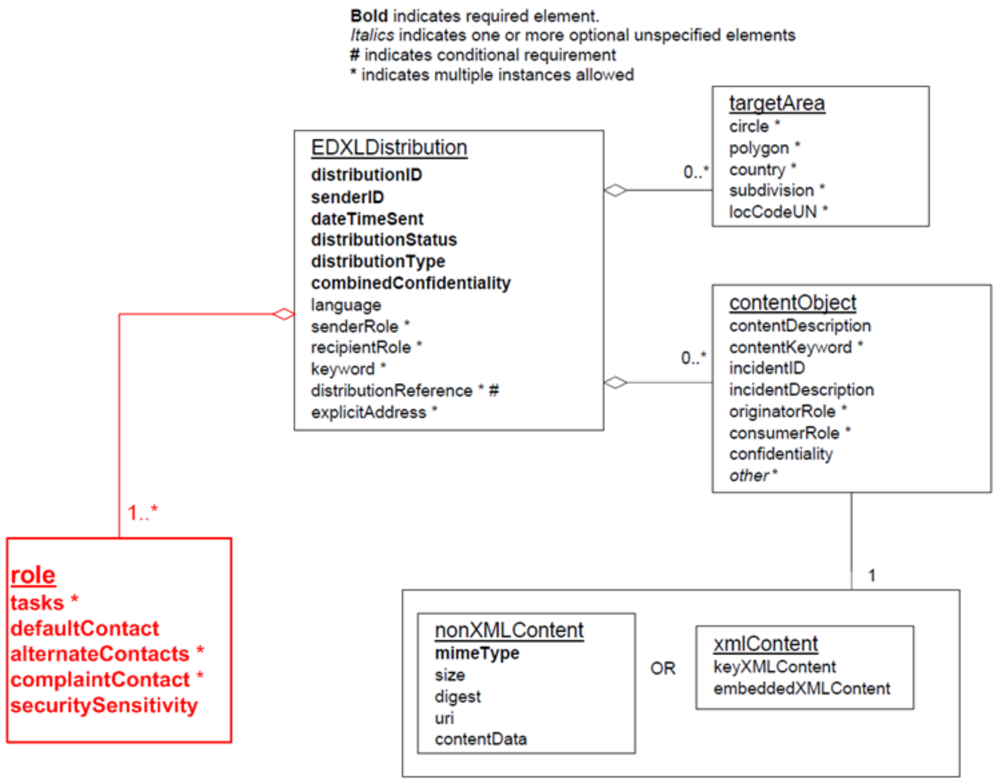


Figure 2.1: EDXL-DE Entity Relationship Diagram

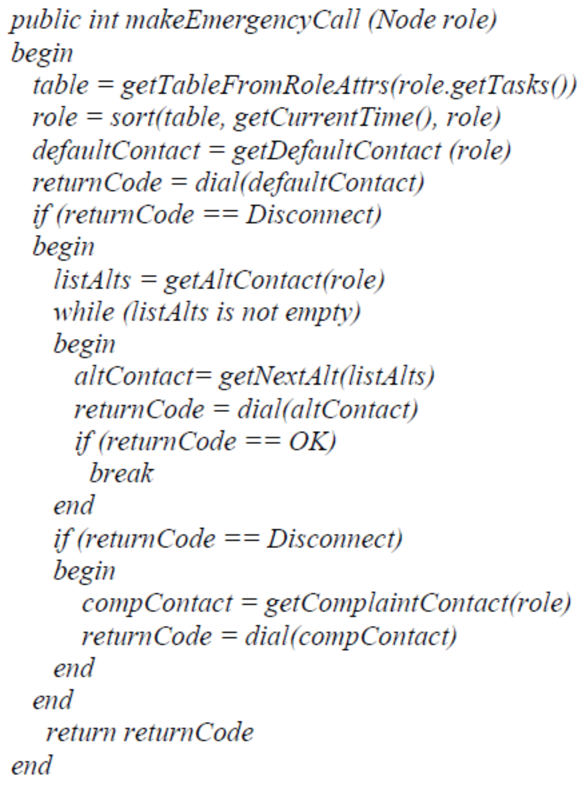
the sender ID is actor@domain-name (such as dispatcher@example.gov) where the domain name is guaranteed using the Internet Domain Name System.

2.2 Enhancing EDXL For Responder Availability

Before we explain our enhancements, several comments are worth mentioning. First, EDXL and CAP messages were designed for multiple purposes such as human-to-human, machine-to-human and machine-to-machine communications, etc., as shown by the fact that distribution element consists of optional fields such as *Sensor Status*, etc. For sensors, attributes like *roles* do not apply, but they do for human responders to emergencies. For example, we want to identify the Paramedic in an emergency response team (the role, but not the person) and his capabilities (such as is he authorized or trained to execute a certain type of medical routine like cardiac resuscitation, etc.). Thus, for human responders, the role is more central than the recipient address and the tasks that he is able to execute in that role. Therefore, in our enhancement the role is a mandatory attribute (marked by 1-* in Figure 2.1).

Because our objective is to enhance reaching the human responders with most suitable capabilities, we need to consider failure modes. One of the most important issues of recipient-address based emergency messages is that if that recipient is unreachable then it becomes the sender's responsibility to find the next available responder. Also any delivery system, such as an automated phone dispatcher, pager, SMS or email system should have an inbuilt mechanism to redirect the message automatically to the next appropriate responder. In order to facilitate this capability, either using an automated redirecting algorithm or in a sender initiated system, we propose creating a lexicon/ontology that has a list of alternative roles (where the role to person/phone number/IP address will be automated). In order to address the failure of these alternatives, we specify a *complaint* role that should deliver the message to the higher authoritative personnel.

The redirecting algorithm can be easily implemented in the Private Branch Exchange (PBX) of the caller. [11] describes three common failure scenarios: Callee Busy, Callee Unanswered, and Global Errors. In all cases, when the call cannot be connected as dialed, the caller Sessions Initiation Protocol (SIP) gateway sends a *disconnect* message with the appropriate error code to the caller's PBX. Before this message is sent to the caller, we can inject the redirection mechanism by providing the PBX with a list of the default, the alternative and the complaint numbers, as will be shown in the algorithm depicted in Figure 2.2. For this to work properly, we made two assumptions. First, we assume that the local PBX has an Emergency Address book that is capable of translating the list of tasks to the local numbers based on their relevancy. Figure 2.3 illustrates an example with the <role> and <tasks> tags. Second, we assume that the order of relevancy can be selected by the local PBX. For example, in Louisiana, floods have more priority than earthquake. However, in California, the order must be reversed. This way, the selection algorithm can be regionalized. For now, we assume that our sorting algorithms address this based on

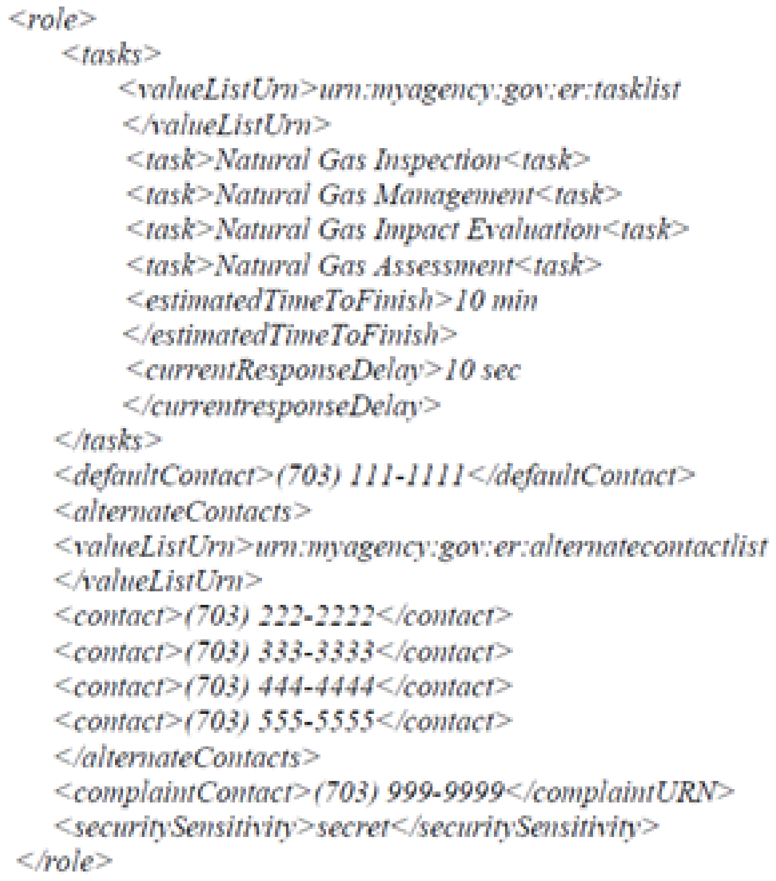


```
public int makeEmergencyCall (Node role)
begin
  table = getTableFromRoleAttrs(role.getTasks())
  role = sort(table, getCurrentTime(), role)
  defaultContact = getDefaultContact (role)
  returnCode = dial(defaultContact)
  if (returnCode == Disconnect)
  begin
    listAlts = getAltContact(role)
    while (listAlts is not empty)
    begin
      altContact= getNextAlt(listAlts)
      returnCode = dial(altContact)
      if (returnCode == OK)
        break
    end
    if (returnCode == Disconnect)
    begin
       compContact = getComplaintContact(role)
       returnCode = dial(compContact)
    end
  end
   return returnCode
end
```

Figure 2.2: EDXL-DE Entity Relationship Diagram

its locality although we are working on separating these concerns. The PBX first makes a call to the *defaultContact*. If the PBX receives the Disconnect message from the local SIP gateway, the PBX will redirect the call to numbers on the alternative list. If there are no more alternatives, the PBX will redirect the call to the complaint number. Figure 2.2 depicts the pseudo-code for the algorithm that can run as an application at the PBX and make repeated attempts to facilitate availability of responders.

Figure 2.2 illustrates a pseudo code redirection algorithm that is to be run at the local PBX. The *makeEmergencyCall* method accepts one parameter of the role node, which has been populated with the tasks that are relevant to the emergency. The *getTableFrom-RoleAttrs* method is then called to retrieve the table of contacts by searching the Emergency



```
<role>
    <tasks>
        <valueListUrn>urn:myagency:gov:er:tasklist
        </valueListUrn>
        <task>Natural Gas Inspection<task>
        <task>Natural Gas Management<task>
        <task>Natural Gas Impact Evaluation<task>
        <task>Natural Gas Assessment<task>
        <estimatedTimeToFinish>10 min
        </estimatedTimeToFinish>
        <currentResponseDelay>10 sec
        </currentresponseDelay>
    </tasks>
    <defaultContact>(703) 111-1111</defaultContact>
    <alternateContacts>
    <valueListUrn>urn:myagency:gov:er:alternatecontactlist
    </valueListUrn>
    <contact>(703) 222-2222</contact>
    <contact>(703) 333-3333</contact>
    <contact>(703) 444-4444</contact>
    <contact>(703) 555-5555</contact>
    </alternateContacts>
    <complaintContact>(703) 999-9999</complaintURN>
    <securitySensitivity>secret</securitySensitivity>
</role>
```

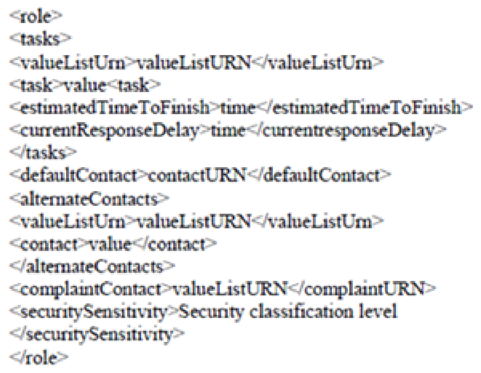
Figure 2.3: Gas Leak Example of the the Role EDXL Enhancement

Address book for the contacts that are associated with the tasks. The table is then sorted based on time to respond and the relevancy. The best matched entry in the table is then added to the role in three separate tags: defaultContact, alternateContact and complaintContact. The defaultContact is then called. If the disconnect is received from the local SIP gateway, each of alternateContacts is then called. If every call to the alternateContacts fails, the complaintContact is called.

2.2.1 Ontological Enhancements or Roles

In the current EXDL-DE specification, a *mandatory* recipient role is given as a list of structures where each element is a potential recipient. <recipientRole> <valueListUrn>valueListUrn</valueLis <value>value</value> </recipientRole>

Here the content of <valueListUrn> is the Uniform Resource Name of a published list of values and definitions, and the content of <value> is a string (which may represent a number) denoting the value itself. Multiple instances of the <value> may occur with a single <valueListUrn> within the <recipientRole> container. In addition, the <recipientRole> is *not* a required element. Our enhancements propose the following additions to a role as depicted in Figure 2.4.



```
<role>
<tasks>
<valueListUrn>valueListURN</valueListUrn>
<task>value<task>
<estimatedTimeToFinish>time</estimatedTimeToFinish>
<currentResponseDelay>time</currentresponseDelay>
</tasks>
<defaultContact>contactURN</defaultContact>
<alternateContacts>
<valueListUrn>valueListURN</valueListUrn>
<contact>value</contact>
</alternateContacts>
<complaintContact>valueListURN</complaintURN>
<securitySensitivity>Security classification level
</securitySensitivity>
</role>
```

Figure 2.4: EDXL XML Enhancement

The objective of this message is to allow each potential responder to expose his role, including a list of tasks he is willing to perform and his own estimated times in completing them. It also exposes the alternative contact information to be used in case the default contact does not respond within the published time. The <complaintContact> list addresses to be used for the purpose of complaining about the quality of service provided by the role players. The attribute <securitySensitivity> indicates the level of security that the role player is authorized to entertain in executing tasks specified for the role. Due to criticality of the <role> element, we propose to make it a required field and have at least one element in the EDXL-DE structure.

2.2.2 An Example Application

We illustrate the use of roles in an emergency scenario and the applicability of the role base to demonstrate the responder availability. A construction worker accidently drilled an 8-inch hole into the gas pipe, which created a gas leak for the entire neighborhood [13]. Construction workers in the area immediately smelled a gas odor and called 911. The operator then notified the gas company, the county police and the fire department. Within a few minutes, the police and fire trucks came and blocked all roads in and out of the neighborhood. They informed the entire neighborhood to evacuate immediately. In response to this gas leak, the gas company had to consult gas experts in the area and finally after about 15 minutes, the gas company was able to talk to an expert on the phone and he decided to shut down the main gas pipe, which left 50 houses in the entire neighborhood without gas. We now trace the calls from the moment the construction worker detected the gas odor to the moment that the gas official made the decision to shut down the gas line for the entire neighborhood. Figure 2.5 depicts the message diagram indicating the interactions between all the emergency responders.

This example shows two obligations imposed upon the responders. First, in order to be reached, gas companies must expose a role with assigned duties of attending to emergencies possibly with multiple contact points with some indication of anticipated waiting time to provide requested services. We take these as the requirements for the externally callable interfaces, and are used to populate the column II and III of table 2.1. Secondly, for the internal requirement, many individuals playing different internal roles may be mapped to the externally visible role for the purpose of serving the external role. Referred to as Role Switching [12] we address the second issue in a subsequent publication. This is similar to how hospitals provide their critical care physicians. For example, on one weekend, a cardiologist could be on call for the critical care unit as an attending physician and the next weekend a

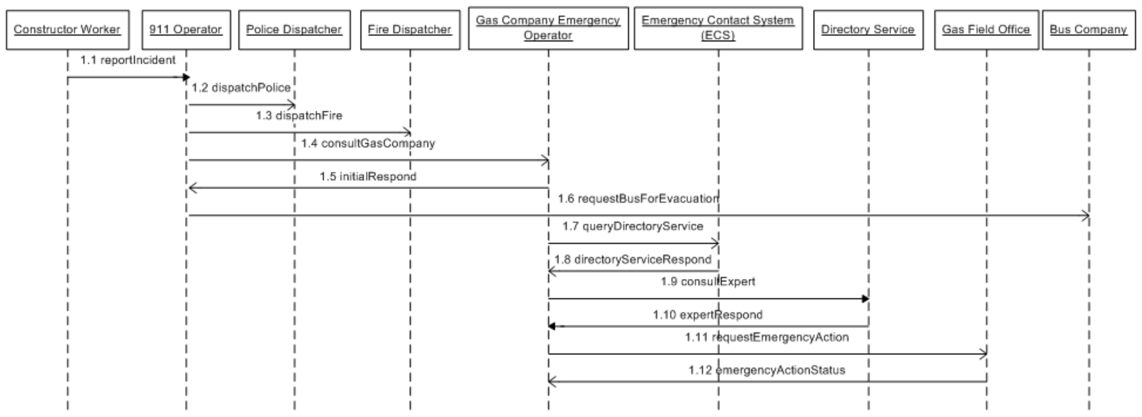


Figure 2.5: Calls Generated by Emergency Scenario

neurologist could be on call for the same facility. Thus, each of these physicians and their contact information are exposed as attributes of an added role in order to make a critical service function all the time. This assignment of extra roles now becomes an obligation on the part of emergency service providers in order to enhance overall availability.

Figure 2.6 depicts the message 1.1 *reportIncident* generated when the 911 operator captures the incident's details from the construction worker at the scene in order to notify the police, fire department and the gas company. The gas company emergency operator internally sends another request message to the gas company directory service to consult the natural gas expert for further instruction. Figure 2.7 depicts the response, 1.8 *directoryServiceRespond*, from the gas company directory service.

2.2.3 Enhancing Availability

We first examine the current CAP message structure to show that it is not possible to ensure the highest availability of the callee and then we show how this can be achieved with our additional modification to the CAP structure.

As depicted in Figure 2.1, the only attribute that is related closely to the recipient is

Table 2.1: Key Words Translation

Roles and Tasks	**Other Contacts**	**Contact Phones**
Role: Emergency Gas technician **Tasks**: (1) Licensed to shut down main valves, (2) (dis)connect household lines, (3) Repair valves **Zip codes**: 22222, 22221, 22223	**Email**: emergency@gasexpert.com **SMS**: (703)111-1111 Response Window: 24 hrs/day Estimated Response Delay: 20 seconds	**Phone**: (703)111-1111 **Alternatives**: (703)222-2222, (703)333-3333 (703)444-4444, (703)555-5555 **Complaint**: (703)999-9999
Role: Emergency Gas technician **Tasks**: (1) Licensed to shut down main valves, (2) (dis)connect household lines, (3) Repair valves **Zip codes**: 22222, 22221, 22204, 22223	**Email**: emergency@gassol.com **SMS**: (703)111-0001 Response Window: 7AM to 10PM EDT, weekdays 9AM 6PM EDT, weekends Estimated response Delay: 15 minutes	**Phone**: (703)111-0001 **Alternatives**: (703)111-0002 (703)111-0003 (703)111-0004 **Complaint**: (703)111-0005
Role: Emergency Gas technician **Tasks**: (1) Licensed to shut down main valves, (2) (dis)connect household lines, (3) Repair valves **Zip codes**: 22222, 22201, 22204, 222205	**Email**: emergency@gaspro.com **SMS**: (703)222-0001 Response window: 6AM 11PM EDT, weekdays 8AM 10PM EDT, weekends Estimated Response Delay: 10 minutes	**Phone**: (703)222-0001 **Alternatives**: (703)222-0002 (703)222-0003 (703)222-0004 **Complaint**: (703)222-0009

the *recipientRole* *, which indicates multiple instances are allowed. We argue that recipient roles are not enough and too general to query any Directory Service. Each person can play one or many roles in an organization and each role consists of a collection of tasks that can be executed by the role player. Instead of searching recipients by their roles, we suggest searching by specific tasks that a responder can do on a daily basis will give more accurate results. By doing so, the search result would provide a list of people who are authorized to execute the tasks that are required to address the emergency at hand.

With the current attributes supported by EDXL-DE, the 911 operator will search on the list of roles provided to him by the 911 system and start dialing people randomly. Hopefully, the operator can speak to someone who can address the current issue related to the emergency without any guarantee of the responders' availability. If the role is too general, the operator may retrieve hundreds of possibilities of responders from the Directory Service. Hopefully, the selected individuals are able to address the current emergency issues and are authorized to make proper decisions on behalf of the emergency relief and rescue efforts.

In addition, the current EDXL-DE structure provides no contact attributes to address the order of the responder availability and call automation. We suggest that the mapping between the role and its associated tasks is required to address the expertise areas of the callee, allowing the matching to be more precise. Similar roles do not equate to similar tasks that a certain individual can undertake. In other words, each individual has a list of tasks that he will have to perform in his daily activities. This list of tasks is changing from time to time to reflect the current state and performance of the individual.

Therefore, the enhanced 911 system must represent all the tasks that associate with a certain role to the operator. This allows him to select appropriate tasks that are associated with the emergency and send them in the query form to the Directory Service. The Directory Service will send back the results of contact phone numbers ordered by the availability of the personnel. The availability here means to do as much as possible to ensure that the callee will pick up the message at the time of need. If the callee is unavailable within the prescribed time, the Directory Service will automatically roll onto the next phone number on the alternative list. The Directory Service will start calling the complaint once a predefined and configurable number of failed attempts have been reached. We will address how to keep individuals tasks up-to-date, consequences, and their requirements, and the implementation of the Directory Service in a subsequent publication.

Furthermore, the current EDXL-DE structure does not address any level of security classification of the emergency. An adversary can take advantage of one of the most vulnerable states, the state of emergency, to launch attacks against our infrastructures, the government, and/or the people. If the emergency operations are discussed with the untrusted individuals who don't have proper access levels, they may launch a counter-attack by disrupting the emergency operations, which in turn cause tremendous delays and cost of lives. Therefore, protecting the emergency discussions, decisions and operations against our enemies during a crisis is critical to secure the national infrastructures and the safety of the people.

As we mentioned in the earlier section, the GETS and WPS capabilities are complements to our work and enhance the availability of the network bandwidth for call establishment. Our work will enhance reaching the appropriate responders and GETS and WPS will help establish the call during the emergency when the network bandwidth and other resources are scarce. For example, during an emergency, the local access network may experience a high call volume and may become congested due to the fix bandwidth. As a result, calls may be dropped randomly or callers may experience high latency. This creates difficulties for emergency personnel to communicate in order to choreograph the emergency handling. If the personnel handling emergencies at the local gas company have the GETS card, they will be able to establish the call with the natural gas expert regardless of the network being congested. This can be easily achieved in three steps: 1) dial 1-710-NCS-GETS, 2) enter their PIN number on the GETS card, and 3) enter the number of the party to be contacted [7]. If the personnel phone number is subscribed to WPS [8], they can easily make the priority call by dialing the number as normal. The service provider network will recognize that the originated phone number has WPS privilege; it will then put the call into the higher priority queue waiting for the available network bandwidth [8].

We can build a quick application ERApp to enhance the use of GETS and WPS by taking a requirement from the users. For example, the user can put a message saying that he needs a gas valve repair technician in zip code 22222. ERApp can translate the text message into a list of attributes and perform the directory lookup on these attributes. With GETS capability, ERApp has been configured with the GETS number with GETS pin to make the priority call to defaultContact. For WPS users, they inherit the priority from the subscription. The ERApp will detect if the attempt to call the defaultContact fails, it will try the list of alternatives. If none of the alternatives responds, it will call the complaint.

ERApp can parse the text message to look for key words. In the message example above, ERApp can extract key words such as gas, valve, technician, and zip code 22222 and then perform translation into the list of numbers. Companies that provide the service will expose their defaultContact, alternatives and complaint numbers along with their key words that reflect the services and the tasks that they can perform. Table 1 illustrates a simple example of the translation table.

We will begin to explore the details of the CAP message and how the systems should react in order to enhance the availability of the callee. In the gas leak example above, there are other CAP messages that the 911 operator or the gas company emergency operator sends to coordinate the evacuation, blockings of the roads in and out of the neighborhood, etc. These CAP messages are extremely important to the emergency coordination, but not relevant to our work. We would like to focus our attention on the CAP message that the gas company emergency operator sends to the Directory Service requesting to speak to the natural gas expert for further instruction and show how the availability of the callee can be achieved. With that in mind, we want to investigate the additional enhancement tags that we add to the CAP message.

At the high level, the gas company emergency operator sends this message to the Directory Service to start dialing the *defaultContact*. The question that may come to mind is, how does the operator retrieve this information? There is the implicit request coming from the operator to the Emergency Contact System (ECS) with certain criteria. The ECS responds with list of contacts ordered by the highest probability that the person will be available and able to address the matter at hand. The Directory Service will start dialing the phone number provided in the *defaultContact* attribute. If the person doesn't answer the phone within the currentResponseDelay, the system will try the list of *alternateContact*. If system fails to establish calls in the predefined and configurable number of attempts or all the contacts in the emphalternateContact list, the *complaintContact* will be used so that the operator can talk to the higher rank official in the organization for further instruction and, of course, to file a necessary complaint of the poor quality of service being provided. The whole purpose of this approach is to minimize the probability of the unavailability of the key personnel.

2.3 Related Works

In recent years, there have been a number of publications on building ontologies to solve different aspects of emergency handling. We discuss a few that are considered to be relevant to our work. Li et al. [46] proposes an ontology for crisis management. Although they define a common set of vocabularies that can be used to facilitate an effective communication, they do not address failure scenarios in reaching key responders in a time of crisis. Yu et al. [47] illustrates a good use of Activity-First Method (AFM) proposed by Mizoguchi [48] to construct an emergency ontology for creating a decision support system from existing emergency documents and use cases. This methodology is aimed at decomposing the emergency documents into data components for further integration based on emergent incidents. Although this emergency ontology helps decision makers sort out existing knowledge and reach critical decisions faster and more efficiently, it does not address how to ensure the availability of decision makers during an emergency.

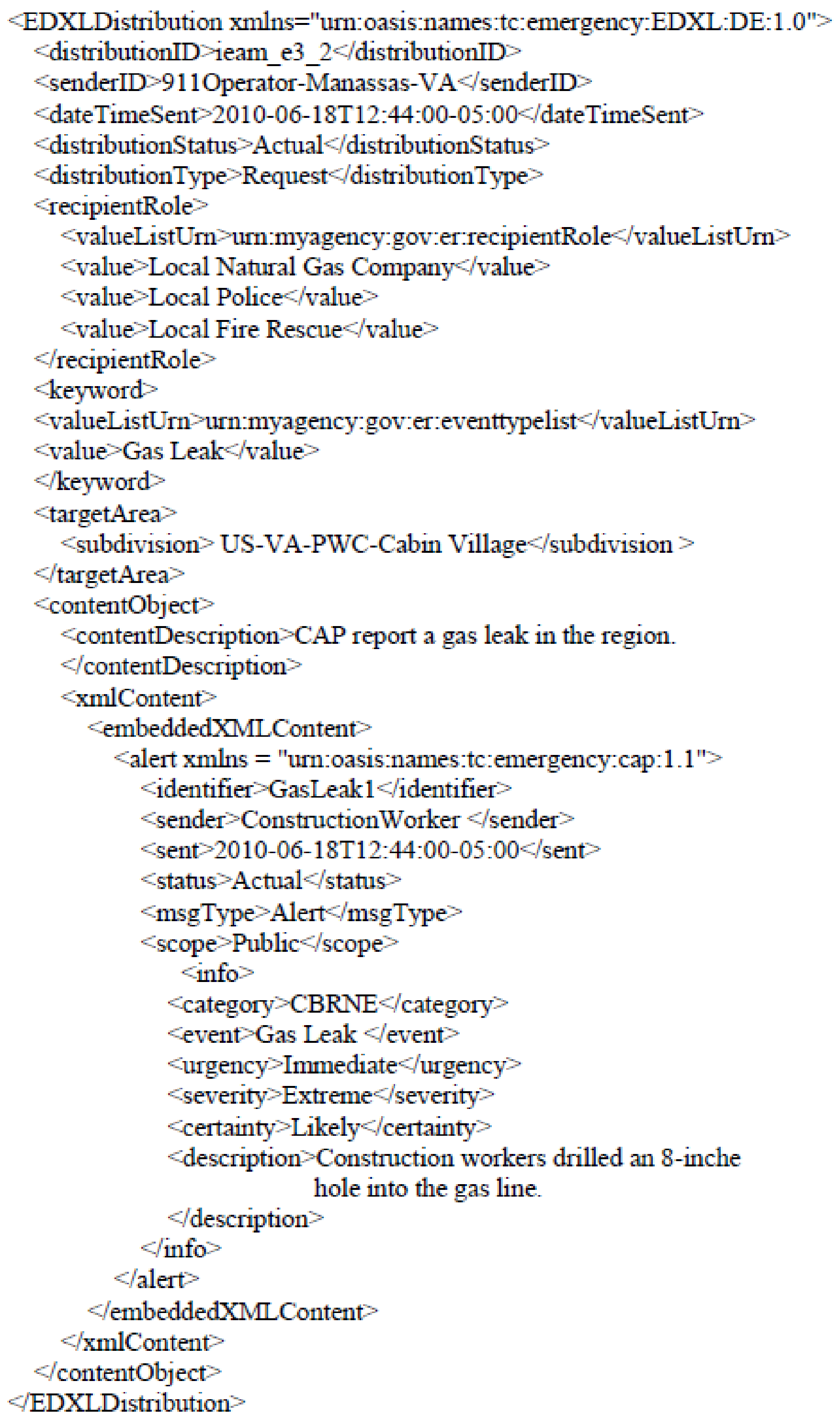
Malizia et al. [49] constructs an emergency ontology for event notification and system accessibility. Using the knowledge that reflects users' needs, ways to present their needs, the nature of the emergency and available technologies makes it possible to reach more people. To build such a complex ontology, the authors use three domain concepts: accessibility, user profiles and devices, and verification of the validity and integrity of knowledge by using first order logic. Although the proposed ontology may address the information needs for sharing and integrating emergency notification messages and provide the accessibility for different kinds of users under different conditions, it does not address the information needs for ensuring the responder's availability at the time of the need. The open ontology approach [50] provides great flexibility to extend into a mission-oriented ontology. In order to do so, an open ontology provides multiple spaces and views that must be taken into account during the design phase. It also provides a theoretical approach to build such an ontology rather than providing a practical open ontology for emergency response. To the best of our knowledge, no one has extended this concept and developed it into a practical open ontology yet.

To facilitate the sharing of information across all levels of government, the Federal Government has initiated the Universal Lexical Exchange (PN-ULEX), which helps define the top sharable objects that can be formed into a coherent message that can be validated via the XML schema [51]. Although ULEX defines sharable contact information, the objective is to provide the contact information for deployable systems and services and not the availability of the contact person during an emergency based on the person's job description. Universal Core (UCore) is another Federal information sharing initiative that supports the national information sharing strategy among all federal departments and agencies. UCore defines an implementable specification in XML schema that enables the information sharing of well-known and comprehensible concepts of who, what, when and where [52]. Although these concepts can address some aspects of information sharing for emergencies, they do not address how the contact would be used to locate the person during an emergency.

The US Federal Government has established a Government Emergency Telecommunications Service (GETS) program [7], which ensures a high probability of call establishment during a crisis when the PSTN is congested. This program provides a specific and recognizable phone number to obtain a higher priority for establishing a call. In recent years, with the increased prevalence of wireless phones, the Federal Government established a Wireless Priority Service (WPS) [8] program, where subscription information is used to identify high priority callers. However, both GETS and WPS services do not guarantee call establishment but rather provide best effort due to the network bandwidth availability. These services are considered complementary to our work on ensuring updated status is maintained regarding the availability of the responder or his alternate.

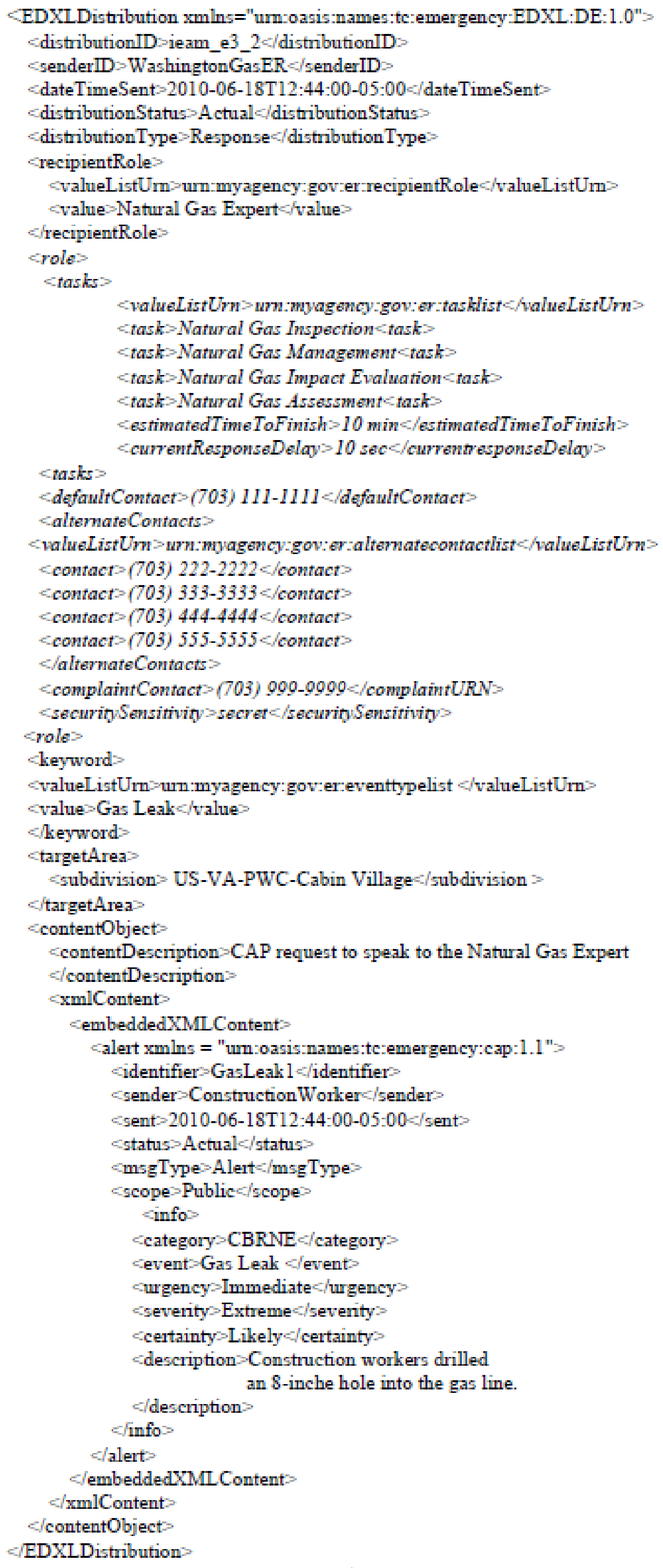
Many standards have been developed by OASIS. These standards have been widely adapted in data communication for emergency handling. One of the recent standard releases is the Common Alerting Protocol (CAP) [5,6], which is the primary communications protocol for exchanging emergency alert messages between different parties. CAP has been used, implemented and deployed by a number of agencies and firms [4]. In this paper, we enhance CAP by adding necessary elements into the CAP schema to enhance reaching the responders in an emergency. We also illustrate the use of these elements in a real life emergency scenario.

Last but not least is the EDXL language, which was developed by OASIS and became a standard in 2006 [2]. We strengthened the EDXL language by adding syntax that can be used to attempt to deliver messages to emergency personnel when the existing mechanisms fail.



```
<EDXLDistribution xmlns="urn:oasis:names:tc:emergency:EDXL:DE:1.0">
   <distributionID>ieam_e3_2</distributionID>
   <senderID>911Operator-Manassas-VA</senderID>
   <dateTimeSent>2010-06-18T12:44:00-05:00</dateTimeSent>
   <distributionStatus>Actual</distributionStatus>
   <distributionType>Request</distributionType>
   <recipientRole>
      <valueListUrn>urn:myagency:gov:er:recipientRole</valueListUrn>
      <value>Local Natural Gas Company</value>
      <value>Local Police</value>
      <value>Local Fire Rescue</value>
   </recipientRole>
   <keyword>
   <valueListUrn>urn:myagency:gov:er:eventtypelist</valueListUrn>
   <value>Gas Leak</value>
   </keyword>
   <targetArea>
      <subdivision> US-VA-PWC-Cabin Village</subdivision >
   </targetArea>
   <contentObject>
      <contentDescription>CAP report a gas leak in the region.
      </contentDescription>
      <xmlContent>
         <embeddedXMLContent>
            <alert xmlns = "urn:oasis:names:tc:emergency:cap:1.1">
               <identifier>GasLeak1</identifier>
               <sender>ConstructionWorker </sender>
               <sent>2010-06-18T12:44:00-05:00</sent>
               <status>Actual</status>
               <msgType>Alert</msgType>
               <scope>Public</scope>
                   <info>
                  <category>CBRNE</category>
                  <event>Gas Leak </event>
                  <urgency>Immediate</urgency>
                  <severity>Extreme</severity>
                  <certainty>Likely</certainty>
                  <description>Construction workers drilled an 8-inche
                                    hole into the gas line.
                  </description>
               </info>
            </alert>
         </embeddedXMLContent>
      </xmlContent>
   </contentObject>
</EDXLDistribution>
```

Figure 2.6: EDXL reportIncident Message



```
<EDXLDistribution xmlns="urn:oasis:names:tc:emergency:EDXL:DE:1.0">
  <distributionID>ieam_e3_2</distributionID>
  <senderID>WashingtonGasER</senderID>
  <dateTimeSent>2010-06-18T12:44:00-05:00</dateTimeSent>
  <distributionStatus>Actual</distributionStatus>
  <distributionType>Response</distributionType>
  <recipientRole>
     <valueListUrn>urn:myagency:gov:er:recipientRole</valueListUrn>
     <value>Natural Gas Expert</value>
  </recipientRole>
  <role>
    <tasks>
              <valueListUrn>urn:myagency:gov:er:tasklist</valueListUrn>
              <task>Natural Gas Inspection<task>
              <task>Natural Gas Management<task>
              <task>Natural Gas Impact Evaluation<task>
              <task>Natural Gas Assessment<task>
              <estimatedTimeToFinish>10 min</estimatedTimeToFinish>
              <currentResponseDelay>10 sec</currentresponseDelay>
   <tasks>
   <defaultContact>(703) 111-1111</defaultContact>
   <alternateContacts>
  <valueListUrn>urn:myagency:gov:er:alternatecontactlist</valueListUrn>
   <contact>(703) 222-2222</contact>
   <contact>(703) 333-3333</contact>
   <contact>(703) 444-4444</contact>
   <contact>(703) 555-5555</contact>
   </alternateContacts>
   <complaintContact>(703) 999-9999</complaintURN>
   <securitySensitivity>secret</securitySensitivity>
  <role>
  <keyword>
  <valueListUrn>urn:myagency:gov:er:eventtypelist </valueListUrn>
  <value>Gas Leak</value>
  </keyword>
  <targetArea>
     <subdivision> US-VA-PWC-Cabin Village</subdivision >
  </targetArea>
  <contentObject>
     <contentDescription>CAP request to speak to the Natural Gas Expert
     </contentDescription>
     <xmlContent>
        <embeddedXMLContent>
           <alert xmlns = "urn:oasis:names:tc:emergency:cap:1.1">
              <identifier>GasLeak1</identifier>
              <sender>ConstructionWorker</sender>
              <sent>2010-06-18T12:44:00-05:00</sent>
              <status>Actual</status>
              <msgType>Alert</msgType>
              <scope>Public</scope>
                  <info>
                <category>CBRNE</category>
                <event>Gas Leak </event>
                <urgency>Immediate</urgency>
                <severity>Extreme</severity>
                <certainty>Likely</certainty>
                <description>Construction workers drilled
                              an 8-inche hole into the gas line.
                </description>
              </info>
           </alert>
        </embeddedXMLContent>
     </xmlContent>
  </contentObject>
</EDXLDistribution>
```

Figure 2.7: EDXL 1.8 directortServiceRespond Message

Chapter 3: Enhancing the Commercial Mobile Alert System

3.1 SMS For Emergency Handling

Informing the public about the nature of an emergency has been a problem for many years. The existing communication mechanisms such as television, public radio, and online news websites only reach a small percentage of the population at any given time. Americans 15 years and older spend “only” 6.12% of their time watching TV [20]. Americans 12 years and older spend 11.2 % and 7% of their time listening to commercial radio [22] and National Public Radio (NPR) [21] at least once a week. Therefore, existing communication mechanisms have their own limitations due to cultural and social aspects. However, over 91% of the US population subscribes to a wireless service [23], which is an effective means of communicating to the general public. In addition, the users carry their wireless devices almost everywhere they go. Different types of alert settings such as vibration or ring-tone can get the users’ attention almost immediately.

In emergency situations, the traditional ways of informing the public about the nature of an emergency have used television and radio network broadcasts. With the proliferation of internet services, emergency information can be published on news websites and delivered right onto users’ wireless devices through a simple registration process. After the 9/11 terrorist attack in Washington, DC, a survey was conducted to assess the areas needing improvements, revealing that even though the voice calls in and out of the damaged area were highly congested, people were still able to use text messaging to communicate with each other [1]. Short Message Service (SMS) popularity has been increasing and becoming a primary means of communication in our society, especially among the younger generation [17]. A few years ago, service providers generated a fixed amount of revenue, which could be up to $.20 per text message sent and received [14, 15]. Furthermore, many service providers have used this SMS capability as a free value-added service to attract new customers [16].

3.2 Objectives and Limitations

Understanding the growth, popularity, and vulnerability [24] of SMS, the Department of Homeland Security initiated the Commercial Mobile Alert Service (CMAS) [19] to broadcast emergency alert text messages to the public. Service providers and equipment vendors help define standards and implementations for CMAS service. Unlike sender-to-receiver SMS, CMAS uses a dedicated Primary Broadcast Control Channel (BCCH) [25] to broadcast the text message, which can reach millions of wireless subscribers within minutes. While CMAS is still in the design and development phase to address the communication aspect of sending emergency alerts to the general public, it inherits a few weaknesses from the Cellular Broadcast Service and the existing protocol handling emergency communications:

1. The CMAS alert message cannot be broadcast to an area smaller than a cell site, which is defined in the Federal Information Processing Standard (FIPS) code [33]. This area varies in sizes, depending on the population density and is too large to receive the broadcast alert for a small-scale emergency. This limits the CMAS usage. The cell site's coverage area is too large to receive an emergency alert for a small-scale emergency such as burning building, apartment power outage, etc. For example, if the CMAS system was available during the 9/11 terrorist attack in the Washington, DC area, it would have sent out an emergency alert to inform all the mobile users working at the Pentagon and the surrounding areas to evacuate immediately.

2. According to the CMAS specification [9], the Common Alerting Protocol (CAP 1.2) will be used to communicate CMAS emergency alerts. However, CAP 1.2 was designed for emergency communication between different levels of government such as federal, state and local, departments and agencies. Most of information in the CAP 1.2 message structure was irrelevant for emergency mobile broadcasting. In addition, the

existing information in the CAP 1.2 message structure does not fully address the essence of all local emergencies. While service providers working with Department of Homeland Security define the message format and structure for CMAS, we took the initial step to refine the CAP 1.2 message structure for emergency mobile broadcasting.

3. CMAS will disseminate three types of alerts: Presidential Alert, Imminent Threat Alert, and America's Missing: Broadcast Emergency Response (AMBER) Alert [26]. CMAS is not designed to broadcast alerts for local emergencies.

4. The last limitation on CMAS is on the broadcast message size, which allows up to 90 characters of clear text [9]. Clearly, this limitation restricts emergency responders to disseminate necessary information of ongoing emergencies.

3.3 Enhancing Cell Broadcast Service for Local Emergency

In this section, we provide the detailed enhancement to CMAS to deliver an alert message to a smaller area than a cell site or FIPS code equivalent.

3.3.1 Considerations for Our Enhancement

In 2003, OASIS sponsored the Common Alerting Protocol (CAP) initiative with the objective of providing fundamental messaging protocols to facilitate interagency emergency communications. The OASIS technical committee on emergency management has developed a set of standards [2, 5], which consist of a set of XML based tags to exchange the information needed to handle an emergency. However, CAP is intended to communicate emergency incidents between emergency systems and operators, etc. It does not intend to broadcast emergency alerts to a large audience. For example, the *Sender ID* in CAP is irrelevant to users who receive the broadcast alert message.

We have investigated the technical specifications of the GSM and UMTS Cell Broadcast Service (CBS) [9] and found that there are no features or options that could be configured to support the broadcast of alert messages for a smaller area than a cell site. However, with significant improvements on the mobile computing platforms, we built an Emergency Response Application (ERApp) to intercept the CMAS CBS alert message and filter it based on the users' locations with respect to the emergency incident's location. To achieve this, we enhanced the CAP message structure to include three additional tags: affected area, spreadable, and location. We will describe the details of the CMAS alert message structure in the next section.

3.3.2 CMAS Alert Message Structure

There are two ways that the CMAS alert message can be sent. First, the message can be sent via raw text. Each of the values can be separated by a delimiter. This method won't give us much space to squeeze necessary datum into the message. Second, the message can be sent using the existing emergency messaging standard, CAP 1.2 [6]. However, the CAP message is in XML format, which requires more space to store an XML alert message. Figure 3.1 shows the CAP 1.2 Entity Diagram.

CAP 1.2 is designed for emergency communication between different levels of government such as federal, state and local, departments and agencies. With the message size limitation from the GSM and UMTS Cell Broadcast Service [9], we need to examine what data fields in the CAP 1.2 are needed. For example, when a tornado occurs in Louisville, Kentucky, the city of Louisville sends an alert message to the state of Kentucky requesting resources for rescue operations. The state emergency operation center needs to validate the sender to see if he is legitimate and that the message is not a hoax. To perform this validation, the state emergency system needs to know the message id, sender id, sent date/time, scope, etc. Conversely, mobile users receiving a CMAS alert message don't need to know the origin because they have no way of validating the message, but rather, they need to know what the emergency is about, where it is, and how to avoid its impact. Therefore, much of the information provided in the CAP 1.2 schema is irrelevant to the CMAS alert message. We enhanced the CAP 1.2 message structure for the CMAS alert message. The enhanced CAP 1.2 message structure is depicted in Figure 3.2 above.

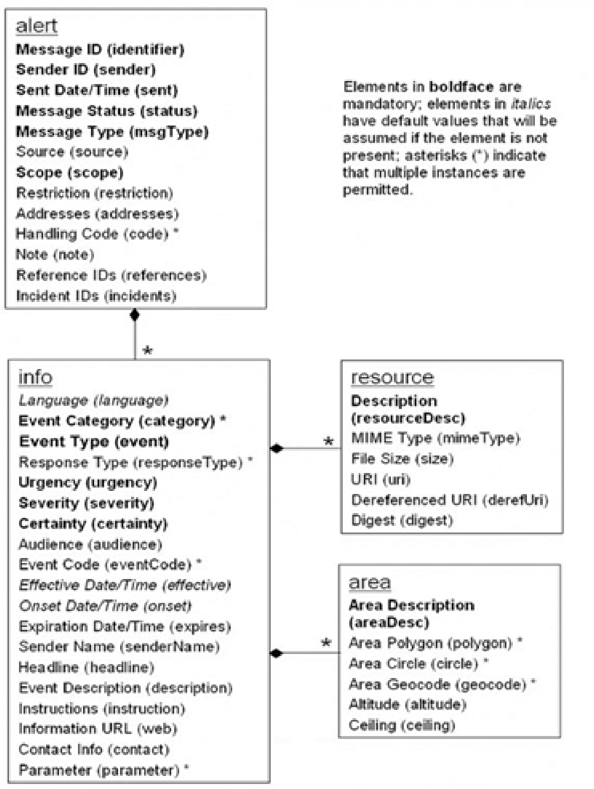


Figure 3.1: CAP 1.2 Alert Message Structure

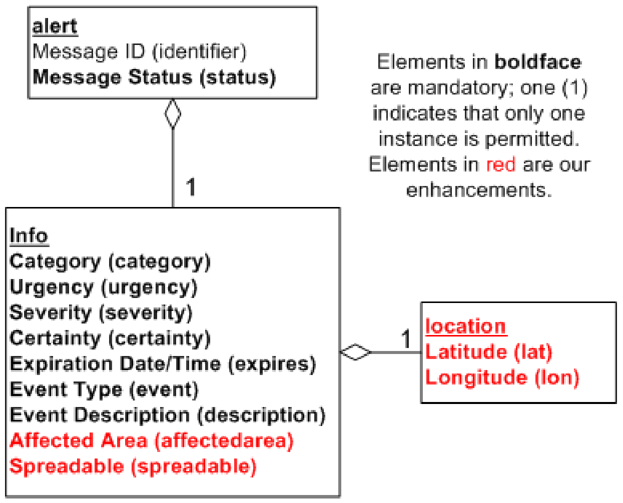

alert
Message ID (identifier)
Message Status (status)
Elements in boldface are mandatory; one (1) indicates that only one instance is permitted. Elements in red are our enhancements.
1
Info
Category (category)
Urgency (urgency)
Severity (severity)
Certainty (certainty)
Expiration Date/Time (expires)
Event Type (event)
Event Description (description)
Affected Area (affectedarea)
Spreadable (spreadable)
1
location
Latitude (lat)
Longitude (lon)

Figure 3.2: Enhanced CMAS CAP 1.2 Alert Message Structure

We extracted relevant information from the existing emergency messaging standard CAP 1.2 that would be beneficial to the mobile users. In addition, we made some new enhancements by adding three tags: affected area, spreadable, and emergency location. The *Affected Area* is a circumference boundary measured in meters from the emergency location. This value entered by the emergency coordinator depends upon the nature of the emergency. All mobile users within this boundary will receive the CMAS alert message. Depending on the devices' configurations, the users may be alerted via the vibration and/or the ring tone. The *spreadable* tag is important when the emergency is about biological, air pollution, or nuclear attacks. It is a clear indication to users that they need protections such as gas masks or anti-pollution filters. *Location* indicates where the emergency is in terms of the latitude and longitude, which can be easily shown on the map. Figure 3.3 illustrates a CMAS mobile message of a tornado alert.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<alert xmlns = "urn:oasis:names:tc:emergency:cap-cmas">
  <identifier>CMAS-01</identifier>
  <status>Actual</status>
  <info>
     <category>Met</category>
     <urgency>Expected</urgency>
     <severity>Severe</severity>
     <certainty>Observed</certainty>
     <expires>2010-10-02T17:00:00-0500</expires>
     <description>Multiple tornados are expected around
                 2PM in the Washington, DC area.
     </description>
     <affectedarea>1000</affectedarea>
     <spreadable>No</spreadable>
     <event>Tornado</event>
     <location>
        <lat>38.882334</lat>
        <lon>-77.171091</lon>
     </location>
  </info>
</alert>
```

Figure 3.3: CMAS Mobile Alert Message - Tornado Example

3.3.3 Area Enhancement for Small-Scale Emergency

In this section, we will discuss our proposed area enhancement for the small-scale emergency. Proposing an enhancement in the Cell Broadcast Service is a time-consuming effort. There are ongoing improvements and empowerments in mobile devices in terms of processing power, battery lifetime, sensors, etc. Therefore, we proposed to build an alternative, Emergency Response Application (ERApp), to assist mobile users in all areas of emergency.

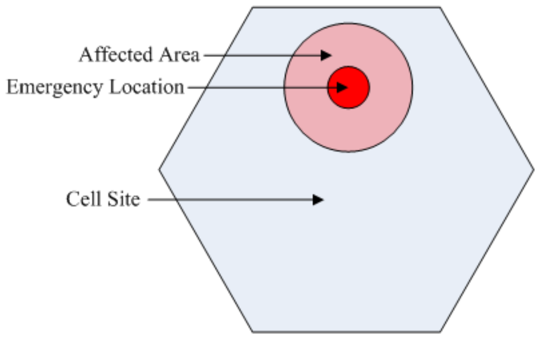


Figure 3.4: Typical Emergency Scenario

There are a variety of features built into ERApp to enhance users' emergency experiences. However, we will only discuss the area enhancement and the alerting features of the ERApp in this paper and leave other features to be published at a later time.

The ERApp is a mobile application built in Java language on top of the Google Android API 2.2 platform. The ERApp works in conjunction with the ERAlert System, a CMAS prototype implementation, which simulates the broadcast of the CMAS alert message to mobile phones using the email texting feature.

When mobile users launch the ERApp, it will perform two updates: (1) update the local repository at the non-emergency time, and (2) retrieve and display on the map their current GPS locations. This GPS location will be updated at a regular basis, based on two configurable parameters, time and distance, that allow an infrequent query of the device's GPS location in order to save battery life. By default, the time is set to 60 seconds and the distance to 50 meters. The location update is triggered when either the 60-second time is expired or the user has gone 50 meters from the location that has been recorded last. The GPS estimates the movement of the user (assuming that the user carries the device at all time) in order to advises him to leave the area that is being affected by the prevailing emergency. The ERApp receives an alert message from the ERAlert System in a form of a byte-encoded XML. The ERApp will have to decode the message into its original XML form.

It will locate and display the emergency on the map using the location tag information. In addition, the ERApp will display the user's current position with respect to the position of the emergency. If the phone's position is within the affected area, the ERApp will alert the users using the device alert settings such as ring tone or vibration and display the alert message. The users must acknowledge within a configurable time. Otherwise, the ERApp will keep vibrating or turning on the ring tone until the user acknowledges the alert. If the user's position is outside the affected area and the ERApp detects the user moving toward the affected area, it will alert the user by turning on the vibration or the ring tone and display the alert message. After acknowledging the alert, the user will be able to view the map of their current positions, the emergency location, and the affected area. Figure 3.4 above illustrates the cell site, affected area and the emergency location.

3.4 Prototype Implementation

This section discusses the details of the prototype implementation. We will also discuss the challenges encountered during the demonstration.

3.4.1 Prototype Development Environment

We implemented this prototype in Java. The client Graphical User Interface (GUI) was developed using a Java Applet. The service engine was developed in Java Web Services running in JBOSS Community Server 5.1.0 GA with MySQL server 5.1 as the backend database. The entire ERAlert project was developed in NetBeans IDE 6.9.1. We self-sign all of the Java Archive (JAR) files with our own certificate to ensure that the JAR files come from a trusted source.

When we implemented this ERAlert prototype, we knew ahead that the Cell Broadcast Service (CBS) has been a standard for a long time but has not been built into the carrier networks in the United States. Testing the CBS functionality is only simulated. To simplify the testing process, we use the email texting feature that is offered by service providers.

3.4.2 Demonstration

In this section, we walk through an example of a tornado in Arlington, VA and how the emergency coordinator composes a CMAS local alert message and broadcasts it to a specific affected area.

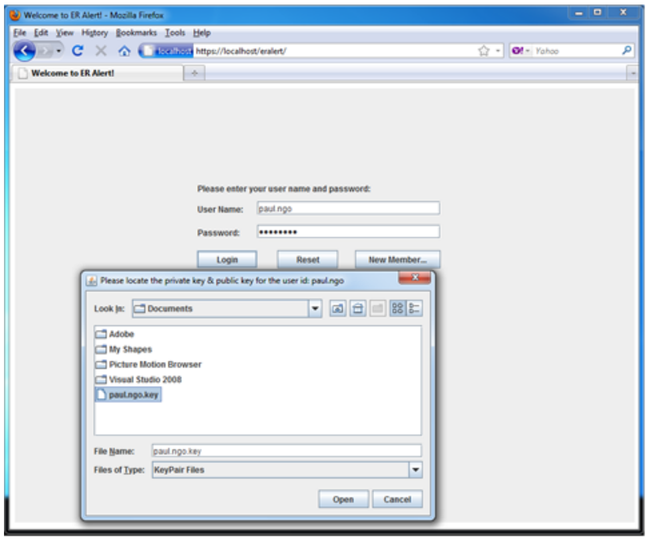
3.4.3 Login Phase

First, the emergency coordinator must register with the ERAlert System. The registration takes two simple steps: receiving the user information and saving the PKI keys. The PKI keys will be used to encrypt and decrypt messages between the Web Interface and the backend server to avoid any security implications such as man-in-the-middle attack or password hijack. We save the PKI certificate on the local system in this prototype implementation to illustrate that we implemented the two-factor authentication. However, in the deployed system, PKI certificates can be stored on a Common Access Card (CAC) or on secured Universal Serial Bus (USB) thumb drives. By default, the password requirement is two lowercase alpha characters, two uppercase alpha characters, two numeric characters, and two special characters. This password requirement can easily be changed in the implementation depending on the need of the deployable system.

Figure 3.5 depicts the login screen for a registered user. He will enter his user identifier and password and click on the Login button. The system will then prompt him to locate his PKI-key file. Once he locates the file, he clicks on the Open button. Upon successful login, ERAlert map screen will show up as depicted in Figure 3.6.

Local Alert Generation Phase

The emergency coordinator will enter the event location and click on the *Go* button, which pinpoints the emergency location on the map. He will then enter all the necessary information such as alert type, event type, message, spreadable, affected area, expired time, category, status, urgency level, severity, and certainty. As soon as he enters the radius for the affected area, the red transparent circle around the emergency location will show up on


Welcome to ER Alert! - Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://localhost/eralert/
Welcome to ER Alert!
Login
Reset
New Member...
Open
Cancel

Please enter your user name and password:

User Name: paul.ngo

Password: ••••••••

Please locate the private key & public key for the user id: paul.ngo

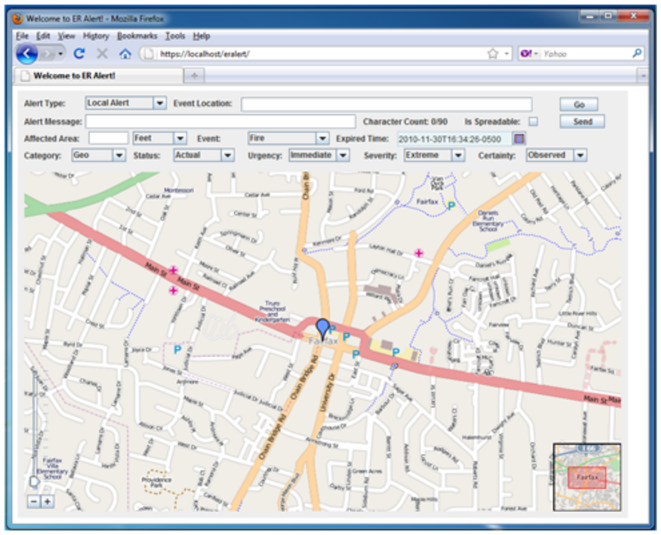
Look In: Documents

Adobe
My Shapes
Picture Motion Browser
Visual Studio 2008
paul.ngo.key

File Name: paul.ngo.key

Files of Type: KeyPair Files

Figure 3.5: ERAlert Login and PKI Certificate


Welcome to ER Alert! - Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://localhost/eralert/
Welcome to ER Alert!
Go
Send
Main St
Chain Bridge Rd
University Dr
Fairfax

Alert Type: Local Alert

Event Location:

Alert Message:

Character Count: 0/90

Is Spreadable:

Affected Area: Feet

Event: Fire

Expired Time: 2010-11-30T16:34:26-0500

Category: Geo

Status: Actual

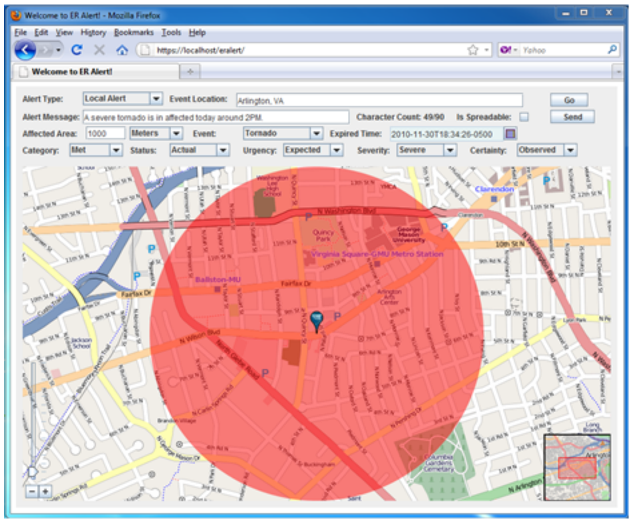
Urgency: Immediate

Severity: Extreme

Certainty: Observed

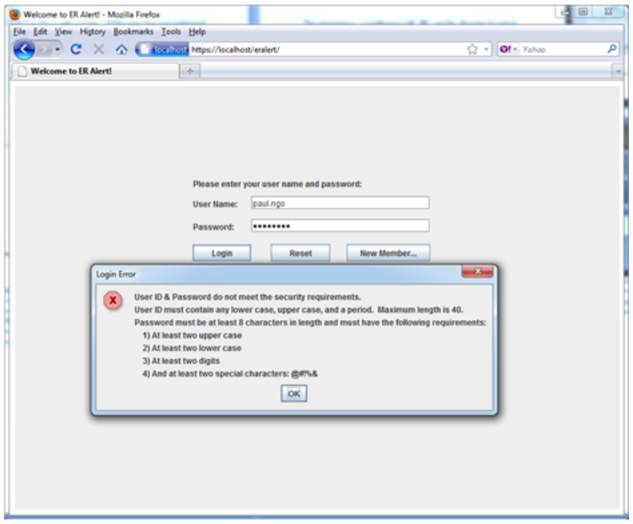
Figure 3.6: ERAlert Map Screen

the map as depicted in Figure 3.7. Finally, he clicks on the *Send* button to broadcast the alert message to all the mobile users in the affected area.


Welcome to ER Alert! - Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://localhost/eralert/
Welcome to ER Alert!
Go
Send

Alert Type: Local Alert
Event Location: Arlington, VA
Alert Message: A severe tornado is in affected today around 2PM.
Character Count: 49/90
Is Spreadable: ☐
Affected Area: 1000 Meters
Event: Tornado
Expired Time: 2010-11-30T18:34:26-0500
Category: Met
Status: Actual
Urgency: Expected
Severity: Severe
Certainty: Observed

Figure 3.7: ERAlert CMAS Alert Message


Welcome to ER Alert! - Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://localhost/eralert/
Welcome to ER Alert!
Login
Reset
New Member...
Login Error
OK

Please enter your user name and password:
User Name: paul.ngo
Password: ••••••••

User ID & Password do not meet the security requirements.
User ID must contain any lower case, upper case, and a period. Maximum length is 40.
Password must be at least 8 characters in length and must have the following requirements:
1) At least two upper case
2) At least two lower case
3) At least two digits
4) And at least two special characters: @#%&

Figure 3.8: ERAlert Login Error Message

Mobile Application ERApp Alert Handling Phase

Once the emergency alert is broadcast to a particular cell site, all the mobile phones in that cell site will receive the alert [9]. The ERApp will listen to the Broadcast Control channel and capture the alert. It will decode the alert to extract the emergency location, the affected area, the event type, etc. The ERApp will display the emergency location, the user's current position, and the affected area. It calculates the distance between the user current location and the emergency location. If the distance is less than the affected area's radius (the user is inside the affected area), the ERApp will display the alert and wait for the user to acknowledge it. While displaying, the ERApp uses the alerting phone settings by beeping and/or vibrating. The user has to acknowledge the alert within a reasonable time. If not, the phone will keep beeping and/or vibrating until the user does acknowledge. If the user is outside the affected area, the ERApp will store the alert on the phone until its expiration time is reached. The ERApp will use the GPS to capture the user's location. If the ERApp detects that the user is moving into the affected area and that the emergency is still in effect, i.e., the alert expiration time has not been reached yet, the ERApp will alert the user with a vibration or beep. The user is outside the emergency affected area depicted in Figure 3.9. Figure 3.10 illustrates the user when he is inside the affected area and the alert is displayed.

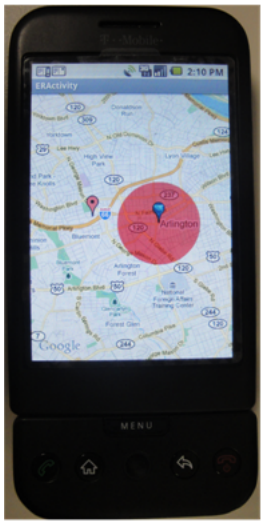
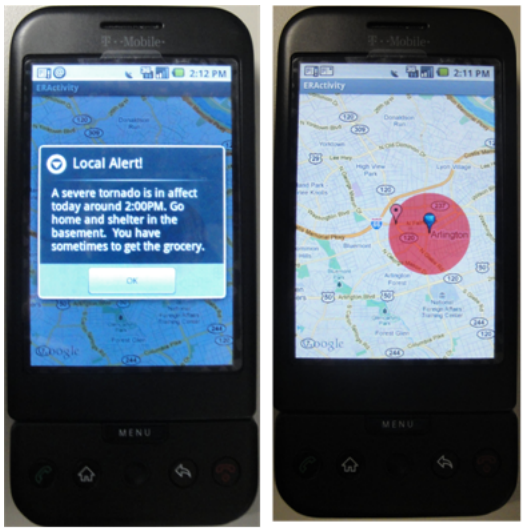


Figure 3.9: Off the Emergency Affected Area



Local Alert!

A severe tornado is in affect today around 2:00PM. Go home and shelter in the basement. You have sometimes to get the grocery.

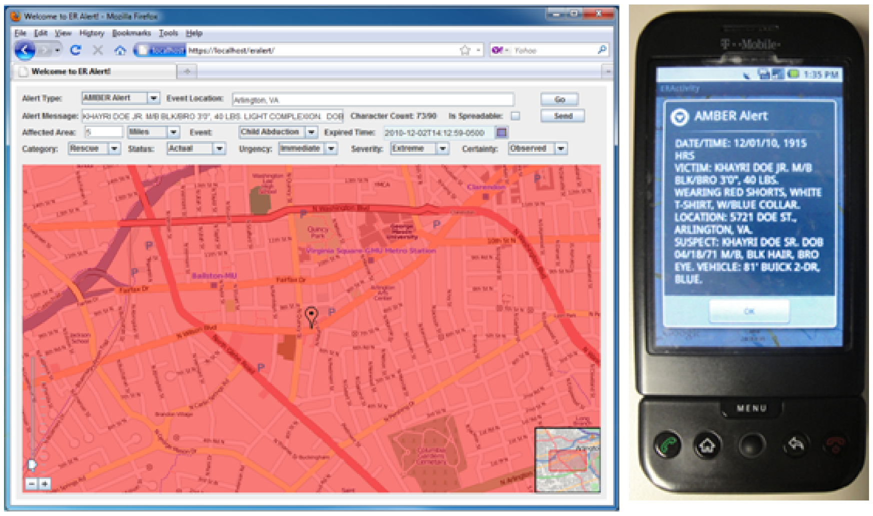
Figure 3.10: Inside the Emergency Affected Area

3.4.4 AMBER Alert Generation Phase

In addition, the prototype also handles other CMAS alert types such as Presidential, Imminent, and AMBER. Due to the amount of information pertaining to the AMBER alert, sending one broadcast message is not enough. The CMAS prototype breaks the AMBER information into small CMAS AMBER alerts and broadcasts them. Upon receiving AMBER alerts, the ERApp will assemble them back into the original CMAS AMBER alert and display it as depicted in Figure 3.11.

3.5 Enhancing the CMAS Message

In this section, we provide an enhancement to CMAS by using different techniques of encoding to deliver detailed emergency information messages, which can be used to assist wireless users in the emergency. This assistance can be varied depending on the nature of the emergency.


Welcome to ER Alert! - Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://localhost/eralert/
Welcome to ER Alert!
Go
Send
T-Mobile
1:35 PM
ERActivity
OK
MENU

Alert Type: AMBER Alert
Event Location: Arlington, VA
Alert Message: KHAYRI DOE JR. M/B BLK/BRO 3'0", 40 LBS. LIGHT COMPLEXION. DOB
Character Count: 73/90
Is Spreadable:
Affected Area: 5 Miles
Event: Child Abduction
Expired Time: 2010-12-02T14:12:59-0500
Category: Rescue
Status: Actual
Urgency: Immediate
Severity: Extreme
Certainty: Observed

AMBER Alert

DATE/TIME: 12/01/10, 1915 HRS
VICTIM: KHAYRI DOE JR. M/B BLK/BRO 3'0", 40 LBS.
WEARING RED SHORTS, WHITE T-SHIRT, W/BLUE COLLAR.
LOCATION: 5721 DOE ST., ARLINGTON, VA.
SUSPECT: KHAYRI DOE SR. DOB 04/18/71 M/B, BLK HAIR, BRO EYE. VEHICLE: 81' BUICK 2-DR, BLUE.

Figure 3.11: AMBER Alert

3.5.1 Consideration for CMAS Message Encoding

The CMAS CAP 1.2 message was built on the XML structure, which is verbose and requires extra space for XML tags and values. Sending the 90-character XML message alert won't give wireless users a meaningful alert message. Therefore, we propose an encoding scheme that can be used to encode more emergency data. ERApp, upon receiving the CMAS broadcast alert message, will decode the message back into its original XML alert message. We will describe the details of the CMAS encoding scheme in Section 3.5.3.

3.5.2 CMAS Architectural Enhancement

For the encoding technique to work properly, we propose an architectural enhancement to CMAS, which occurs during the handover time. The ERApp on mobile devices need a `codepage` in order to decode the emergency alert. A codepage contains a list of emergency message formats and region-specific emergency tag repositories, which can be identified by the unique XML schema's namespace. This unique Uniform Resource Identifier (URI) or Uniform Resource Name (URN) in the namespace can be located in the emergency XML alert message. Each emergency tag repository contains the location-specific list of emergency name-value pairs. The name includes the XML emergency tag and attribute names and the value is its equivalent byte representation. The codepage can be downloaded automatically to the device during the handover time, which requires a minor enhancement to the handover procedure. To simplify our discussion, we consider the Global System for Mobile Communications (GSM). Other networks such as Code Division Multiple Access (CDMA), Universal Mobile Telecommunications System (UMTS), Long Term Evolution (LTE), etc., have a similar handover procedure.

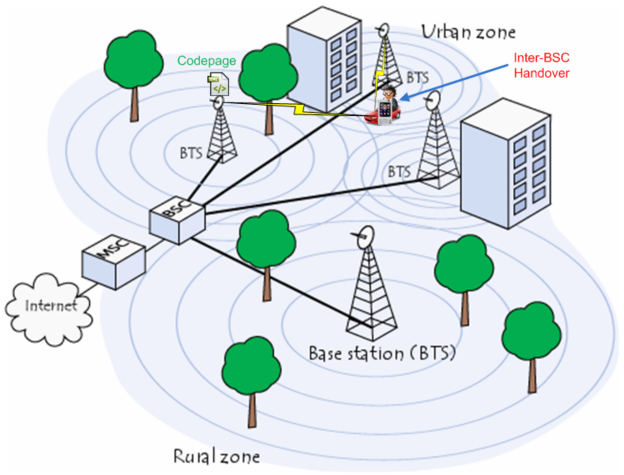


Figure 3.12: Handover Enhancement

There are four types of handover that can be performed in the GSM system: Intra-BTS handover, Inter-BTS Intra BSC handover, Inter-BSC handover and Inter-MSC handover [27]. These four types of handovers have one significant similarity, which is the synchonization between the Mobile Station (MS) and the Base Transceiver Station (BTS) that occurs while users are either on the phone or off the phone while switching. We transmit the codepage from the BTS to the MS during the synchronization of the old and new BTSs [27]. Figure 3.12 illustrates a user driving a car with an Android phone from an urban to a rural area during which the inter-BSC handover occurs and the new codepage from the rural area is sent to the user's Android phone during the handover synchronization process.

A second approach is to request the codepage based on the GPS location upon the installation of our proposed ERApp. This can happen at non-emergency time, when the bandwidth is available for downloading the codepage.

Due to having the ability to decode the emergency alert in the ERApp, we propose two encoding/decoding methods: *predefined* and *just-in-time* methods. The predefined encoding provides that the ERAlert and ERApp must follow the exact same message format specified in the codepage, which must be communicated prior to the broadcast of the alert message. One major advantage of this predefined method is that it does allow more space for more emergency data because we do not encode the tag and attribute names. A good candidate for predefined encoding method is the AMBER alert. However, this method suffers from the loss of flexibility in including or excluding specific tags or attributes associated with a specific emergency.

Just-in-time encoding provides that ERAlert doesn't follow any alert message formats as long as the alert message is in compliance with the emergency tag repository provided in the codepage. The just-in-time encoding requires more space to include the tag and attribute names than the predefined encoding.

3.5.3 CMAS Encoding Schemes

The CMAS encoding scheme is the combination of encoding techniques such as WBXML [28], Prime Power [29], Base64 [30], etc. Before we describe the encoding algorithm, we need to describe the CMAS message data fields and their values. Figure 3.13 illustrates the CMAS CAP message enhancement data fields.

Data fields such as category, urgency, severity, and certainty are taken directly from the CAP 1.2 message. These data fields have a set of static and predefined values. In addition, the *disasterType* field also have static and predefined set of values as well in the CAP 1.2 enhancement for CMAS alert. Therefore, we can encode these data fields using our enhanced WBXML. Before we describe our encoding scheme, we would like to describe briefly two related encoding techniques to show how we can leverage these encoding techniques for our

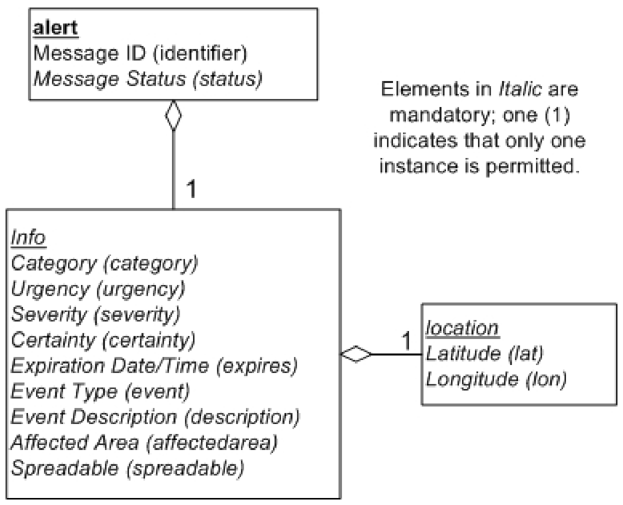


Figure 3.13: CAP Message Structure enhanced for CMAS

proposed encoding.

WBXML Encoding

The WBXML encoding scheme converts all XML tags and attributes to bytes. The XML tag has the begining tag, the value, and the ending tag such as *<category>Met</category>*. The WBXML encodes one byte for the beginning tag, one byte for the value, and one byte for the ending tag. A similar method is used to encode an attribute. Algorithm 1 is the WBXML pseudo code.

Algorithm 1 : WBXML Encoding Algorithm (Input: XML Stream)

```
Require: $xmlStream \neq null$
1: $tokenSteam \leftarrow new\ ByteArrayOutputStream()$
2: $handler \leftarrow new\ WBXMLContentHandler()$
3: $reader \leftarrow XMLReader.createXMLReader()$
4: $reader.setContentHandler(handler)$
5: $xmlSource \leftarrow new\ InputSource(xmlStream)$
6: $reader.parse(xmlSource)$
7: $tokens \leftarrow$handler.getTokens()
8: while $tokens.hasNext()$ do
9:    $aToken \leftarrow (Token)tokens.next()$
10:   $tokenStream.write(aToken.getValue())$
11: end while
12: return $tokenStream$
```

The WBXML algorithm requires a valid XML stream (input parameter). It creates a new token stream in line 1 as the byte array output stream to store byte values as the result of the encoding. In line 2, the algorithm creates the WBXML handler, which implements all the callback methods such as beginning tag, closing tag, text value, etc., and loads the codepage based on the namespace specified in the XML alert message. In line 3, the algorithm creates the XML Reader `reader` and registers the WBXML handler with the XML reader in line 4. In line 5, the algorithm creates the input source from the XML stream, which is required as the input parameter for the `reader.parse` method in line 6. The `reader.parse` method visits all the tags and attributes in the pre-order fashion. When the parser identifies either beginning or closing tag name, it calls the appropriate callback methods defined in the `handler`, which looks up the tag name in the tag repository and adds a byte token (encoded byte value for a tag) into the list. In the end, the `handler.getTokens` in line 7 call returns the list of byte tokens to be written into the token stream. From line 8 - 11, the algorithm loops through the list of tokens and writes out the encoded byte values to the output token stream, which is returned in line 12.

We find that this encoding technique is useful for our purpose because it encodes values of known tags and attributes. But if we already have the list of fields and their known values ahead of encoding time and the order in which they are encoded, we can carefully craft our encoding technique to eliminate the last ending tag and thereby save space.

Prime Power Encoding

The Prime Power encoding method is able to encode any XML document into a single (but numerically large) integer. However, encoding and decoding processes may take a lot of CPU power and time to encode and decode a simple XML document. We have implemented a Prime Power encoder to test its feasibility. The encoding time would depend on the depth and flatness of the XML document, and our experiments took hours to encode a 10-tag XML document. Consequently, we concluded that Prime Power encoding is inefficient and infeasible to encode the CMAS alert message. Algorithm 2 is the Prime Power pseudo code.

Algorithm 2 : Prime Power Encoding Algorithm (Input: XML Stream)

```
Require: $xmlStream \neq null$
1: $handler \leftarrow new\ XMLPPContentHandler()$
2: $reader \leftarrow XMLReader.createXMLReader()$
3: $reader.setContentHandler(handler)$
4: $xmlSource \leftarrow new\ InputSource(xmlStream)$
5: $reader.parse(xmlSource)$
6: return $handler.getPPValue()$
```

Similar to the WBXML algorithm, the Prime Power algorithm also requires a valid XML stream as the input parameter. In line 1, the algorithm creates the XML Prime Power Content Handler (XMLPPContentHandler). The `handler` will implement all the callback functions and load the codepage based on the namespace in the XML alert message. The algorithm then creates the `reader` (XML Reader) in line 2. The `handler` is then registered with the `reader` in line 3 so that callback functions will be called when the XML tags are encountered. The `xmlSource` is created from the `xmlStream` in line 4. The `reader.parse` method [31] will perform a post-order visit from the leaf node up to the root node in line 5. The first three prime numbers: 1, 2, and 3 are reserved, first for the default node prime, second for internal node, and third for the leaf node. At every node and node value, the parser will perform the prime encoding by taking a prime number from the smallest available primes and raise it to the power of the equivalent integer value that represents it in the tag repository. For example, let's consider the severity tag in the example illustrated in the listing 3.1. The severity tag has an integer value of 2 (two) and its Severe value of 2 (two). To encode the severity tag, the parser will first encode the severity tag value. Because the severity value is the leaf node, the parser will raise the leaf prime number 3 (three) to the power of 2 (two). The integer value so far is 9 (nine). The parser will continue to encode the severity tag by encoding the severity tag itself, which can be calculated by raising the internal node prime of 2 (two) with the tag integer value of 2 (two). Therefore, the severity tag alone has the integer value of 4 (four). To complete the encoding, the parser will multiply the tag integer value, which is 4 (four) to a number of 1953125 which can be calculated by raising the next available prime of 5 (five) to the tag value integer of

9 (nine). Therefore, the prime power encoding for the severity tag has a (big) integer value of 7812500. The parser will repeat this encoding process. This big integer will be extremely large and become unmanageable quickly. Finally, the handler returns the final encoded big integer value that represents the XML document in line 6.

```
<?xml version = ”1.0” encoding = ”UTF−8”?>
<alert xmlns = ”urn:oasis:names:tc:emergency:cap−cmas”>
  <identifier>CMAS−01</identifier>
  <status>Actual</status>
  <info>
     <category>Met</category>
     <urgency>Expected</urgency>
     <severity>Severe</severity>
     <certainty>Observed</certainty>
     <expires>2010−10−02T17:00:00−0500</expires>
     <description>Multiple tornados are expected around
                  2PM in the Washington, DC area.
     </description>
     <affectedarea>1000</affectedarea>
     <spreadable>No</spreadable>
     <event>Tornado</event>
     <location>
        <lat>38.882334</lat>
        <lon>−77.171091</lon>
     </location>
  </info>
```

```
</alert>
```

Listing 3.1: CMAS Mobile Alert Message - Tornado Example

CMAS Encoding

In order to provide more meaningful CMAS alert message for XML trees, we created our own XML encoding scheme, which significantly reduced the number of bytes required to encode. In addition, our method can be extended to encode more complex XML schemas by pre-computing the maximum (tree) depth of the XML schema and the maximum number of known values for each (tag/attribute, value) pair. The *just-in-time* method provides more flexibility than the *predefined* method in which ERAlert does not have to define and communicate the message format to the ERApp prior to the sending of the alert message. Detailed algorithms for both methods are described in the following section.

Just-in-time Encoding Our just-in-time encoding algorithm has two phases: (1) preprocessing phase and (2) encoding phase. The preprocessing phase will build the XML tag repository from the XML schema. For each depth of the XML tree, the preprocessing phase examines all possible tag names and associates each tag name with an integer value starting at zero (0). This process will be repeated for all the tag values, attribute names and attribute values.

The preprocessing phase generates a codepage for all the region-specific tag repositories. Each tag repository can be identified and retrieved by the unique XML schema's namespace. The codepage it generates will be location-specific, pertaining to the emergencies that occur regularly in that region. For example, the codepage in Florida can describe *tsunami*, *hurricane*, and *tornado* related tags because the state of Florida has a long coast line. However, the codepage in California may describe only *earthquakes* and *wildfires*.

As described in Algorithm 3, the preprocessing phase will use the depth, and the maximum number of tags, tag values, attributes names, and attributes values to determine the encoding scheme. Furthermore, the preprocessing phase examines all the tags that have static values and consolidates them to fit in the selected encoding schema. This process will generate the combined tag in the tag repository, which will reduce the number of encoding bytes significantly.

Algorithm 3 : Encoding Scheme Algorithm (Input: XML Emergency Schema)

```
Require: $xmlSchemaStream \neq null$
 1: $depth \leftarrow getDepth(xmlSchemaStream)$
 2: $maxTags \leftarrow getMaxTags(xmlSchemaStream)$
 3: $numTagBits \leftarrow log_2(maxDepth) + log_2(maxTags) + 1$
 4: $maxTagValues \leftarrow getMaxTagValues(xmlSchemaStream)$
 5: $maxAttrValues \leftarrow getMaxAttrs(xmlSchemaStream)$
 6: $maxValues \leftarrow max(maxTagValues, maxAttrValues)$
 7: $numValueBits \leftarrow log_2(maxValues) + 1$
 8: $numEncodingBits \leftarrow max(numTagBits, numValueBits)$
 9: $encodingScheme \leftarrow getEncodingScheme(numEncodingBit)$
10: return $encodingScheme$
```

The Encoding Scheme Algorithm 3 requires a valid XML Emergency schema. It visits all nodes in the entire XML emergency schema in the in-order fashion to calculate the *depth* in line 1 and maximum number of tags *maxTags* in line 2. Then, it computes the number of bits required to encode these tags and the depth in line 3. The algorithm then computes the maximum number of tag values *maxTagValues* in line 4 and the maximum number of attribute names and attribute values *maxAttrValues* in line 5, of which the maximum *maxValues* between the two is computed in line 6. The algorithm then computes the number of bits to encode the values in line 7. Finally, the number of encoding bits for this XML schema is computed in line 8. In line 9, the encoding scheme *encodingScheme* is determined from one of the four values such as 8 bits (one byte), 16 bits (two bytes), 32 bits (four bytes) or 64 bits (eight bytes). The encoding scheme is returned in line 10.

An example application of Algorithm 3 is illustrated in Figure 3.14. One-byte encoding scheme is appropriate to encode an XML document with the depth (say *depth* `x` in line 1) of eight (8) and the maximum number (say *maxTags* `y` in line 2) of tags at each depth is 16. The formula that determines the number of encoding bits (say *numTagBits* z_1 in line 3) from the depth and maximum number of known tags is computed as

$$log_2(x) + log_2(y) + 1 \leq z_1 \tag{3.1}$$

where `x` is the depth, `y` is the maximum number of tags at each depth in each parent tag, and z_1 is the number of encoding bits.

The encoding of a tag value, an attribute name and its value illustrated in Figure 3.15 does not require encoding the depth because they are parts of the current node. Now, to compute the number of encoding bits for the values *numValueBits* z_2 in line 7, we need to calculate the maximum number of tags values *maxTagValues* in line 4, and the maximum number of attributes and attribute values *maxAttrValues* in line 5. The number of encoding bits for the values can be computed as

$$log_2(v) + 1 \leq z_2 \tag{3.2}$$

where *maxValues* `v` in line 6 is the maximum of both number of tag values, and number of combined attributes and attributes values for each tag. The encoding scheme can be computed as

$$z = max(z_1, z_2) \tag{3.3}$$

where *numEncodingBits* `z` is the number of encoding bits required, which is calculated in line 8. For the illustration of this paper, we use the 8-bit enncoding scheme to encode a known tag or a known attribute. From this encoding scheme, the maximum number (`v`) of tag values, and number of attribute names and values is 128 for each tag.

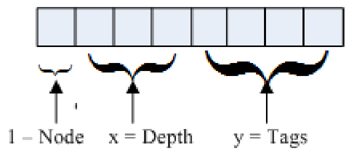


Figure 3.14: Node Depth and Tags

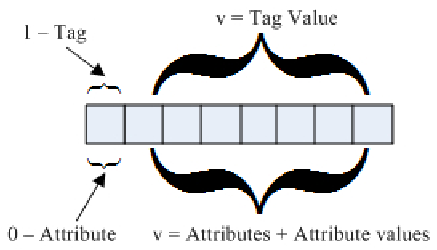


Figure 3.15: Attribute Depth and Tags

For example, as in the Tornado alert message depicted in the listing 3.1, the *category*, *urgency*, *severity*, *certainty* and *spreadable* tags have static values. If we encode each tag separately, it will take a total of ten bytes to encode five tags and five tag values. If we combine these tags together, we can encode them in three bytes, depending on the number of static values for each tag. Figure 3.16 illustrates the combined info tag. The *spreadable* tag is the boolean value so it takes on one bit. However, the category has 12 possible values so it takes four bits.

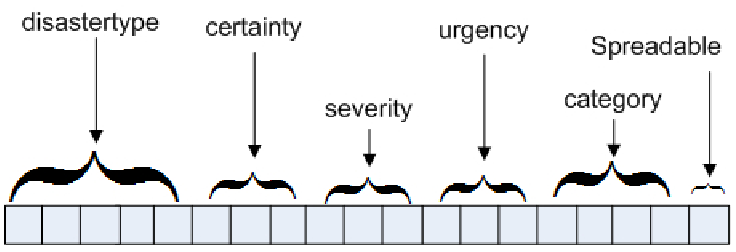


Figure 3.16: Combined Tag

After the XML alert message document has gone through the preprocessing phase, it is ready for encoding. Algorithm 4 shows the CMAS encoding, which we call the Mobile XML Encoding (MXML).

The MXML algorithm requires a valid XML stream as the input parameter and the codepage. In line 1, if combined tags are found in the tag repository, the algorithm will

Algorithm 4 : MXML Encoding Algorithm (Input: XML Stream)

```
Require: $xmlStream \neq null$
1: $xmlStream \leftarrow combinedTags(xmlStream)$
2: $handler \leftarrow new\ MXMLContentHandler()$
3: $reader \leftarrow XMLReaderFactory.createXMLReader()$
4: $reader.setContentHandler(handler)$
5: $xmlSource \leftarrow new\ InputSource(xmlStream)$
6: $reader.parse(xmlSource)$
7: $masterNode \leftarrow handler.getMasterNode()$
8: $outputStream \leftarrow new\ ByteArrayOutputStream()$
9: $masterNode.writeStream(outputStream)$
10: $arrayBytes \leftarrow outputStream.toByteArray()$
11: $outputStream.close()$
12: return $arrayBytes$
```

generate a new XML stream with the combined tags from the orginal XML stream. In Algorithm 4, the MXMLContentHandler `handler` is created in line 2, which implements all the callback functions and loads the codepage based on the XML namespace specified in the XML alert message. In line 3, the XMLReader `reader` is created. The `handler` is registered with the `reader` in line 4. The InputSource `xmlSource` is created from the `xmlStream` in line 5. In the encoding phase, the `MXMLContentHandler` creates the internal tag repository. When the `reader.parse` method [31] is called in line 6 with the `xmlSource` as the parameter, the reader will read each of the XML tags, attributes and their values in the pre-order fashion. The parser will encode the XML document from the root element at depth zero (0) down to its leaves. The encoding process will create the two-byte header. The first byte will represent the encoding scheme consisting of two parts: two-bit (the encoding scheme identifier) and six-bit (the depth of the XML schema), which will represent the unique URN identifier. The encoding method uses the pre-order visit to encode every tag name, tag value, attribute name and attribute value. For every value that the parser encounters, it will look up that value in the tag repository based on the depth level and its parent node for its byte value representation. The value is encoded by inserting its byte representation into a byte array. All the encoding bytes in the array from the child elements will be appended to the parent node. If the value is not statically known, (i.e., the value is not registered in the tag repository) it will be encoded as a text value with the null byte at the beginning and the end of the text. The identifier and description tag values are the examples of the dynamic form. After the encoding phase is complete, a byte array stream containing the encoding bits is returned.

Due to the nature of the Cellular Broadcast Service (CBS), the alert message must be in readable text. Therefore, we must convert the encoding byte array stream into the array of readable characters using the Base64 encoding. On the average, the encoding process adds additional length to the original message length [32] computed using Eq. 3.4:

$$Base64\ Length = (Bytes + 2 - ((Bytes + 2) MOD\ 3))/3 * 4 \tag{3.4}$$

Because the CBS uses an independent Broadcast Controll Channel with its own bandwidth to broadcast an emergency alert and it doesn't compete with other channels for mobile telephony for bandwidth, the alert message can be broadcast as many times as necessary to complete sending the entire alert message if the alert message length is more than 90 characters. In other words, if the alert message after the Base64 encoding is more than 90 characters in length, it can be broken into multiple 90-character messages including the headers. The Emergency Response Application (ERApp) deployed on the smart phone will use the header fields and concatenate these messages back to the original alert message. The header fields are (1) 2-byte alert identification, (2) 1-byte sequence number and (3) 1-byte number of sub messages.

Predefined Encoding Our predefined encoding algorithm also has two phases: (1) preprocessing phase and (2) encoding phase. The preprocessing phase is very similar to the just-in-time method. In addition, the preprocessing phase will define the alert message format and store it in the codepage. ERAlert and ERApp will use this alert message format to encode and decode the alert message respectively.

3.5.4 Experimental Evaluation

In order to evaluate performance, we classify our alert information into two data categories: static and dynamic. Static data fields have predefined values agreed upon at the design time from which emergency operators can select. Category, urgency, severity, certainty, event, etc., are the examples of static fields. Dynamic data field values won't be known until the system is in operation. Event location, expiration time, and description are the examples of dynamic fields. Based on these two data categories, we construct a model for each data category. We will use a tornado example in the previous section to illustrate these models. We pick one of the static data element *category*. <category>Met</category>. Let `s` be the description of the category data field and value and `D(s)` is the length of `s`. Therefore, $D_{normal}(s) = 24\ characters$.

We now construct our CMAS encoding model that describes `s`. In the tag repository for CMAS alert, all the tokens and their values are defined. Each token is represented by an integer value starting by zero (0) and each value associated with a token is also represented by an integer starting with 1. The category field and MET value are represented by 0 and 2 respectively. When encoding the two values 0 and 2, we encode them as byte instead of integer values. Therefore, the CMAS encoding only takes 2 bytes instead of 8 bytes (4 bytes per integer). Therefore, $D_{CMAS}(s) = 2\ bytes$. Let `d(s)` be the minimal length of `s` that describes the category and its value.

For the dynamic data category, we encode these data fields and their values according to their data types. For example, the expiration time can be encoded as a string that represents the timestamp, which takes up to 20 characters in length. However, this data field also can be encoded as a long value of number of milliseconds or an integer value of number of seconds since Jan 1, 1970 at 00:00:00 GMT. Because the number of milliseconds is considered insignificant to this purpose, we encode the expiration time as an integer value which takes only 4 bytes.

We ran these encoding algorithms multiple times on the Dell Latitude E6400 Duo Core Processors and the average of the performance time was calculated. Table 3.1 shows the

Table 3.1: Performance Evaluation Summary

Encoding Algorithm	**Processing Time**	**Encoding Length**	**Base64 Encoding Length**
WBXML	179ms	267 bytes	356 bytes
Prime Power	Infeasible	N/A	N/A
MXML - Predefined	160ms	98 bytes	132 bytes
MXML	158ms	118 bytes	160 bytes

result of encoding a 565-character tornado message from all the encoding algorithms mentioned above. It shows that the encoding that gives the shortest length in readable text is the MXML - Predefined. It gives the shortest description of the tornado local alert that can be broadcasted over the air interface. The Prime Power is declared to be infeasible at the current state of the machine power because when we ran the test, the machine CPU was utilized to 100% and stayed at 100% for hours. All the encoding techniques give the initial binary string that must be converted to readable text using Base 64 encoding. Because CMAS only takes 90 characters per broadcast, this tornado alert will have to be broadcast twice. ERApp will accept the broadcast messages, concatenate them, and decode the message.

3.6 Related Works

In this section, we will discuss one of our recent publications in the emergency arena, as well as other related works. We also briefly discuss the relevancy of these works with respect to our work presented in this paper.

In April 2007, the Integrated Public Alert and Warning System (IPAWS) [53] was established by FEMA under the Executive Order 13407 signed by President George W. Bush on June 26th, 2006. The IPAWS mission is to ameliorate the public safety at all levels of government by providing integrated and interoperable services to communicate timely alerts and warnings effectively in the conservation of life and property. The IPAWS project focused on two communication strategies (e.g., radio broadcast and mobile broadcast). IPAWS also promote the standard protocol to communicate alerts and warnings across all levels of government emergency systems. In addition, one of the IPAWS projects focused on modernizing and expanding the legacy Emergency Alert System (EAS) to take advantage of cutting-edge technologies. The CMAS is one of major projects under the IPAWS program, focusing on mobile devices.

Many universities and organizations such as LSU [54] and GMU [55] have implemented alert mechanisms that require cellphone and/or email subscriptions from the users. These systems leverage the existing commercial telecomm and service provider infrastructures as the communication protocols to deliver emergency alerts to the users. During an emergency, these systems will place an extreme heavy load onto the networks by sending massive emails or text messages, which can in turn create a denial of service attack on the networks [24]. As results, these alert messages are not guaranteed to be delivered to the users' devices in a timely manner.

Chapter 4: The Availability of Emergency Responders

4.1 Requirements

In this section, we specify some requirements and objectives for availability that organizations must implement in their processes and operations in order to ensure the availability of responders. A requirement contains the word "shall" and is identified by the letters "**AR**". *Objective* is a feature or function that is desirable and may be required for emergency response. An Objective contains the words "it is desirable" and is identified by the letters "**AO**".

AR#1: An emergency handler shall be available for each service at all times.

AR#2: All handler roles shall be scheduled at all times.

AR#3: An on-duty handler shall have a way to be replaced in case of a personal need without violating AR1 & AR2

AO#1: It is desirable that an organization provide a predefined number of individuals as backups (at least three backups as discussed in this paper) for each role at any given time.

We illustrate these requirements with a real-life emergency incident that occurred in a neighborhood [13]. The incident occurred when excavators struck a gas line while installing sound-proof barriers. An eight-inch break ruptured a major gas line and left the neighborhood without gas for days. Thirty people in the neighborhood were evacuated to a safe area while the utility crew were containing the gas leak. This emergency illustrates the importance of the availability of the utility crew who responded to the incident and were authorized and able to shut down the gas line. Any delay in terms of getting the utility crew on the phone can potentially cause a disaster that may result in the loss of life and property. Apparently, the utility crew on duty in the area were reached minutes after the gas odor was detected by the construction workers at the site.

Now we apply some of the "what-if" use cases to this incident and show how our requirements will fill in the gaps. First is, what if the utility crew is unreachable at the time of the emergency call? We looked through the emergency management documents [34] and found that there is no requirement for organizations to provide an emergency handler for their services at any time. Our AR#1 provides such a requirement. Organizations then map the services that they can provide to designated roles that are assigned to individual employees. From here onwards, we will use the term *role(s)* to mean emergency service provider roles. Second is, what if an authorized person would like to schedule the crew on a regular basis so that the emergency response service is available 24/7 without an interruption? Our AR#2 provides such a requirement for organizations to implement so that designated schedulers have the capability to do this. Third is, what if the assigned utility crew for handling emergency calls during this period anticipates some personal events that prevent them from fulfilling their duties? Currently, there is no automated way to find a replacement for them. Our AR#3 provides such a requirement.

4.2 Availability Model

In this section, we discuss the availability model and delay calculations of the architectural design.

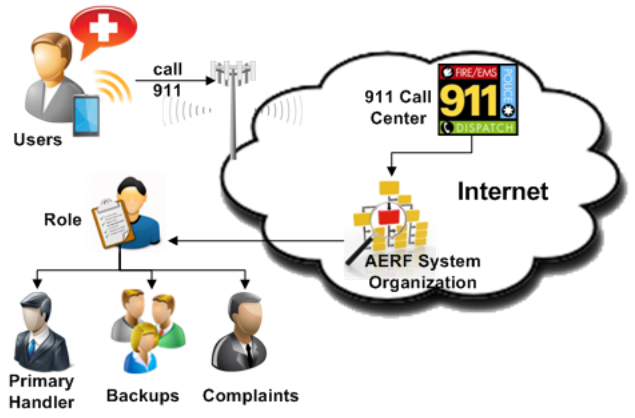


Figure 4.1: AERF Framework Model

4.2.1 Availability of Emergency Responders

Figure 4.1 shows an end-to-end AERF framework model from the emergency call to the emergency responder handling it. A user that requires emergency assistance calls the 911 hot-line in step 1. The 911 operator at the calling center triages and determines the role of the appropriate responder in step 2. The call is then forwarded to the person playing that role in step 3.

In order to satisfy AR#3 and AO#1, the system itself needs to take the following steps. At the organization switch board, an automatic query is made to the AERF system to retrieve the list of handlers, which contains the Primary Handler, the Backup list (including First, Second, and Third Backups, which we call the *Calling Ladder*), and the complaints arising out of the role handler's faults in step 4. The calling ladder is then returned to the organization switchboard, which routes the call to the appropriate Primary Handler in step 5, who may triage the call again. If the Primary Handler doesn't pick up the call, the switchboard will route the call to the next person in the Calling Ladder in step 6 to the First Backup. It will repeat this process until someone in the Calling Ladder responds to the emergency call. If none in the Calling Ladder picks up, the call will be routed to the complaints which is the Department Chair or the deputy to submit a complaint in step 7. This is shown in Figure 4.2.

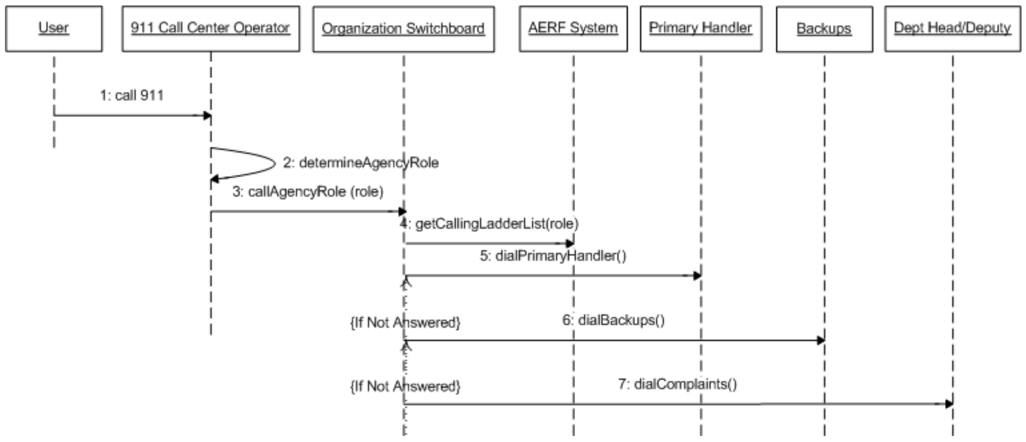


Figure 4.2: AERF Sequence Diagram

Our AERF framework model ensures that there is always a Primary Handler available at all times for all the services that are mapped to the roles in the organization. The AERF framework allows all personnel reporting their availability status and depending on where they are in the Calling Ladder and the effected work shift, the framework will back-fill other personnel to play the same role, thereby addressing the AR#3. For example, if Mr. A is scheduled to work today from 8AM to 4PM shift and he is not able to commute to work, he will use the ERApp on his smart phone device to send a timeout request. The AERF system at his organization will examine his timeout request and back-fill other personnel who are designated as backups for his shift. In order to do so, the AERF system will send a request to each of the personnel in the backup list. Once someone (say Ms. B) agrees to fill the role temporarily, Mr. A will be released.

4.2.2 The Mathematical Model

Now we develop a mathematical model to represent the time line from the initial call to the completion of all the actions being executed to resolve an emergency situation. Let t be the total time of the coordinating efforts that is required to resolve an emergency and t can be defined as follows:

$$t = t_r + \sum_{i=1}^{n} t_s + \sum_{j=1}^{m} t_a \tag{4.1}$$

where t_r is the requesting time, t_s the servicing time, n the number of services, t_a the action time, and m the number of actions being executed. The requesting time can be further defined as the duration of time from the moment that the caller makes an emergency call to request emergency assistance to the moment that a 911 operator hangs up the call. The servicing time is defined as time difference from the moment that the 911 operator requests some assistance on behalf of the caller by making subsequent calls to emergency handlers and begins the dialog with them to the moment that a decision is made and the emergency plans are put in place for action. Lastly, the action time is the total time required to execute an emergency plan from start to completion. In this paper, we are not addressing how to improve the requesting time and the action time but focusing on reducing the servicing time t_s, which can be further defined in more details as follows:

$$t_s = \sum_{k=1}^{p} (t_l^k + t_d^k + t_w^k + t_t^k) \tag{4.2}$$

$$where\ t_t^k > 0\ if\ only\ if\ \forall k' > k,\ t_l^{k'}, t_d^{k'}, t_w^{k'}, t_t^{k'} = 0$$

$$and\ \forall k' < k,\ t_w^{k'} = t_w^{max}$$

where p is number of backup personnel, t_l the lookup time, t_d the dialing time, t_w the waiting time, and t_t the talk time.

The probability of the primary handler without backups not picking up the emergency call can be calculated as follows:

$$P'_{current} = (1 - P_{ph}) \tag{4.3}$$

The probability of the primary handler and k^{th} backup doesn't pick up the emergency call can be calculated as follows:

$$P'_{AERF} = (1 - P_{ph})(\prod_{i=1}^{k}(1 - P_i)) < P'_{current} \tag{4.4}$$

We conclude that the probability of the primary handler not answering the emergency call is greater in the current system than in a system using the AERF framework. Thus,

$$P'_{current} > P'_{AERF} \tag{4.5}$$

4.3 AERF Architectural Design

This section describes the AERF framework. Figure 4.3 shows the interactions between various components within the AERF framework including user, organization, and third-party spaces.

4.3.1 The User Space

The User space defines system services available to users. Each user is equipped with a mobile device with an Emergency Response Application (ERApp). We provide some features to enhance the availability of emergency responders such as registration and creating the timeout event in order to fulfill the requirements.

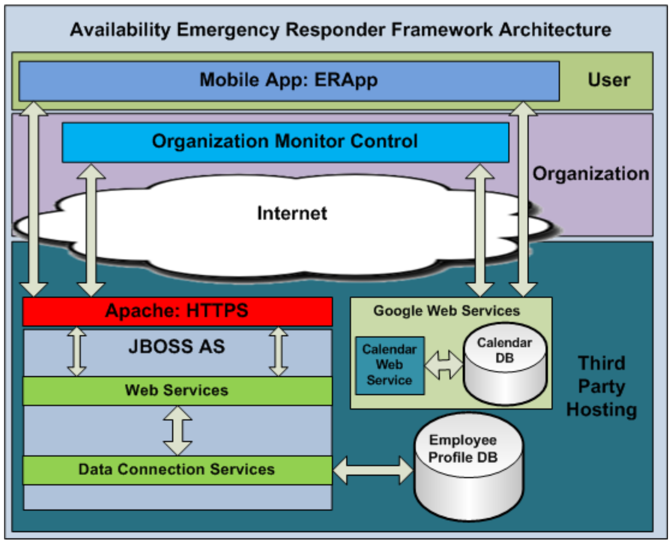


Figure 4.3: AERF Architecture

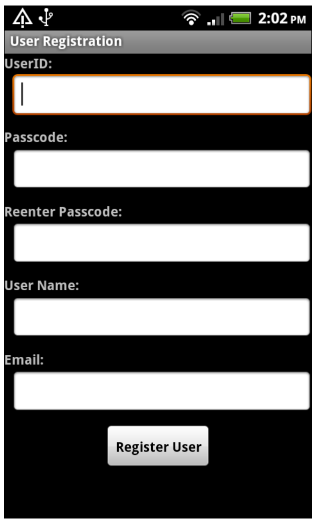
User Registration

The ERApp allows a user to register with his/her organization. Figure 4.4(a) depicts the registration screen on the mobile device, requesting the user to enter the id, pass code, name, and the email address. Upon successful registration, he/she is able to use all other functions in the ERApp. In addition, the ERApp will integrate with user's existing email calendar. This is required to creating the timeout event feature, which is described in the next section.

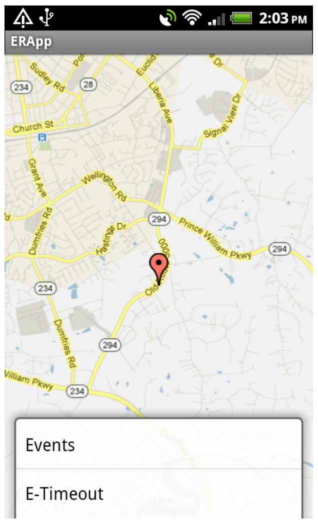
Timeout Event

One of the important functions is the ability to create calendar events or timeout events as shown in Figure 4.4(b). A calendar event can be work related, such as a meeting, an appointment, etc. A timeout event informs the organization that the user is not available during this timeout period. A timeout event can be an unplanned personal event. When creating a timeout event, the user enters the timeout period and the reason as shown in

Figure 4.5(a). There is a list of default justifications that are used on a regular basis as shown in Figure 4.5(b) or the user can create his/her own incident. The user will receive an acknowledgement either via email and/or text message from the organization as shown in Figure 4.6(a). At this time, the AERF framework will try to find a replacement by sending a work request to the backup personnel as depicted in Figure 4.7(a). Once a backup person accepts the work request, he/she will receive the work confirmation as shown in Figure 4.7(b). If a replacement is found, the timeout requester will receive the confirmation as shown in Figure 4.6(b). If none of the backups responds in a timely manner, the AERF system will submit a complaint to the department chair or the deputy to find a replacement. If there is no action taken by the department chair and/or the deputy within a configurable time frame, the system administrator will follow up with them and resolve the situation. If there is an emergency during this time frame, the emergency call will be forwarded to the department chair and the deputy who will serve as the role backup by default. Figure 4.8 is the sequence diagram describing the message flow that manages a timeout event.

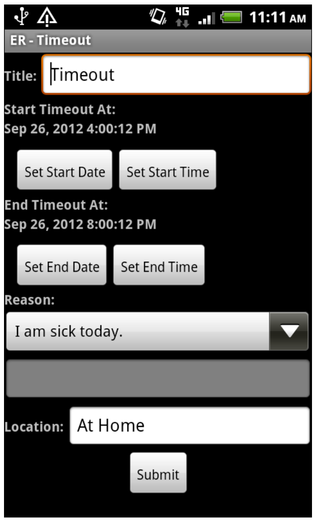


(a) ERApp Registration



(b) ERApp Events

Figure 4.4: ERApp Features



11:11 AM

ER - Timeout

Title: Timeout

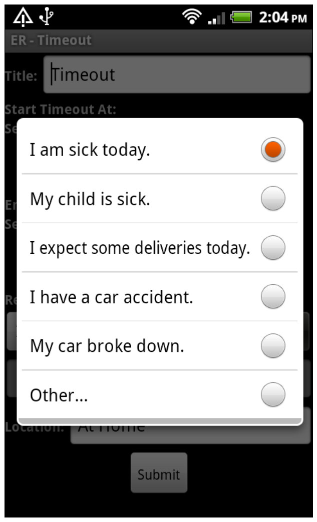
Start Timeout At:
Sep 26, 2012 4:00:12 PM

End Timeout At:
Sep 26, 2012 8:00:12 PM

Reason:
I am sick today.

Location: At Home

(a) Event Timeout



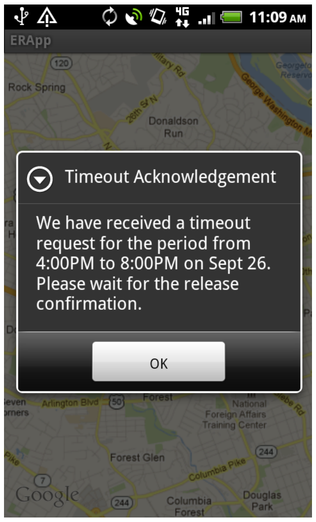
2:04 PM

ER - Timeout

I am sick today.	●
My child is sick.	○
I expect some deliveries today.	○
I have a car accident.	○
My car broke down.	○
Other...	○

(b) Timeout Reasons

Figure 4.5: Event Timeout



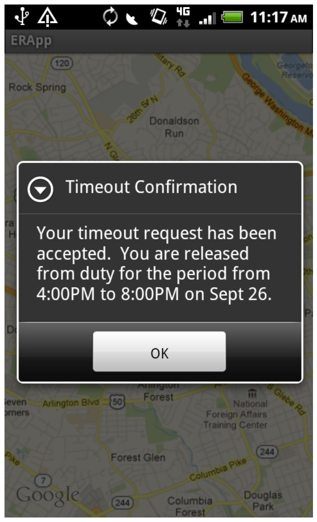
11:09 AM

ERApp

Timeout Acknowledgement

We have received a timeout request for the period from 4:00PM to 8:00PM on Sept 26. Please wait for the release confirmation.

(a) ERApp Timeout Acknowledgement



11:17 AM

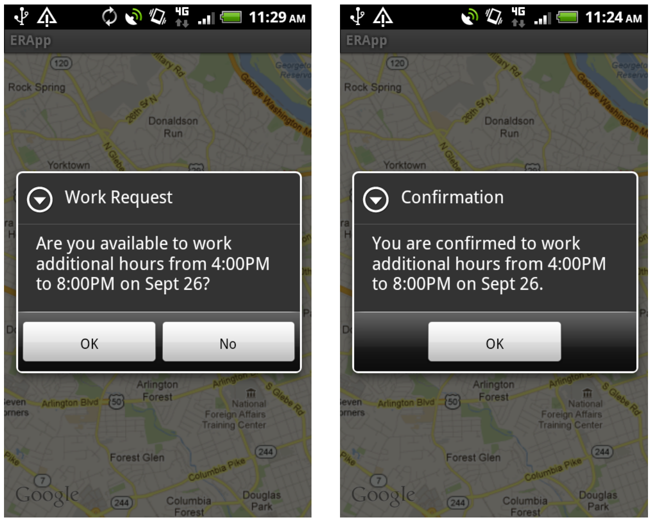
ERApp

Timeout Confirmation

Your timeout request has been accepted. You are released from duty for the period from 4:00PM to 8:00PM on Sept 26.

(b) ERApp Timeout Confirmation

Figure 4.6: ERApp Timeout Responses


Work Request

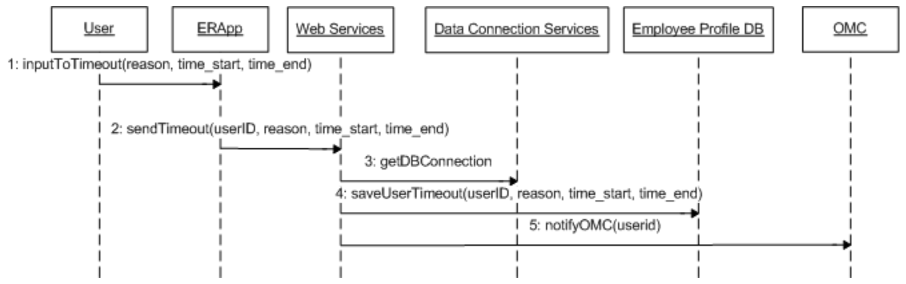
Are you available to work additional hours from 4:00PM to 8:00PM on Sept 26?

Confirmation

You are confirmed to work additional hours from 4:00PM to 8:00PM on Sept 26.

(a) ERApp Work Request (b) ERApp Work Confirmation

Figure 4.7: ERApp Work Responses



User
ERApp
Web Services
Data Connection Services
Employee Profile DB
OMC
1: inputToTimeout(reason, time_start, time_end)
2: sendTimeout(userID, reason, time_start, time_end)
3: getDBConnection
4: saveUserTimeout(userID, reason, time_start, time_end)
5: notifyOMC(userid)

Figure 4.8: Timeout Sequence Diagram

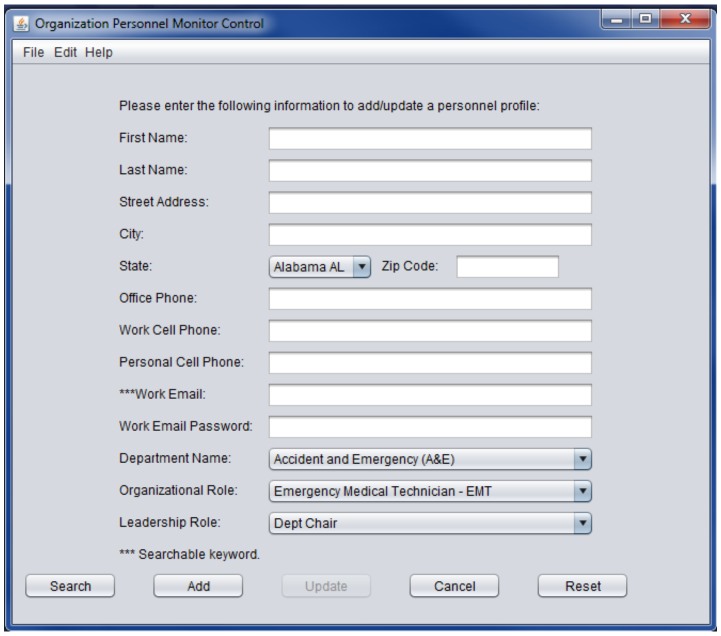
4.3.2 The Organizational Space

The Organizational Space supports use cases that belongs to an organization. The Organization Monitor Control (OMC) is designed to support all the use cases for organization administrators. Depending on the leadership role of the personnel, certain use cases can be available when they log into the OMC. In addition, system administrators who monitor the

OMC at the organization level have a special privilege to view all the Primary Handlers for all the roles in the organization. Their primary job is to monitor the status of all Primary Handlers for the current work shift. All the use cases of the OMC are described in the subsections below.

Updating Personnel Profile

The updating personnel function is reserved for all personnel regardless of their roles. But before the personnel can update their profiles, they must create an account with the OMC in a secure way that creates the PKI for them.



Please enter the following information to add/update a personnel profile:

First Name:	
Last Name:	
Street Address:	
City:	
State:	Alabama AL
Zip Code:	
Office Phone:	
Work Cell Phone:	
Personal Cell Phone:	
***Work Email:	
Work Email Password:	
Department Name:	Accident and Emergency (A&E)
Organizational Role:	Emergency Medical Technician - EMT
Leadership Role:	Dept Chair

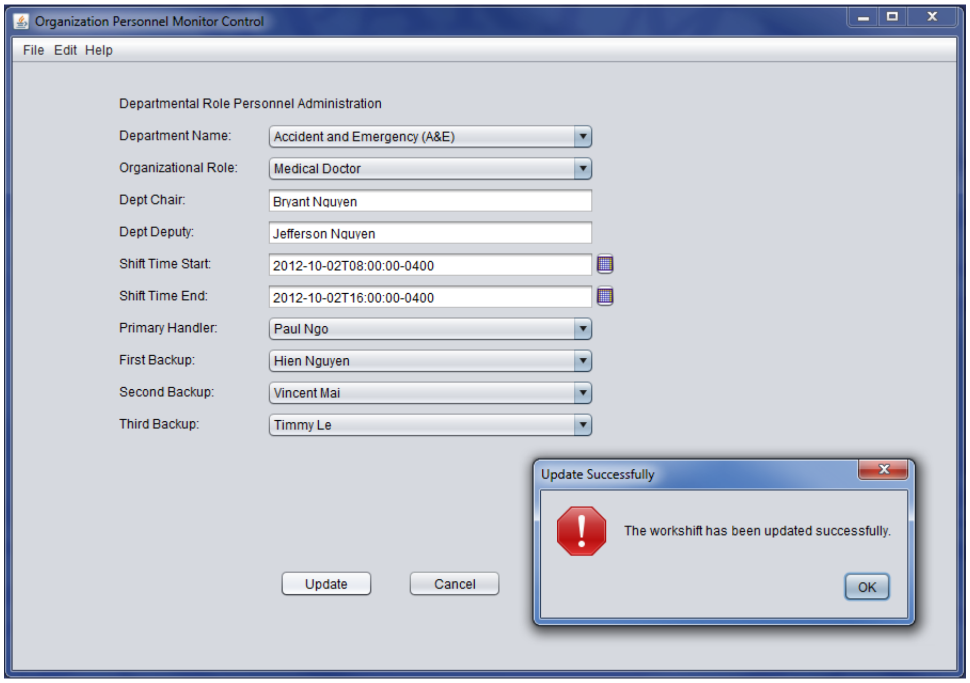
*** Searchable keyword.

Figure 4.9: OMC Personnel Profile Screen

Figure 4.9 shows the personnel profile screen where the personnel can update their information. The OMC will encrypt the personnel profile data using PKI before sending an update message to the Web Services residing on the database in the Third Party Space. Therefore, the Privacy and Personal Information (PPI) of the personnel is highly protected and secured.

Creating/Updating Work Shifts

These use cases are reserved for the Department Chair and the deputy, who can assign which personnel to serve as the Primary Handler or in the Backup list for a certain work shift. Figure 4.10 shows the work shift screen where the Department Chair or deputy can create or update a work shift.



Departmental Role Personnel Administration

Department Name:	Accident and Emergency (A&E)
Organizational Role:	Medical Doctor
Dept Chair:	Bryant Nguyen
Dept Deputy:	Jefferson Nguyen
Shift Time Start:	2012-10-02T08:00:00-0400
Shift Time End:	2012-10-02T16:00:00-0400
Primary Handler:	Paul Ngo
First Backup:	Hien Nguyen
Second Backup:	Vincent Mai
Third Backup:	Timmy Le

The workshift has been updated successfully.

Figure 4.10: OMC Work Shift Screen

Figure 4.11 shows the sequence diagram for creating a work shift and a work shift policy and hand it over to the Policy Enforcement Point (PEP). The PEP is the entity that protects resources (in this case, the work shift policy). The PEP forms a request based on attributes such as subjects, actions, resources and other relevant information [10]. In step 1, the department chair or deputy will use the work shift screen to enter and submit the

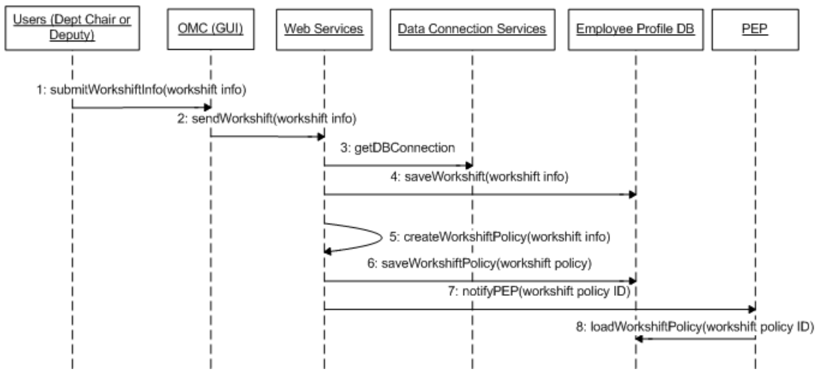

Users (Dept Chair or Deputy)
OMC (GUI)
Web Services
Data Connection Services
Employee Profile DB
PEP
1: submitWorkshiftInfo(workshift info)
2: sendWorkshift(workshift info)
3: getDBConnection
4: saveWorkshift(workshift info)
5: createWorkshiftPolicy(workshift info)
6: saveWorkshiftPolicy(workshift policy)
7: notifyPEP(workshift policy ID)
8: loadWorkshiftPolicy(workshift policy ID)

Figure 4.11: Work Shift Sequence Diagram

work shift information to the OMC. The OMC will send the work shift information to the Web Services in step 2. The Web Service retrieves the database connection in step 3 and saves the work shift information in the database in step 4. The Web Services will create the work shift policy in step 5 and save it in the Employee Profile database in step 6. The Web Services then notifies the PEP in step 7 and the PEP will load and enforce the policy in step 8.

Replacing Personnel

Each role in the organization must have a roster of personnel with one listed as a Primary Handler, First Backup, Second Backup, and Third Backup for a period of time called a *Work Shift*. For a small organization, the framework does not require all of these backups to be filled for the work shift. However, it is in the organization's best interest to meet the objective **AO#1**. The idea is that if the framework receives an availability status notification from anyone in the Primary Handler (or in any other positions in the Calling Ladder), the framework will send the replacement request to the next person in the Calling Ladder. The framework will wait for the acknowledgement to be received within a specified time frame. Once the framework receives a confirmation, it will move him/her up to fill up that empty spot. If not, the framework will send the replacement request to the next person and repeat the process. If there aren't available people in the Calling Ladder or none has confirmed the replacement, the framework will send an urgent notification to the scheduler for further actions and send an alarm to the Organizational Monitor Control (OMC) so that the administrator can take further actions to ensure that the Primary Handler slot for that role has been filled.

Improper Modification

To prevent illegal policy updates, we allow the database access only for authorized personnel and encrypt the policy at rest.

Delegating Leadership

In case the department chair and the deputy are not able to perform the update to the work shift policy, they can delegate the update of this role work shift policy to one of their staff members. The framework will create a temporary permission and grant it to the delegated staff member. When he/she logs into the OMC, the framework will query the PEP for what permissions this staff member is allowed. After the delegation period ends, the framework will revoke this permission.

4.3.3 Third Party Space

This space hosts all the database transactions such as personnel updates, work shifts updates, policy updates, etc. All of these transaction must be protected via encrypted communications. We define a clear interface so that the Organizational Space can use to communicate with the database in the Third Party Space. We host the web server, the application server, and the physical database in the Cloud. Even though we have a number of choices for the web, application, and database technologies, we choose Apache, JBOSS, and MySQL for the web, application, and database servers respectively in our prototype.

4.4 AERF Implementation

This section describes our implementation of the AERF framework.

4.4.1 Security Implementation

In this section, we discuss the security implementation to ensure that the communications between different spaces are protected and secured.

Figure 4.3 depicts the AERF framework architecture in which we explicitly define the space boundaries between architectural entities. When an entity in the User space communicates with an entity in the Organization space, data has to leave its space and enter the Internet before entering the destination space. We use an encryption mechanism to secure all users' information. Due to the amount of data being transfered in the communication medium, the Public Key Infrastructure (PKI) [35] method is used for the encryption of users' PPI data.

Each entity creates its own public and private key pairs. The public key can be openly distributed to other entities. In our framework implementation, we distribute the public key via secure (HTTPS) web services.

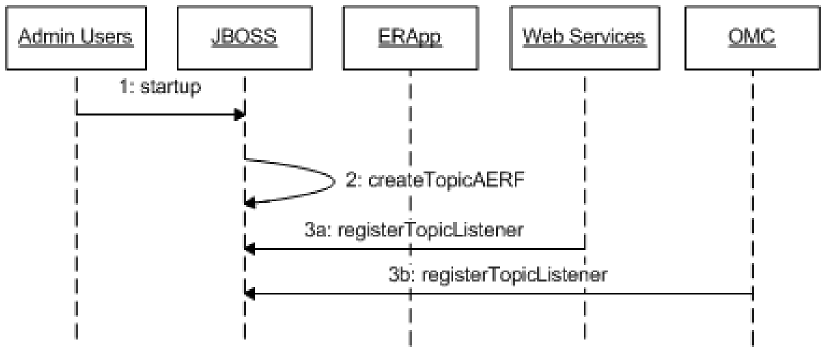


Figure 4.12: ERApp & OMCs JMS Sequence Diagram

The ERApp in the User space and OMC in the Organizational space never communicate with one another directly, but when the user submits a *timeout* event to the server via Secure Web Services, the server converts the *timeout* event to the message and communicates it to the OMC via the Java Message Service (JMS) [36]. The server can send a JMS message to multiple subscribers who are listening to a topic. We set up a JMS topic named AERF that is registered to JBOSS AS at the startup. A JMS topic is a subscription base mechanism that allows multiple OMC instances subscript to a topic and receive messages when there is something new to be broadcast. Figure 4.12 shows the sequence diagram of communication events that are involved with JMS from the ERApp to the OMCs. In step 1, the administrator starts up the JBOSS server. During the startup, JBOSS creates the AERF topic. In step 3a and 3b, the web services and OMC register as listeners for that topic.

4.4.2 Role-Based Access Control (RBAC) Implementation

This subsection describes how we use the RBAC Profile for XACML [40] with some enhancements [37–39]. In the OMC, the leadership personnel such as the scheduler has permissions to create new work shifts for an organizational role in their departments and assign personnel to the roles of Primary Handler, First Backup, Second Backup and Third Backup. Once the schedulers commit on the work shift, a new policy is created, encrypted, and written to the policy repository resided on the organizational server. For demonstration purposes, we use AES 256 to encrypt the policy file and XACML to write the organizational policies. We establish three types of policies: personal policy, organizational policy, and work shift policy. Below is the description of how each type of policies being used and for what purpose.

Monitoring the Primary Handler's Availability Status

Each personnel has his/her own policy that contains a set of rules and conditions that must be satisfied before any new work shift can be assigned to the personnel in questions.

Replacement of the Primary Handler

Before discussing the details of the algorithm to replace the Primary Handler, we need to discuss the *createReqCtx* algorithm.

Algorithm 5 : OMC **creatReqCtx** Algorithm (Input: IPersonnel pp, IValidator validator)

```
Require: $pp \neq null$
Require: $validator \neq null$
 1: $requestCtx \leftarrow createRequestCtx()$
 2: $sT \leftarrow new\ SubjectType()$
 3: $listAttrs \leftarrow validator.getAttrs()$
 4: $iterA \leftarrow listAttrs.iterator()$
 5: while $iterA.hasNext()$ do
 6:     $attr \leftarrow iterA.next()$
 7:     $sn \leftarrow attr.getAN()$
 8:     $sv \leftarrow pp.getAV(sn)$
 9:     $st \leftarrow attr.getAT()$
10:     $sT.getAttrs().add(createSubAT(sn, sv, st))$
11: end while
12: $rT \leftarrow new\ ResourceType()$
13: $rn \leftarrow validator.getResName()$
14: $rv \leftarrow validator.getResValue()$
15: $rt \leftarrow validator.getResType()$
16: $rT.getAttrs().add(createResAT(rn, rv, rt))$
17: $aT \leftarrow new\ ActionType()$
18: $an \leftarrow validator.getActName()$
19: $av \leftarrow validator.getActValue()$
20: $at \leftarrow validator.getActType()$
21: $aT.getAttrs().add(createActAT(an, av, at))$
22: $reqT \leftarrow new\ RequestType()$
23: $reqT.getSubject().add(sT)$
24: $reqT.getResource().add(rT)$
25: $reqT.setAction(aT)$
26: $requestCtx.setRequest(reqT)$
27: return $requestCtx$
```

Algorithm 5 is the *createReqCtx* pseudo code, that creates the request context in order to evaluate against the organization policies to determine if a person can be qualified to replace another person. The algorithm requires two parameters: *pp* and *validator*, which implement the *IPersonnel* and *IValidator* interfaces respectively. The *IPersonnel* interface provides a set of methods that allows read and write access to the personnel properties such as number of working hours, shift start time, shift end time, leadership role, organization role, etc.. The *IValidator* interface provides a set of methods that allows read-only access to the organization policy. The organization provides the implementations for these interfaces in the AERF framework. These two parameters must be provided to algorithm 5.

On line 1, the default request context *requestCtx* is created by calling the method *createRequestCtx* that comes with XACML package. On line 2, the subject type sT is created. The list of subject attributes is retrieved from the *validator* on line 3. The *validator* keeps all the subject attributes such as leadership role, total working hour, etc. The attribute iterator *iterA* is created on line 4. From lines 5 through 11, the algorithm retrieves each attribute *attr* from the *iterA* on line 6, the attribute subject name *sn* on line 7, and the corresponding subject attribute value *sv* from the personnel *pp* based on the subject attribute name from the *attr* on line 8. On line 9, the type of the subject attribute *st* is retrieved from *attr*. On line 10, the new subject attribute type is created and added into the list of the subject type sT attributes.

On line 12, a resource type rT is created. The algorithm then retrieves a resource name *rn personal-profile*, its value *rv resource-id*, and its type *rt string* from the *validator* on lines 13, 14, and 15 respectively. The new resource attribute type is created and added into the list of the resource type rT attributes on line 16. The action type aT is created on line 17. The new action attribute type is created on line 17. The algorithm then retrieves an action name *an*, its value *av*, and its type *at* from the *validator* on lines 18, 19, and 20 respectively. A new action attribute type is created and added into the list of the action type aT attributes on line 21. The new request type *reqT* is created on line 22. The subject type sT is then added to the list of subject types for the request type on line 23. The resource type rT is then added to the list of resource types for the request type on line 24. On line 25, the action aT is then set to the request type *reqT*, which is set into the request context *requestCtx* on line 26. Finally, the request context *requestCtx* is returned on line 27.

Algorithm 6 is the OMC *replacePH* pseudo code. It requires two input parameters:

Algorithm 6 replacePH (Input: ITOReq reqTO, IController controller)

```
Require: $reqTO \neq null$
Require: $controller \neq null$
 1: $isDone \leftarrow false$
 2: $plcRep \leftarrow PolicyRep.getInstance()$
 3: $validator \leftarrow OrgPValidator.getInstance()$
 4: $st \leftarrow reqTO.getStartTime()$
 5: $et \leftarrow reqTO.getEndTime()$
 6: $role \leftarrow reqTO.getOrgRole()$
 7: $rwsPlc \leftarrow plcRep.getRWSPlc(role,\ st,\ et)$
 8: $listPP \leftarrow rwsPlc.getPPinCallingLadder()$
 9: $listPP \leftarrow controller.getPPRole(role,\ listPP)$
10: $listPP.append(controller.getDefBackup(role))$
11: $iterPP \leftarrow listPP.iterator()$
12: while $iterPP.hasNext()$ do
13:    $pp \leftarrow iterPP.next()$
14:    $ppPlc \leftarrow pp.getPPPlc()$
15:    $requestCtx \leftarrow createReqCtx(pp,\ validator)$
16:    if $validator.checkOrgPlc(requestCtx)$ then
17:       $controller.sendConReq(pp)$
18:       if $controller.waitConResp(pp)$ then
19:          if $validator.acqLock(rwsPlc, ppPlc)$ then
20:             $plcRep.updateRWSPlc(pp,\ rwsPlc)$
21:             $validator.relLock(rwsPlc)$
22:             $plcRep.updatePPPlc(pp,\ ppPlc)$
23:             $validator.relLock(ppPlc)$
24:             $controller.sendNewWS(pp)$
25:             $controller.sendCon(reqTO)$
26:          else
27:             $controller.sendInc(reqTO)$
28:          end if
29:          $isDone \leftarrow true$
30:          $break$
31:       end if
32:    end if
33: end while
34: if $!isDone$ then
35:    $controller.sendDeny(reqTO)$
36: end if
37: $listLS \leftarrow controller.getLS(role)$
38: $iterLS \leftarrow listLS.iterator()$
39: while $iterLS.hasNext()$ do
40:    $pp \leftarrow iterLS.next()$
41:    if $isDone$ then
42:       $controller.sendNewWSPlc(pp, rwsPlc)$
43:    else
44:       $controller.sendActReq(pp, rwsPlc)$
45:    end if
46: end while
```

reqTO and *controller*. The *reqTO* is the request timeout which is created when the OMC receives the timeout message from the Primary Handler. This request timout implements the *ITOReq* interface. The *controller* has all the communication APIs with the organization database. It also has the ability to communicate with personnel involved in the replacement process via ERApp through text-messaging, emailing and requested response exchanges with them.

The algorithm initializes *isDone* flag to be false on line 1. If this flag is set to *true*, the *replacePH* has found a replacement for the Primary Handler and *false* if it has not. On line 2, the algorithm acquires an instance of the policy repository and assigns it to *plcRep*. One line 3, the *validator* is assigned to the instance of the organization policy validator, which implements the IValidator interface. The start time *st*, end time *et*, and the organizational role *role* are retrieved from the *reqTO* on lines 4-6 respectively. The role work shift policy is retrieved from the policy repository *plcRep* and is assigned to *rwsPlc* on line 7.

It will retrieve the list of personnel *listPP* in the Calling Ladder from the role work shift policy on line 8. The framework will retrieve an additional list of personnel from the organizational database that are listed under this role on line 9 and append this list to *listPP*. The framework retrieves and appends the default backup personnel to the list on line 10. The framework then retrieves the *iterPP* iterator from the *listPP* on line 11 and asks the iterator if it still has the next personnel profile to be examined on line 12.

The framework then goes through each personnel record *pp* on line 13 and retrieves the personnel profile policy *ppPlc* from the personnel record on line 14. The framework then creates the request context on line 15 in order to validate against the organization policy on line 16. If the validator comes back with a *Deny*, the framework will move to the next personnel on line 13 again.

However, if the validator comes back with a *Permit* on line 16, the controller will send a confirmation request to the personnel on line 17 and wait for a confirmation response from the personnel on line 18. If the positive response comes back within a configurable time frame, the framework will update the role work shift policy to replace the timeout requester with this personnel as the Primary Handler on line 20. In order to update the role work shift policy and the personnel profile policy, the framework must acquire both write locks on the role work shift policy and the personnel profile policy to gain full access and update on line 19. The framework will continue only if the acquisition of both policies' locks is successful. The validator releases the lock on the role work shift policy on line 21. The framework then updates the personnel profile policy to reflect the replacement on line 22 and releases its lock on line 23. The framework will notify the personnel that he/she is officially the Primary Handler for the work shift on line 24. In addition, the framework will send the confirmation to the timeout requester to indicate that the timeout request has been processed successfully and the requester is released from the Primary Handler role on line 25. The framework will also set the *isDone* flag to be *true* on line 29 to indicate that the framework has found a replacement for the Primary Handler. If the validator is not able to acquire both locks on the role work shift policy and the personnel profile policy, the framework informs the timeout requester that it can not complete finding a replacement because some other OMC instances possess the lock on the role work shift policy on line 27.

If *isDone* is still *false* on line 34, the framework will send the deny to the timeout requester to indicate that it can not find a replacement at this time due to the unavailability of the personnel on line 35.

At this point, the framework still has some bookkeeping tasks. It receives the list of leadership personnel *listLS* from the organizational database on line 37 and creates the iterator *iterLS* on line 38. The framework then goes through each of the personnel in the leadership. If the framework has found a replacement (*isDone=true*), it has to notify the leadership personnel that the role work shift policy has been updated with the new Primary Handler on line 42. Otherwise, the framework will send an obligational action request to the leadership for their intervention to resolve the issue on line 44.

4.4.3 Policy Implementation

Our framework provides the flexibility to change and implement new organizational policies. There are two ways that an organization can achieve this: expressing policies in XACML document or formulating the policies in the code. Before we show how this done, we illustrate a simple policy of finding a replacement for an employee who has indicated a timeout for the current work shift. This policy will address two conditions which must be satisfied during the evaluation. The first condition is that the employee must have the leadership role of either Employee, department chair, deputy, Vice President, or President. The second condition is that the employee must not work over 12 hours within the 24-hour period. Below is how this policy can be expressed in XACML and in the code.

XML Policy

Figure 4.13 shows the simple organization policy in XML.

In this simple policy, we specify *deny-overrides* under the policy attribute name *RuleCombiningAlgId* on line 5 to dictate how we want the XACML engine to evaluate all the rules in this policy. If any rule returns *Deny*, the XACML engine stops evaluating further and returns *Deny* to indicate that the policy denies the action on the resource. We have only one *Target* tag in which we have one *Resource* called *personnel-profile* and the type of this resource is a string. To ensure that the request indicates the correct resource, a *ResourceMatch* tag on line 10 has a match id of *string-equal* to validate the requested resource value matches with the resource value in this policy. We also have one *Action* called *access* from lines 18 to 24 and the type of this action is also a string on line 20. Similar to *ResourceMatch*, an *ActionMatch* from lines 19 to 23 is to ensure that the action value from the request must be validated against the action value in the policy.

Furthermore, we have three rules to be evaluated. The first rule is called *rule_matching_role* from lines 27 to 43 and if the *Condition* from lines 28 to 42 in this rule is satisfied, the *Effect*, which is *Deny*, is returned. In this rule, we have three nested *Apply* functions on lines 29, 30, and 33. The outermost *Apply* function on line 29 has a function ID of *not*. The

```
 1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
 2 <Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
 3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 4   xsi:schemaLocation="http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-policy-schema-os.xsd"
 5   RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides"
 6   Version="1.0" PolicyId="INOVAFairfaxHospitalPolicy">
 7   <Target>
 8     <Resources>
 9       <Resource>
10          <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
11            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">personnel-profile</AttributeValue>
12            <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
13                                          DataType="http://www.w3.org/2001/XMLSchema#string"/>
14          </ResourceMatch>
15       </Resource>
16     </Resources>
17     <Actions>
18       <Action>
19          <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
20            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">access</AttributeValue>
21            <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
22                                        DataType="http://www.w3.org/2001/XMLSchema#string"/>
23          </ActionMatch>
24        </Action>
25    </Actions>
26  </Target>
27  <Rule Effect="Deny" RuleId="rule_matching_role">
28    <Condition>
29      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:not">
30        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-at-least-one-member-of">
31          <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
32                                      AttributeId="leadershipRole" />
33          <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
34            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Employee</AttributeValue>
35            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Dept Chair</AttributeValue>
36            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Dept Deputy</AttributeValue>
37            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">President</AttributeValue>
38            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Vice President</AttributeValue>
39          </Apply>
40        </Apply>
41      </Apply>
42    </Condition>
43  </Rule>
44  <Rule Effect="Deny" RuleId="rule_matching_working_hour">
45    <Condition>
46      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-greater-than">
47        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-one-and-only">
48          <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#integer"
49                                      AttributeId="totalWorkingHours" />
50        </Apply>
51        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">12</AttributeValue>
52      </Apply>
53    </Condition>
54  </Rule>
55  <Rule Effect="Permit" RuleId="rule_permit_all"/>
56 </Policy>
```

Figure 4.13: Organizational Policy

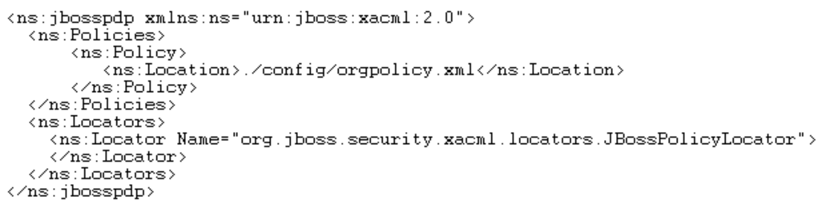
inner has a function ID of *string-at-least-one-member-of* on line 30. In addition, there is a *SubjectAttributeDesignator* tag on line 31 in this *Apply* function, which has two attributes: *DataType*, which has a *string* value, and *AttributeId* which has a value of *leadershipRole*. These attribute IDs and data type play an important role in the *Request* later. And the innermost has a function ID of *string-bag* on line 33, which contains all the attribute values of leadership roles for the string bag operation. The rule basically says that if the string value of the subject attribute designator named *leadershipRole* does not match at least one member of the leadership role values, return *Deny*. In other words, if the leadership role does match at least one member of the leadership role values, return *Permit*.

The second rule is called *rule_matching_working_hour* from lines 44 to 54 and if the *Condition* from lines 45 to 53 in this rule is satisfied, the *Effect*, which is *Deny*, is returned.

There are two nested *Apply* functions on lines 46 and 47. The outermost *Apply* function on line 46 has a function ID of *integer-greater-than.* It also has the *AttributeValue* of type *integer*, which has a value of twelve. The innermost *Apply* function on line 47 has a function ID of *integer-one-and-only.* These function IDs dictate the XACML engine how to evaluate this rule. In addition, there is a *SubjectAttributeDesignator* tag on line 48 in this *Apply* function, which has two attributes: *DataType*, which has a *integer* value, and *AttributeId*, which has a value of *totalWorkingHours.* The rule basically says that if the integer value of the subject attribute designator named *totalWorkingHours* is greater than twelve, return *Deny.* In other words, if the total working hours within a 12-hour period is less than or equal to 12, return *Permit.*

The third rule is called *rule_permit_all* on line 55 and it has no condition to be evaluated. Therefore, the *Effect* which is *Permit* is returned. Basically, when the XACML engine evaluates this rule, the other two rules described above must not return *Deny.* If so, the *Permit* in this rule is returned.

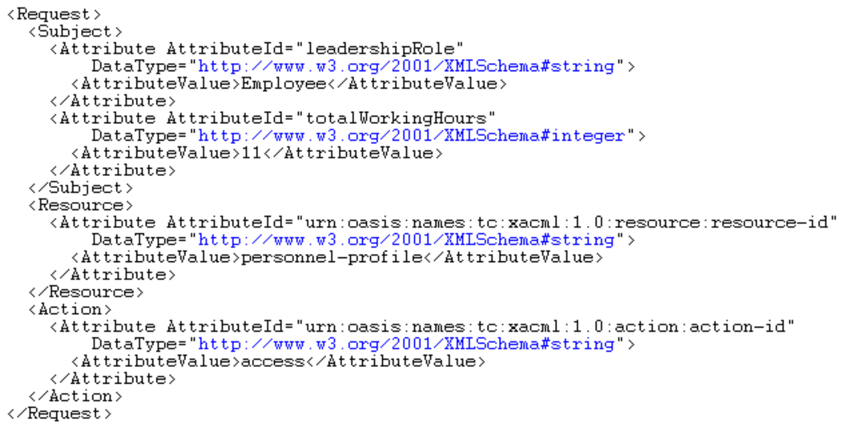
Because we use the JBoss XACML implementation, we encapsulate the policy XML document in the config.xml, which tells JBoss which class to load in order to locate the policy file.



```
<ns:jbosspdp xmlns:ns="urn:jboss:xacml:2.0">
  <ns:Policies>
      <ns:Policy>
         <ns:Location>./config/orgpolicy.xml</ns:Location>
      </ns:Policy>
  </ns:Policies>
  <ns:Locators>
    <ns:Locator Name="org.jboss.security.xacml.locators.JBossPolicyLocator">
    </ns:Locator>
  </ns:Locators>
</ns:jbosspdp>
```

Figure 4.14: JBoss XACML configuration file

Figure 4.14 illustrates the JBoss XACML configuration file. In this configuration file, an organization can have a number of policy files. Each policy file must be in its own tag called *ns:Policy*, which has the *ns:Location* tag. The physical location of the policy file is specified in this tag. We need to indicate the class name of the Policy Locator in the *ns:Locator* tag to JBoss so that JBoss can load the right class to locate the policy file.



```
<Request>
  <Subject>
    <Attribute AttributeId="leadershipRole"
        DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>Employee</AttributeValue>
    </Attribute>
    <Attribute AttributeId="totalWorkingHours"
        DataType="http://www.w3.org/2001/XMLSchema#integer">
      <AttributeValue>11</AttributeValue>
    </Attribute>
  </Subject>
  <Resource>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
        DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>personnel-profile</AttributeValue>
    </Attribute>
  </Resource>
  <Action>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
        DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>access</AttributeValue>
    </Attribute>
  </Action>
</Request>
```

Figure 4.15: XACML Request in XML

Figure 4.15 illustrates an XACML request for this simple policy. The request always has the Subject, Resource, and Action tags. Since we indicated in the policy two subject attribute designators whose attribute IDs are *leadershipRole* and *totalWorkingHours*, we must provide these two attributes in the request. In addition, we also indicate the resource and action in the policy. We must provide them in the request as well. Loading an XACML request in the XML works well if we validate a single request against a policy. In our framework, the request is dynamic because it comes from the users and the time of the request is unknown. Therefore, an XACML request in the XML file is not a robust option for our framework.

Java Coding Policy

XACML comes with the set of Java classes that allows the policy and the request to be created at runtime. Figure 4.16 illustrates how the simple policy can be created in the Java code.

In the OMC createReqCtx Algorithm (Algorithm 5), we describe the psuedo code of how the request context is created. The implementation of this algorithm requires the implementation of three interfaces: IPersonnel, IValidator, and IAttr. IPersonnel is the interface that allows to retrieve specific properties of a personnel. The implementation of IPersonnel interface is specific per emergency organization. Each organization may have different set of hiring and working requirements. IValidator is the interface that is also organizationally specific. It also encapsulates the organizational policies and allows to validate their employee working status against them. IAttr is the interface for attributes and their properties. The idea is that the validator retrieves the list of attributes required to evaluate a policy and the algorithm will retrieve the attribute value based on the attribute name from the IPersonnel. Therefore, IPersonnel, IAttr, and IValidator are the key interfaces and must work together for the success of the policy enforcement.

In the OMC replacePH Algorithm (Algorithm 6), we describe the psuedo code of how the replacement of a primary handle works. The algorithm needs the implementation of a time-out request interface ITOReq that allows the algorithm to get detailed information from the time-out request. Table 4.1 lists all the methods, return types, parameters, and their descriptions.

4.4.4 Third Party Implementation

In this third party space, we use Apache HTTP Web Server [42], JBoss Application Server version 5.1.0 GA [43], and MySQL Server [44]. We configure the Apache HTTP Web Server and set up the server's firewall to accept and respond to requests via Secure HTTP (HTTPS) port (443) only. We also recommend that the email and calendar services be hosted in this space. An organization can use their current email and calendar services and provide a simple service connection and implementation to the AERF framework. The open source Google email and calendar services [45] were chosen to illustrate the features in this framework.

User Manager, as its name implies, manages the Organization Monitor Control's user accounts. These accounts can be the OMC administrators and other leadership positions in the organization such as department chair, deputy, etc. A new user account can be easily added to the OMC system via a simple registration process. The User Manager service will handle the account transactions between the OMC and the MySQL user database. All the transactions from the OMC to the User Manager and from the User Manager back to the OMC via web services are encrypted with the server public key and the user's public key respectively. The public keys between the two parties are exchanged during the user registration process.

User Manager and AERF Services were implemented and deployed to the JBoss Application Server. These web services primarily accept and respond to requests from the OMC, which resides at the organization space. Each space has its own security guards against external and internal threats. When the information leaves one space and enters another, it can be tampered with, looked at, or disrupted by external parties with different intentions before reaching its destination or never reaching its destination at all. Keeping the authenticity of the information and allowing only authorized people to access it are tough challenges. The Public Key Infrastructure (PKI) was adapted to counter these security issues.

The AERF Service handles all the transactions of the AERF framework from the creation of the personnel, creation of work shifts, the assignment of the Primary Handler, to the establishment of the Backup list based on organization roles. Again, all the transactions are encrypted using the PKI scheme.

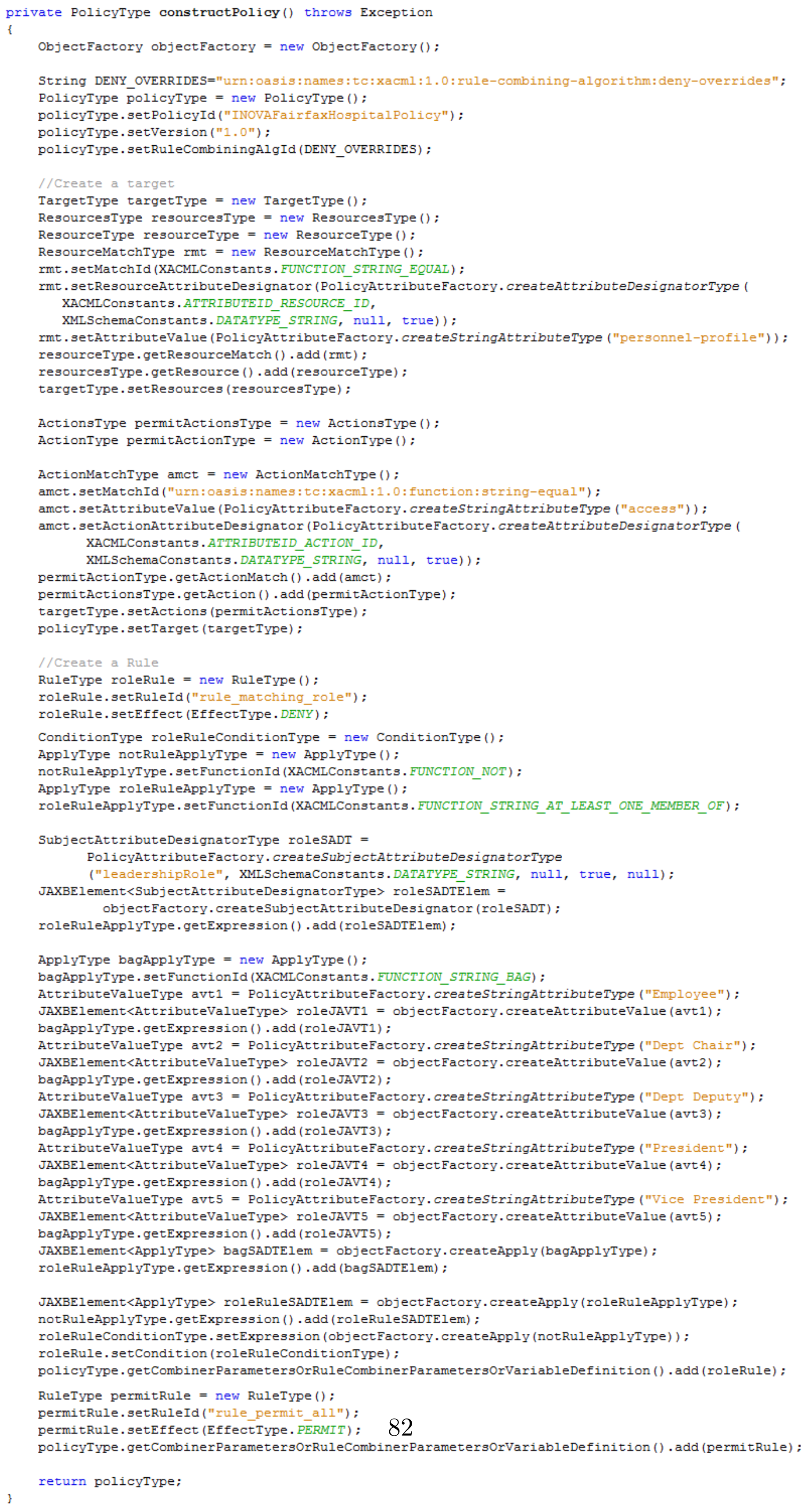
4.5 Related Works

In this section, we describe briefly other significant works aimed at improving the availability of emergency responders.

First is the National Response Framework (NRF) [56] developed by the Federal Emergency Management Agency as guiding principles that allow all response partners to prepare for and provide a unified and consistent national response to disasters and emergencies. The NRF establishes a comprehensive and national approach to deal with all hazards to domestic incident response. However, the NRF doesn't establish any guidelines on ensuring the availability of emergency responders. Without an intelligent approach to address the availability of partners' personnel in order to address emergency problems, we will continue to have potential and unpredictable chaos in dealing with them.

Second is the RFID-Based Secure Mobile Communication Framework for Emergency Response Management [57]. We consider this work as a complement to ours because the proposed framework addresses the availability of the latest and detailed information that can be presented to the emergency personnel. We believe that emergency personnel do need the latest and most detailed information about the pending emergency so that they can make an intelligent decision on what actions to resolve the emergency.

For the future work, we propose an integration of our AERF framework with 911 call centers so that the emergency call will be routed via 911 operators directly to emergency responders. We will investigate the forwarding feature in the Session Initiation Protocol (SIP) [58] and build a call forwarding algorithm to accept the role of emergency responders, translate it into the Primary Handler's calling number, and perform the automatic dialing. This future work will complete the end-to-end emergency call from an emergency caller to a handler.



```java
private PolicyType constructPolicy() throws Exception
{
    ObjectFactory objectFactory = new ObjectFactory();

    String DENY_OVERRIDES="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides";
    PolicyType policyType = new PolicyType();
    policyType.setPolicyId("INOVAFairfaxHospitalPolicy");
    policyType.setVersion("1.0");
    policyType.setRuleCombiningAlgId(DENY_OVERRIDES);

    //Create a target
    TargetType targetType = new TargetType();
    ResourcesType resourcesType = new ResourcesType();
    ResourceType resourceType = new ResourceType();
    ResourceMatchType rmt = new ResourceMatchType();
    rmt.setMatchId(XACMLConstants.FUNCTION_STRING_EQUAL);
    rmt.setResourceAttributeDesignator(PolicyAttributeFactory.createAttributeDesignatorType(
       XACMLConstants.ATTRIBUTEID_RESOURCE_ID,
       XMLSchemaConstants.DATATYPE_STRING, null, true));
    rmt.setAttributeValue(PolicyAttributeFactory.createStringAttributeType("personnel-profile"));
    resourceType.getResourceMatch().add(rmt);
    resourcesType.getResource().add(resourceType);
    targetType.setResources(resourcesType);

    ActionsType permitActionsType = new ActionsType();
    ActionType permitActionType = new ActionType();

    ActionMatchType amct = new ActionMatchType();
    amct.setMatchId("urn:oasis:names:tc:xacml:1.0:function:string-equal");
    amct.setAttributeValue(PolicyAttributeFactory.createStringAttributeType("access"));
    amct.setActionAttributeDesignator(PolicyAttributeFactory.createAttributeDesignatorType(
          XACMLConstants.ATTRIBUTEID_ACTION_ID,
          XMLSchemaConstants.DATATYPE_STRING, null, true));
    permitActionType.getActionMatch().add(amct);
    permitActionsType.getAction().add(permitActionType);
    targetType.setActions(permitActionsType);
    policyType.setTarget(targetType);

    //Create a Rule
    RuleType roleRule = new RuleType();
    roleRule.setRuleId("rule_matching_role");
    roleRule.setEffect(EffectType.DENY);
    ConditionType roleRuleConditionType = new ConditionType();
    ApplyType notRuleApplyType = new ApplyType();
    notRuleApplyType.setFunctionId(XACMLConstants.FUNCTION_NOT);
    ApplyType roleRuleApplyType = new ApplyType();
    roleRuleApplyType.setFunctionId(XACMLConstants.FUNCTION_STRING_AT_LEAST_ONE_MEMBER_OF);

    SubjectAttributeDesignatorType roleSADT =
          PolicyAttributeFactory.createSubjectAttributeDesignatorType
          ("leadershipRole", XMLSchemaConstants.DATATYPE_STRING, null, true, null);
    JAXBElement<SubjectAttributeDesignatorType> roleSADTElem =
            objectFactory.createSubjectAttributeDesignator(roleSADT);
    roleRuleApplyType.getExpression().add(roleSADTElem);

    ApplyType bagApplyType = new ApplyType();
    bagApplyType.setFunctionId(XACMLConstants.FUNCTION_STRING_BAG);
    AttributeValueType avt1 = PolicyAttributeFactory.createStringAttributeType("Employee");
    JAXBElement<AttributeValueType> roleJAVT1 = objectFactory.createAttributeValue(avt1);
    bagApplyType.getExpression().add(roleJAVT1);
    AttributeValueType avt2 = PolicyAttributeFactory.createStringAttributeType("Dept Chair");
    JAXBElement<AttributeValueType> roleJAVT2 = objectFactory.createAttributeValue(avt2);
    bagApplyType.getExpression().add(roleJAVT2);
    AttributeValueType avt3 = PolicyAttributeFactory.createStringAttributeType("Dept Deputy");
    JAXBElement<AttributeValueType> roleJAVT3 = objectFactory.createAttributeValue(avt3);
    bagApplyType.getExpression().add(roleJAVT3);
    AttributeValueType avt4 = PolicyAttributeFactory.createStringAttributeType("President");
    JAXBElement<AttributeValueType> roleJAVT4 = objectFactory.createAttributeValue(avt4);
    bagApplyType.getExpression().add(roleJAVT4);
    AttributeValueType avt5 = PolicyAttributeFactory.createStringAttributeType("Vice President");
    JAXBElement<AttributeValueType> roleJAVT5 = objectFactory.createAttributeValue(avt5);
    bagApplyType.getExpression().add(roleJAVT5);
    JAXBElement<ApplyType> bagSADTElem = objectFactory.createApply(bagApplyType);
    roleRuleApplyType.getExpression().add(bagSADTElem);

    JAXBElement<ApplyType> roleRuleSADTElem = objectFactory.createApply(roleRuleApplyType);
    notRuleApplyType.getExpression().add(roleRuleSADTElem);
    roleRuleConditionType.setExpression(objectFactory.createApply(notRuleApplyType));
    roleRule.setCondition(roleRuleConditionType);
    policyType.getCombinerParametersOrRuleCombinerParametersOrVariableDefinition().add(roleRule);
    RuleType permitRule = new RuleType();
    permitRule.setRuleId("rule_permit_all");
    permitRule.setEffect(EffectType.PERMIT);
    policyType.getCombinerParametersOrRuleCombinerParametersOrVariableDefinition().add(permitRule);

    return policyType;
}
```

Figure 4.16: XACML Java Policy Implementation

Table 4.1: Interfaces for Policy Implementation

Interface	Method	Return Type	Description
IPersonnel	getAV(String AN)	String	Return the attribute value for this attribute name (AN).
IValidator	getSubjectAttrs	List IAttr	Return the list of subject attributes in the policy.
	getResName	String	Return the primary resource name.
	getResValue	String	Return the primary resource value.
	getResType	String	Return the primary resource type.
	getActName	String	Return the primary action name.
	getActValue	String	Return the primary action value.
	getActType	String	Return the primary action type.
	acqLock	boolean	Return true if the lock can be acquired sucessfully.
	relLock	void	Release the lock on the policy.
	checkOrgPlc	boolean	Return true if the organization rules and conditions in the organization policy has been satisfied.
IAttr	getAN	String	Return the attribute name.
	getAT	String	Return the attribute type.
ITOReq	getTimeStart	long	Return the time start in milliseconds.
	getTimeEnd	long	Return the time end in milliseconds.
	getOrgRole	String	Return requester's organization role.
	isSevere	boolean	Return true if the timeout request reason is considered to be SEVERE at the severity level.

Chapter 5: The Emergency Navigation Assistance Framework

5.1 Navigation Assistance Framework Requirements and Use Cases

In this section, we will specify some requirements and objectives that responsible organizations might implement in their processes and operations in order to provide emergency information to other trusted organizations. Below is the list of our requirements and objectives. *Requirements* are features or functions that are necessary to meet the minimum needs of the navigation assistance. A requirement contains the word "shall" and is identified by the letters "**R**". An *Objective* is feature or function that is desirable, but not mandatory, that the NAF should be capable of incorporating into its architecture at some future point. An Objective contains the words "it is desirable" and is identified by the letters "**O**".

R#1: There shall be a way to provide the current information about an ongoing emergency.

R#2: There shall be a way to provide directions to avoid an ongoing emergency.

O#1: It is desirable that users provide daily events in their calendars and expose them to the trusted entity.

5.1.1 Use Case Study

In this subsection, we describe use cases that are derived from the requirements above.

Use Case 1: A driver with an ERApp running on his hand-held device drives to work as a part of his regular routine.

The following two use cases occur when the driver receives a CMAS message informing him that there is a tornado in the area. ERApp appears on his hand-held device, showing the location of the ongoing tornado, his location and the work location. He then determines that:

Use Case 2: his route to work has not been impacted by the ongoing tornado.

Use Case 3: his route to work is in danger by the ongoing tornado.

The first use case illustrates a sunny day scenario where drivers don't encounter any problems on the road that prevent them from arriving at work on time. However, traffic accidents and natural emergencies such as tornados, heavy rains, blizzards, snow storms, hail, or other severe weather conditions would prevent drivers from arriving at work on time. Delays expected in these emergencies can be up to hours. The second two use cases illustrate that an emergency has occurred in the area. In this case, we illustrate the ongoing tornado because the tornado is a medium scale emergency and its movement can be tracked by the National Weather Center (NWC). The NWC then can provide the Navigation Assistance Framework crucial data such as the direction and the speed of the tornado. We can then use these data to calculate and estimate the impact further.

5.2 Navigation Assistance Framework Architecture Design and Implementation

In this section, we will describe major components and their functionality that constitutes to the Navigation Assistance Framework (NAF) Architecture. We then discuss, in more detail, our architecture design. First, we provide a high-level description of each component and its function in the overall architecture.

5.2.1 Architecture Components

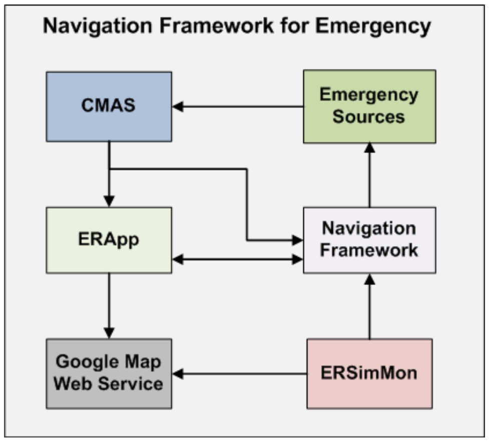


Figure 5.1: Navigation Assistance Framework High-Level Components

Figure 5.1 depicts the high-level architecture components for the Navigation Assistance Framework. It consists of Emergency Sources, Commercial Mobile Alert System (CMAS) [18], Navigation Assistance Framework, Emergency Response Application (ERApp), Emergency Response Simulation Monitor, and Google Map Services.

Emergency Sources are emergency systems that have the capability to monitor the emergency progression and to provide updates if needed to other systems. For example, the National Hurricane Center is considered one of the Emergency Sources. The CMAS broadcasts emergency 90-character text messages to all mobile users. The CMAS receives emergency data from Emergency Sources. The CMAS operator will generate the broadcast message based on the emergency data and broadcast it. There are a few enhancements as described in sections 3.5 and 3.3 that we made to improve the content of the broadcast message as well as the area effected by the emergency to which the message is sent.

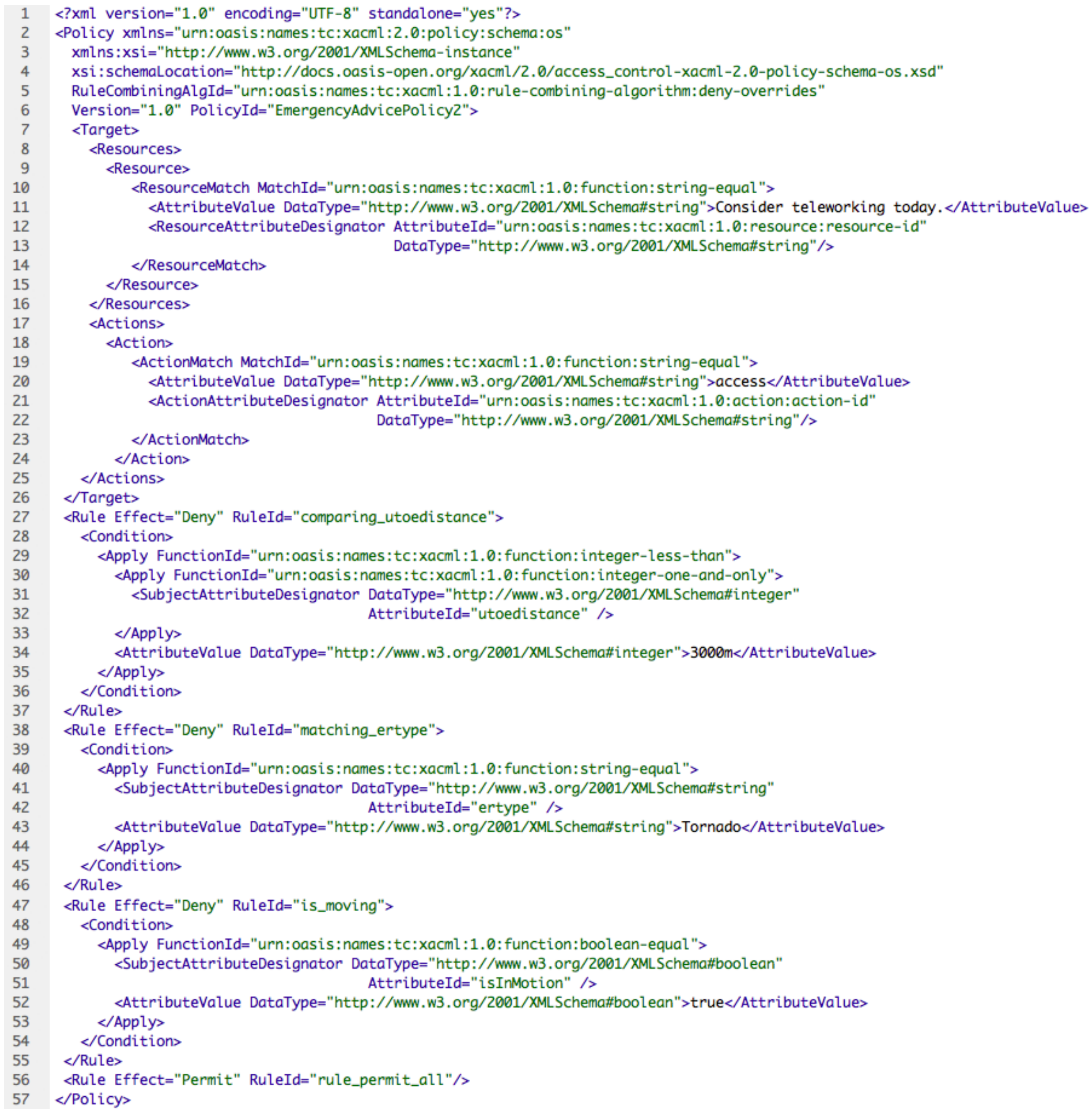
The Emergency Response Application (ERApp) is a smart phone platform, which intercepts the enhanced CMAS broadcast message, displays the user's location with respect to the emergency location on the map, and provides the navigational assistance and recommended actions to help the user navigate out of the ongoing emergency. The Navigation Assistance Framework provides the set of interfaces that Emergency Sources need to implement and acts as a listener to the emergency data and advice policies. When the emergency data or advice policies are available, the Emergency Sources send them to the Navigation Assistance Framework. The ERApp can make an advice policy update request to the NAF when the ERApp detects that the user is in motion during an ongoing emergency. The ERApp applies the advice policies to see if the current status is satisfied the conditions on the policy and displays the emergency advice of that policy. For example, the advice policy can simply say that if the user is at rest and is 3000 meters away from the center of the emergency, ERApp will display the advice that the user needs to consider teleworking for today. Figure 5.2 shows the policy written in XACML [10].

The Emergency Simulation Monitor (ERSimMon) uses the Navigation Assistance Framework to simulate an emergency and the people who are trying to navigate through it. The ERSimMon uses the Google Maps API web services to query with the list of emergency constraints and road congestion information for the best routes that the users can take to reach their desired destinations.

For these components to work together smoothly and seamlessly, we need to make a couple of enhancements to the existing services such as the Google Maps API web services and Emergency Source Web Services. These enhancements allow the Navigation Assistance Framework to be used in the most effective way and expose its full capabilities. Here is a list of enhancements:

First, we make an enhancement to the Google Maps API web services [67] to include a list of constraints and road blocks. The original URL to get directions from Google is `http://maps.googleapis.com/maps/api/directions/xml?origin=[]&destination=[]&sensor=[true|false]`

where the *origin* parameter specifies the origination address. The *destination* parameter



```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-policy-schema-os.xsd"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides"
  Version="1.0" PolicyId="EmergencyAdvicePolicy2">
  <Target>
    <Resources>
      <Resource>
         <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
           <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Consider teleworking today.</AttributeValue>
           <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
                                        DataType="http://www.w3.org/2001/XMLSchema#string"/>
         </ResourceMatch>
      </Resource>
    </Resources>
    <Actions>
      <Action>
         <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
           <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">access</AttributeValue>
           <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
                                      DataType="http://www.w3.org/2001/XMLSchema#string"/>
         </ActionMatch>
       </Action>
   </Actions>
 </Target>
 <Rule Effect="Deny" RuleId="comparing_utoedistance">
   <Condition>
     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-less-than">
       <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-one-and-only">
         <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#integer"
                                     AttributeId="utoedistance" />
       </Apply>
       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">3000m</AttributeValue>
     </Apply>
   </Condition>
 </Rule>
 <Rule Effect="Deny" RuleId="matching_ertype">
   <Condition>
     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
                                   AttributeId="ertype" />
       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Tornado</AttributeValue>
     </Apply>
   </Condition>
 </Rule>
 <Rule Effect="Deny" RuleId="is_moving">
   <Condition>
     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:boolean-equal">
       <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#boolean"
                                   AttributeId="isInMotion" />
       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#boolean">true</AttributeValue>
     </Apply>
   </Condition>
 </Rule>
 <Rule Effect="Permit" RuleId="rule_permit_all"/>
</Policy>
```

Figure 5.2: Emergency Advice XACML Policy

specifies the destination address. The *sensor* parameter indicates that the directions request comes from a device with a location sensor. There are a few optional parameters such as mode= [driving|walking|bicycling|transit], waypoints, alternatives= [true|false], avoid= [tolls|highways], language, units, region, departure_time, and arrival_time. None of these parameters provide us the directions which avoid the emergency road blocks or help us navigate around the pending emergency such as tornado. At best, the *alternative* parameters provide us several routes to the destination, but they don't guarantee that the routes will avoid emergency road blocks or the pending emergency.

Therefore, our enhancement would be adding one parameter *eblocks* into the Google Maps API web service, which gives the GPS location and the radius of the blocking area. The parameter has three values separated by comas: latitude, longitude, and radius. For example: eblocks=38.8462236,-77.3063733,500m. In this example, we indicate that the emergency occurs in Fairfax, VA which has the GPS location of 38.8462236,-77.3063733 and we should avoid all the roads around that particular location of at least 500 meters in radius.

Second, we enhance the emergency source service to provide us with the most up-to-date emergency information. We provide emergency sources with a set of APIs so that they can provide their implementation and connect with our framework via web service.

5.2.2 Supporting Use Cases

We describe the functions built into the Navigation Assistance Framework to support these use cases. In the sunny day scenario, users can get the navigation assistance from the regular GPS or the ERApp. Without any emergency occurrences during the rush hours, users can anticipate their on-time arrivals at their desired destinations. However, emergency incidents do occur at unexpected times and have the potential to create a long delay in travel time. With the ERApp installed on handheld devices, users are able to receive an enhanced CMAS broadcast emergency message in section 3.5 that provides more details about the ongoing emergency incident. In addition, ERApp is equipped to receive minute updates about the ongoing emergency status including tracking information such as GPS location, time, intensity, effected area, etc. These information are crucial to ERApp to further assist users in terms of providing better and more informative advice and directions to assist users reach their destinations. The goal is to avoid possible road blocks and dangerous areas being affected by the emergency incident. We can achieve this goal if we have the up-to-date emergency information.

Emergency Data

Depending upon the type of emergency incidents, information may come from different sources. For example, tornado data may come from National Weather Center. Hurricane data may be retrieve from National Hurricane Center. Road closure in the local area may come from the local police department. Therefore, we need to establish a method of retrieving these emergency data from various sources and determine if the connection is either push, pull, or both.

The Navigation Assistance Framework acts as a centralized emergency process which dispatches emergency information to all devices and receives regular updates from emergency sources. With this, the Navigation Assistance Framework needs to subscribe to all of the emergency sources to pull and push emergency data. In addition, the NAF needs to allow ERApp to subscribe to receive emergency updates. Depending on the users' circumstances and their activities, the NAF allows the pull subscription in which the ERApp requests emergency data in a form of a pull as needed.

Each emergency has its own unique data set that is relevant to our framework. For a tornado, we collect the following emergency data: time when the tornado touched down, its track including the GPS location of the tornado, wind speed and direction, and the storm intensity measured in Fujita Scale (F-Scale) [68].

For road closure emergencies such as a car accident, road construction, water main break, etc., we collect the following emergency data: starting date and time of the closure, the anticipated date and time of the re-opening of the road, GPS location, and the radius of the affected area. The purpose of getting these data is to provide the framework the magnitude of the emergency in terms of size, GPS location, and its severity. This allows the framework to approximate the danger area and to provide minute updates to ERApp so that ERApp can assist users to navigate around the danger area to their destinations safely and in a timely matter.

Updating the Directions

After the Navigation Assistance Framework receives the emergency data from the Emergency Source, it sends the updated data via text messaging to all the devices that have ERApp installed and are registered with the NAF. The ERApp determines what the next step is going to be. ERApp will formulate a new query to get the updated routes to the destination, given that the ongoing emergency is not going to be in the forecast path. We make a general assumption that users will manually enter into their event calendars the location addresses of where they will be and from what time to what time they are going to be there. These calendar fields such as location, time start and time end are in the Internet Calendar Specification [59]. The ERApp will use this location address as the destination. The ERApp will send the GPS location and the radius of the affected area to Google Maps, which in turn provides the updated directions to the destination.

The format of the text message must be agreed upon and interpreted the same way as intended. The format is composed by the list of name-value pairs. The name must be abbreviated by 2 capitalized characters. Each character is one byte. The value must be in primitive types. These names and and associated types must be accessible by the NAF and the ERApp. Table 5.1 provides these names, associated types and short descriptions of each attribute.

According to the SMS Specification [69], an SMS Message has a limitation of 160 readable characters. If we represent this SMS message in the readable characters, there may not be enough readable characters to hold values of all these attributes. Therefore, we represent these values in binary values and then we encode using Base 64 Encoding [30].

Providing Relevant Advice

The ERApp also provides its user with relevant advice based on her current circumstance, which can be illustrated by the list of to-do tasks and/or appointments/meetings that she

Table 5.1: Text Message Values

Parameter	**Type**	**Description**
ID	Integer	Emergency Identification
ET	Integer	Emergency Type
LT	Long	Latitude
LN	Long	Longitude
RD	Integer	Radius
DR	Byte	Emergency Direction
SP	Integer	Speed of moving emergency
AP	String	Advice policies in XML format

needs to do or go to, respectively. We illustrate this use case with the following scenario.

Let's say Mr. A departs from his house in Centreville, VA at 8AM on his way to a doctor's appointment at 9AM in Arlington, VA, a couple of miles from his work place. Normally, it takes only 45 minutes to work on a sunny day. However, as soon as he enters the road, he receives an alert from his ERApp that warns him about an ongoing tornado in Fairfax, VA with the affected area of 2-mile radius. He immediately is asked to change his routes to avoid the tornado. However, according to the new calculation of the estimated time of arrival, he will arrive at the doctor office at 9:30AM. With this current situation, he won't be able to arrive in time for the doctor's appointment. The ERApp will advice him to call the doctor's office to cancel the appointment in a form of message notification. ERApp then advises him to go straight to work and provide the updated direction for him.

5.2.3 Implementation

This subsection will detail the implementation of the Navigation Assistance Framework. We also suggest some interfaces that Emergency Sources and Google Maps Web Services need to implement to make this framework. However, we provide the prototype implementation of these interfaces to show how they all work together.

Emergency Data Collection

Figure 5.1 shows that the NAF receives emergency data from two sources such as CMAS and Emergency Sources. CMAS sends the CMAS message as described in section 3.5 to the NAF in its broadcast. Two important pieces of data are the GPS location of the ongoing emergency and the radius of the affected area. The NAF will use this information to load the ongoing emergency onto the map as shown in Figure 5.3.

The NAF continues to request direct updates from the Emergency Sources periodically or if there is a new update. For example, in the case of a tornado, the tornado's location, direction, and wind speed are changing minute by minute. The Emergency Sources can push it to the NAF directly. This information is very important to the NAF. We provide the interface in table 5.2 for the Emergency Sources so they can implement.

Table 5.2: Interface for IESource Implementation

Interface	**Method**	**Return Type**	**Parameters**	**Description**
IESource	sendUpdate	int	(int iEID, HashMap mapNVP)	Send update values for the ongoing emergency.
	pullUpdate	HashMap	(int iEID)	Return the update name-value pair hashmap for the ongoing emergency.

We need to establish an understanding of the emergency data that we receive from each of the Emergency Sources. This understanding allows us to fully interpret data. We also

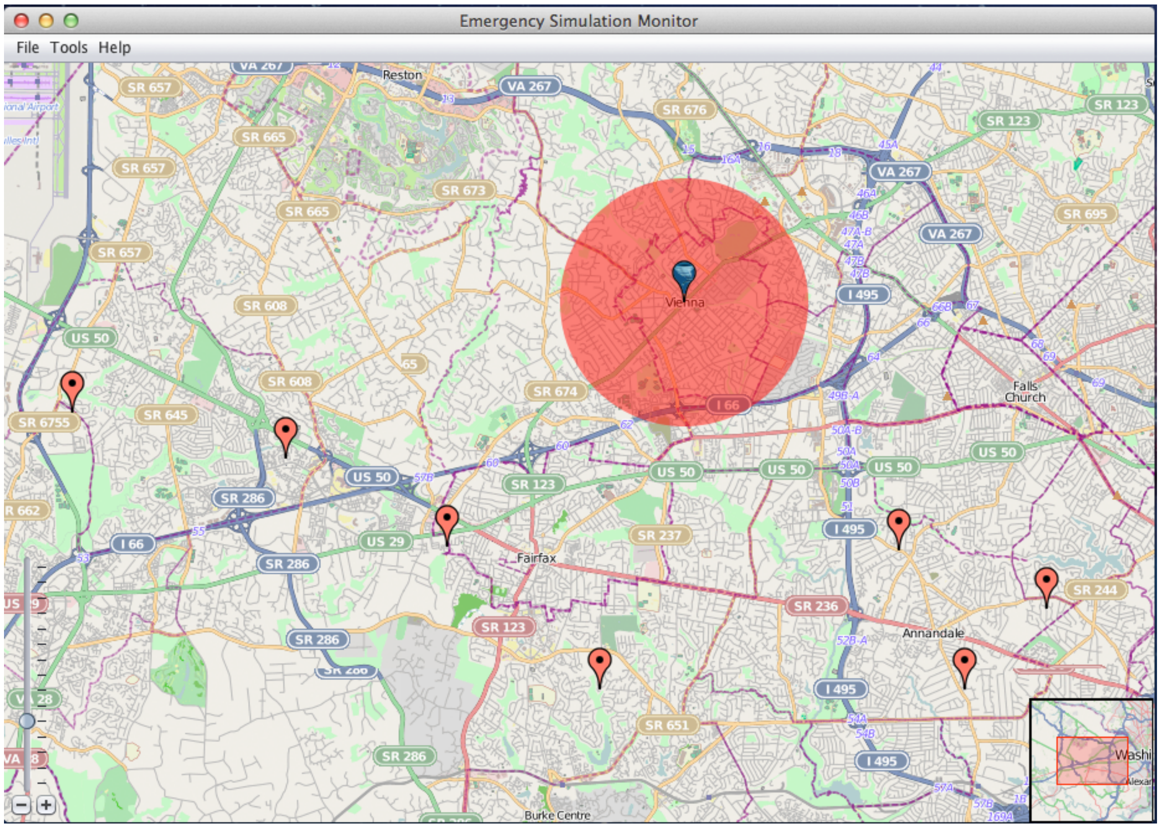


Figure 5.3: Ongoing Tornado

establish the agreement of the names for data. Here are some of the suggested names of tornado data that interest us:

These suggested names and types are the binding agreements between the NAF and Emergency Sources. We use these names based on the type of emergencies to retrieve their associated values and convert them based on the attribute types.

Navigation Update

The NAF sends an updated text message to all the registered ERApps installed on Android devices. The ERApp will have to first use the Base 64 Decoder to decode the text message to the binary string.

Table 5.3: Data Name Space

Emergency	**Data Name Space**	**Data Type**	**Description**
Tornado	TND_Longitude	Long	GPS longitude location.
	TND_Latitute	Long	GPS location latitude location.
	TND_Direction	String	Current Direction: East, West, North, South, North-East, South-West, etc.
	TND_Wind_Speed	Integer	Wind Speed measured in miles/hour.
	TND_Radius	Integer	Radius of the affected area measured in mi (miles), m (meters), km (kilometers).
Road Blocks	RB_Longitude	Long	GPS longitude location.
	RB_Latitute	Long	GPS location latitude location.
	RB_Radius	Integer	Radius of the affected area measured in mi (miles), m (meters), km (kilometers).

Algorithm 7 shows how to extract attribute names and values from the SMS message. It requires that the required parameter *smsMsg* is not null. On line 1, the hashmap *mapNVP* is created to store all the decoded names and values. The Base 64 instance is created from line 2. On line 3, we decode the smsMsg using the Base 64 decoder. The decoded string is then assigned to *binaryStr*. On line 4, the map attribute *mapAttrs* is retrieved from the SMSAttrs object, which maintains the list of agreed names and types for the SMS message that is communicated between the NAF and the ERApp. We then create the byte array input stream with the *binaryStr* as the input parameter and assign it to *baInputStream* on line 5. On line 6 to 15, we go into the *while* loop and read one attribute at a time. The attribute name is a 2-character string. We read the first two bytes on line 7 into the byte array called *baName*. On line 8, the attribute name *attrName* is created as a string value. Based on the attribute name, we can retrieve other information from the hash map

Algorithm 7 :**decodeSMS** Algorithm (Input: String smsMsg)

```
Require: $smsMsg \neq null$
1: $mapNVP \leftarrow new\ HashMap()$
2: $base64 \leftarrow Base64.getInstance()$
3: $binStr \leftarrow base64.decode(smsMsg)$
4: $mapAttrs \leftarrow SMSAttrs.loadAttrs()$
5: $baIS \leftarrow new\ ByteArrayInputStream(binStr)$
6: while $baIS.available()\ >\ 0$ do
7:     $baName \leftarrow baIS.read(2)$
8:     $attrName \leftarrow new\ String(baName)$
9:     $attr \leftarrow mapAttrs.getAttr(attrName)$
10:    $len \leftarrow attr.getAttrLen()$
11:    $baValue \leftarrow baIS.read(len)$
12:    $type \leftarrow attr.getType()$
13:    $attrValue \leftarrow convertValue(type,\ baValue)$
14:    $mapNVP.put(attrName,\ attrValue)$
15: end while
16: return $mapNVP$
```

of attributes on line 9. We call this object *attr*. The byte length *len* of the attribute value is retrieved from the attribute on line 10. The attribute value in the byte array is read from the byte array input stream *baInputStream* on line 11. The type of the attribute is also retrieved from the attribute object on line 12. We then convert the byte array value according to its type on line 13 by calling a simple *convertValue* function. we then hash the attribute name with its value into the Name Value Pair hash map *mapNVP*. Finally, we return the hash map of name and value pairs on line 14.

The ERApp then interprets use these information to query the Google Maps web services for updated directions. In the prototype implementation, we use Google Calendar to retrieve users' calendar event information such as the location and the time.

Algorithm 8 illustrates the direction which avoids the affected area of the ongoing emergency and helps users navigate to their destinations. The algorithm takes three parameters, which must not be null. The first parameter *mapNVP* is the hash map containing name-value pairs and which is checked against null value. The second parameter *up* is the user profile, which contains the email credentials to access the calendar. And the third parameter *calURL* is the Google Calendar web service URL. On lines 1 and 2, the algorithm initializes

Algorithm 8 :**getUpdatedDirections** Algorithm (Input: HashMap mapNVP, UserProfile up, String calURL)

```
Require: $mapNVP \neq null$
Require: $uprofile \neq null$
Require: $calURL \neq null$
 1: $xmlDirDoc \leftarrow null$
 2: $calEvent \leftarrow null$
 3: $clientCal \leftarrow newCalendarService()$
 4: $clientCal.setCredentials(up.getWEmail(),\ up.getWEmailPC())$
 5: $calQuery \leftarrow newCalendarQuery(calURL)$
 6: $resultEvents \leftarrow clientCal.query(calQuery)$
 7: if $resultEvents.getEntries().size() > 0$ then
 8:    $resultEvents \leftarrow sortEvents(resultEvents)$
 9:    $iterEvents \leftarrow resultEvents.getEntries().iterator())$
10:    while $iterEvents.hasNext()$ do
11:       $calEntry \leftarrow iterEvents.next()$
12:       if $calEntry.getTimeStart() > now()$ then
13:          $calEvent \leftarrow calEntry$
14:          $break$
15:       end if
16:    end while
17: end if
18: if $calEvent\ is\ NOT\ null$ then
19:    $dest \leftarrow calEvent.getLocation()$
20:    $erLat \leftarrow mapNVP.get("LT")$
21:    $erLon \leftarrow mapNVP.get("LN")$
22:    $erR \leftarrow mapNVP.get("RD")$
23:    $urlDir \leftarrow formURL(erLat, erLon, erR, dest)$
24:    $url \leftarrow URL(urlDir)$
25:    $inputStream \leftarrow url.openstream()$
26:    $dbf \leftarrow DocumentBuilderFactory.newInstance()$
27:    $db \leftarrow dbf.newDocumentBuilder()$
28:    $xmlDirDoc \leftarrow db.parse(inputStream)$
29:    $xmlDirDoc.getDocumentElement().normalize()$
30: end if
31: return $xmlDirDoc$
```

two temporary variables *xmlDirDoc* and *calEvent* to null respectively. The *xmlDirDoc* is the updated direction in XML format, which is the return value for this algorithm. On line 3, the calendar service is created for the ERApp client called *clientCal*. On line 4, client calendar is set with the credentials including the email address and the email passcode, which are used to authenticate the calendar service. The calendar query is created from the calendar URL on line 5. We begin to query calendar events from the calendar service on line 6. We check if there is any entry in the return result on line 7. We then sort all the events based on time from the earliest to the latest on line 8. We create the event iterator on line 9 and go through all the calendar events on line 10. We retrieve the calendar event entry *calEntry* on line 11. If the event time is greater than the current time on line 12, we set the calendar event *calEvent* to that *calEntry* on line 13 and exit out of the while loop on line 14.

On line 18, the *calEvent* is tested for null value. If it is null, then the algorithm ends there with and return the null *xmlDirDoc* on line 31. If *calEvent* is not null, the destination will be retrieved from the calendar event on line 19. The algorithm retrieves the emergency latitude, longitude, radius of the affected area from the hash map name-value pairs *mapNVP* on lines 20, 21, and 22 respectively. The Algorithm then forms the Google map URL *urlDir* with parameters such as the current location, destination, affected area radius, and and the emergency GPS location on line 23. The URL object is created from the *urlDir* on line 24. The input stream *inputStream* is created from the URL object on line 25. On line 26, the document builder factory *dbf* instance is created, which in turn creates the document builder *db* on line 27. The algorithm parses the input stream to create the XML document *xmlDirDoc* on line 28. The document element is then normalized on line 29. The algorithm returns the *xmlDirDoc* on line 31. The ERApp can invoke any generic built-in application such as Maps, Navigation, etc. with the *xmlDirDoc* updated direction to provide assistance to the users.

Determine the Need for Alternative Routes

This section discusses the algorithm to determine the need if commuters need to get an alternative route to their destination. If the direction of the mobile users to the destination crosses the emergency area determined by the emergency location and its affected area radius, we need to get an alternative route to avoid the emergency area. We can easily retrieve the directions of users' moving vehicles by using the Accelerometer sensor and GPS sensor to determine the vector (speed and direction) of the moving vehicle. But this direction is only temporary and not the primary direction of where they are heading. Therefore, we need to retrieve the direction from where they are to their destinations. We retrieve the location, speed, direction, and the affected area (radius from the emergency location) of the ongoing emergency from the CMAS message as discussed in section 5.2.3.

Algorithm 9 : **isAltRouteNeeded** Algorithm (Input: HashMap mapNVP, GPSLocation gpsULoc, GPSLocation gpsDest)

```
Require: $mapNVP \neq null$
Require: $gpsULoc \neq null$
Require: $gpsDest \neq null$
 1: $isAltRouteNeeded \leftarrow false$
 2: $lat \leftarrow mapNVP.gets("LT")$
 3: $lng \leftarrow mapNVP.gets("LN")$
 4: $rd \leftarrow mapNVP.gets("RD")$
 5: $gpsELoc \leftarrow new\ GPSLocation(lat,\ lng)$
 6: $distUtoE \leftarrow getFlyingDist(gpsULoc,\ gpsELoc)$
 7: $bearingEL \leftarrow calculateBearing(gpsULoc,\ gpsELoc)$
 8: $bearingDest \leftarrow calculateBearing(gpsULoc,\ gpsDest)$
 9: $angleUEtoT \leftarrow arcsin(rd/distUtoE)$
10: $angleBE \leftarrow bearingEL - angleUEtoT$
11: $angleEE \leftarrow bearingEL + angleUEtoT$
12: if $bearingDest \geq angleBE\ AND\ bearingDest \leq angleEE$ then
13:     $isAltRouteNeeded \leftarrow true$
14: end if
15: return $isAltRouteNeeded$
```

Algorithm 9 discusses the need for the alternative routes. The algorithm accepts three parameters: *mapNVP*, *gpsULoc*, and *gpsDest*, which must match the requirement that they cannot be null. The first parameter is the hash map of the name-value pairs from the SMS message sent by the NAF. The second parameter is the GPS user location. And the third paramter is the GPS destination location. On line 1, *isAltRouteNeeded* is false. On lines 2 to 4, latitude *lat*, longitude *lng*, and affected area radius *rd* of the emergency are retrieved. The GPS location *gpsELoc* of the emergency is created on line 5. The flying distance [63] *distUtoE* from the user location to the emergency location is calculated on line 6. The bearing angle *bearingEL* formed between the North and the line from the user location to emergency location is calculated one line 7. The bearing angle *bearingDest* formed by the Northern line and the line from the user location to the destination location is calculated on line 8. The angle *angleUEtoT* formed by the line from the user location to the emergency location and the tangent line is calculated on line 9. The angle *angleBE* marked the beginning of the emergency effected area is calculated on line 10. On line 11, the angle *angleEE* marked the end of the emergency effected area is calculated. We are now ready to verify if the bearing angle of the destination is in the angle range of the beginning and end of the emergency affected area on line 12. If the *bearingDest* is within the range, the *isAltRouteNeeded* is set to true on line 13 and returned on line 15. The figure 5.4 provides the visual map these locations.

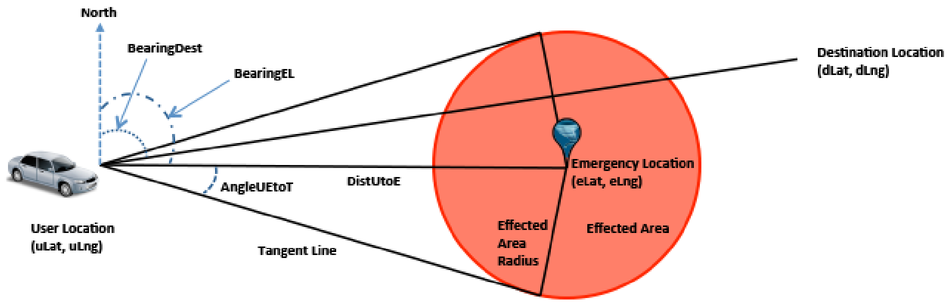


Figure 5.4: Alternative Route Decision

Providing Emergency Advice

The ERApp uses the accelerometer sensor built in the hand-held devices to detect the user's movement. By comparing the x and y acceleration components from one moment to the next, the ERApp can estimate if the user is moving or at rest. The ERApp then can compare the distance between the user location and the emergency location to know if he is approaching the emergency area. If the distance calculation indicates that the user is moving toward the emergency area, the ERApp can provide some intelligent advice to the user based on the nature of the emergency.

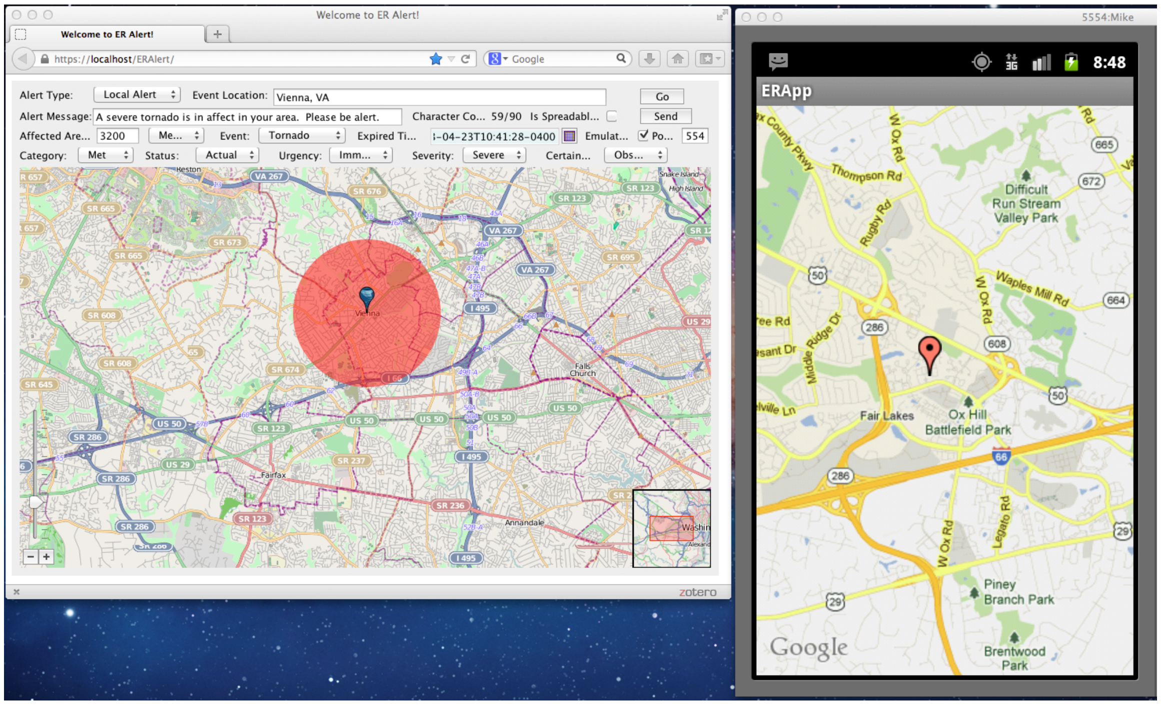
The advice can also be given based on the user location with respect to the ongoing emergency. For example, if the user is inside the affected area of a tornado, the relevant advice would be drive to the nearest shelter immediately. We can compare the distance from the user location to the emergency location with the radius of the affected area to see if the user is inside the affected area.

5.3 Experimentation

Before most, we need to generate a tornado alert informing the all people in the local area that the tornado is coming. After we set the emergency location to be in Vienna, Virginia. We also set the radius of the effected area to be 3200 meters from the center of the tornado. We also set the expired time, category, certainty, status, urgency, and severity of the tornado. Instead we broadcast this alert out, we will broadcast the message to an emulator running on port 5554. Figure 5.5 shows the preparation of the tornado alert and the ERApp running on the emulator showing the user's location.

The CMAS authority is ready to send the broadcast tornado alert to the emulator by clicking on the Send button. Figure 5.6 shows the tornado with the effected area in red and the user's location.

We built an Emergency Respond Simulation Monitor (ERSimMon) prototype to simulate an individual driving to work during an ongoing tornado. Figure 5.3 shows the map of the area, the ongoing tornado, and several marker points. These marker points represent users that are currently on the map. Figure 5.7 shows the configuration settings that are necessary for the simulation. In these configuration settings, Distance Increment is set to 50 meters for the duration of 200 milliseconds, which is indicated by the Sleeping


Welcome to ER Alert!

https://localhost/ERAlert/

Alert Type: Local Alert Event Location: Vienna, VA Go

Alert Message: A severe tornado is in affect in your area. Please be alert. Character Co... 59/90 Is Spreadabl... ☐ Send

Affected Are... 3200 Me... Event: Tornado Expired Ti... -04-23T10:41:28-0400 Emulat... ✓ Po... 554

Category: Met Status: Actual Urgency: Imm... Severity: Severe Certain... Obs...

Figure 5.5: Prepare the Tornado Alert

Duration. Emulation Location and ADB Location are required to run the Android Phone emulation. Speed Display is set to Miles per Hour. As the user moves from his location to the destination, it can display the speed at which the user is moving.

Figure 5.8 shows how to search, add, modify, or delete markers. Two required fields are the user name and the emulator name. The Address indicates the start point of the user on the map. The Destination Address indicates the ending point of the user on the map after the simulation is complete. These addresses are real address because we use the Google Maps API web service to retrieve the user's location and place the marker on the map. Type is the internal classifications or groups of users. For example, all the professors can be grouped together under Professor type. Telnet Server is the loopback server that the emulator is running on. The ERSimMon will set the new position as the user makes a movement. This process simulates the actual driving of the user and at a certain time

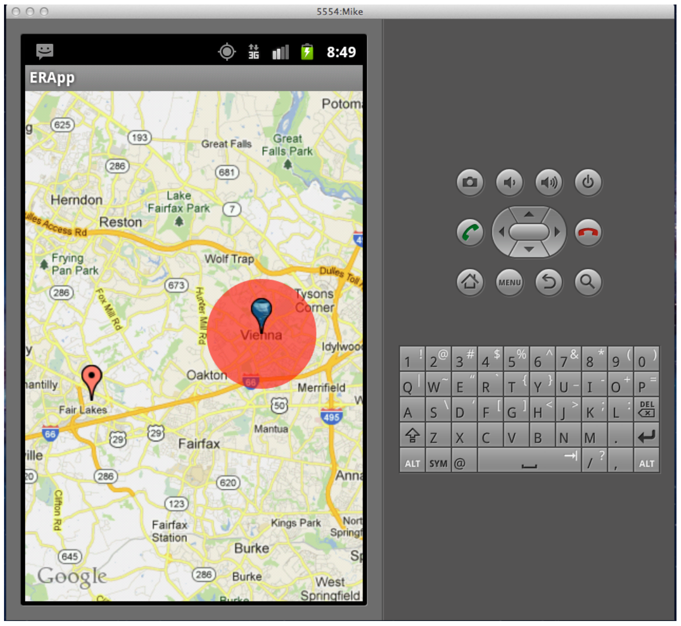


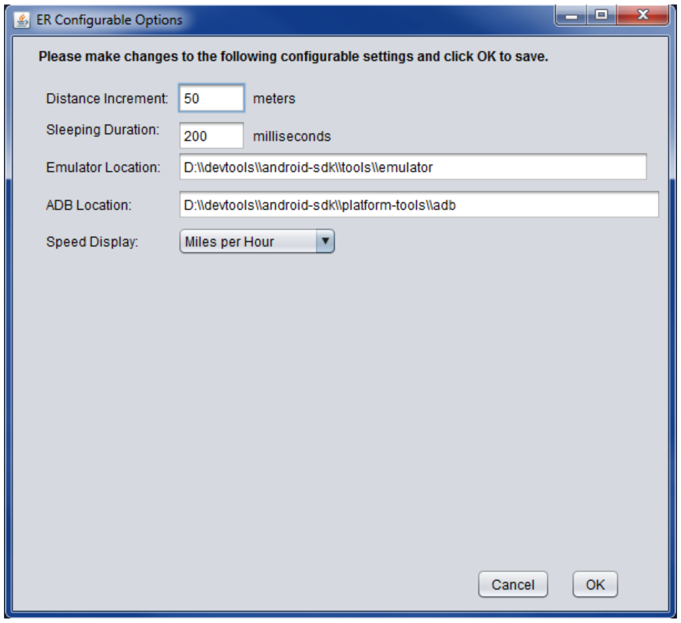
Figure 5.6: Tornado Alert on ERApp

interval, the GPS on the user hand-held device will detect its new position, which triggers the ERApp to update the position on the map.

Figure 5.9 shows the screen that which users (markers) will be simulated. After making selections, click Simulate button at the bottom of the screen.

Figure 5.10 shows the initial screen that simulates the driving of the selected user. The simulation will spawn the emulator on the right, showing the user's location and the ongoing tornado.

Figure 5.11 shows that simulation determines the bearing angle between the user's location to the destination to be 85.64 degrees, the bearing angle between the user's location to the first tangent line to the effected area to be 50.35 degrees and the bearing angle between the user's location to the second tangent line to be 86.73 degrees. Clearly, the user is in the path of the tornado. The ERSimMon presents the alternative route to the user. Let go ahead and click No for now. We would like to show you the driving into the affected area.



Please make changes to the following configurable settings and click OK to save.

Distance Increment: 50 meters

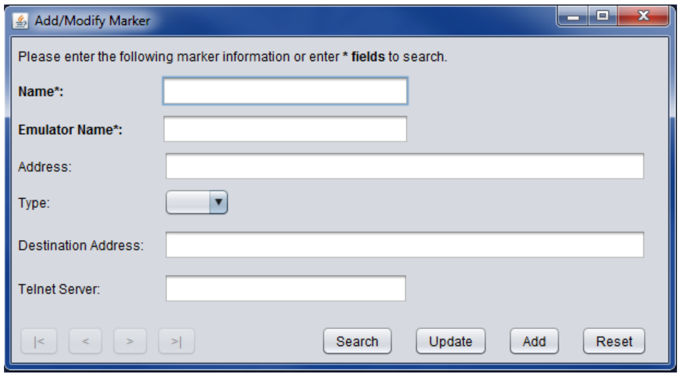
Sleeping Duration: 200 milliseconds

Emulator Location: D:\\devtools\\android-sdk\\tools\\emulator

ADB Location: D:\\devtools\\android-sdk\\platform-tools\\adb

Speed Display: Miles per Hour

Figure 5.7: ERSimMon Settings



Please enter the following marker information or enter * **fields** to search.

Name*:

Emulator Name*:

Address:

Type:

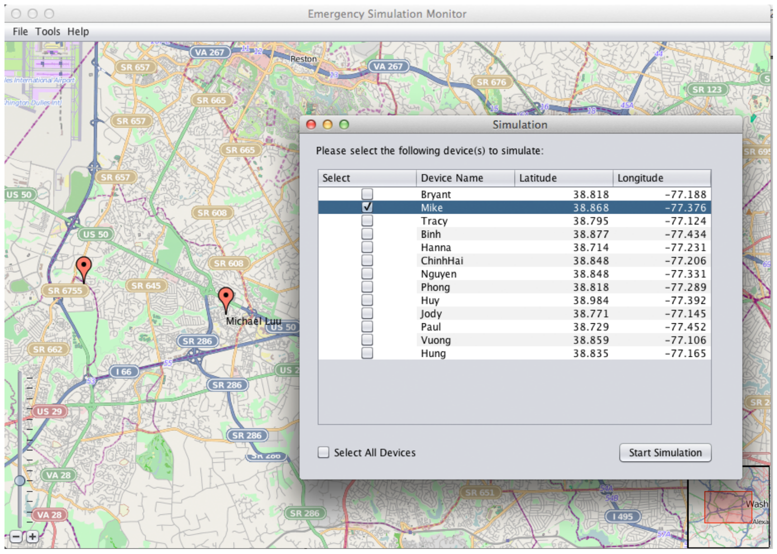
Destination Address:

Telnet Server:

Figure 5.8: ERSimMon Add/Modify Markers

Figure 5.12 shows the user driving at the speed of 55.92 miles per hour into the effected area.

Figure 5.13 shows that upon entering the affected area, the CMAS message is displayed to the user along with the emergency advice.



Please select the following device(s) to simulate:

Select	Device Name	Latitude	Longitude
☐	Bryant	38.818	-77.188
✓	Mike	38.868	-77.376
☐	Tracy	38.795	-77.124
☐	Binh	38.877	-77.434
☐	Hanna	38.714	-77.231
☐	ChinhHai	38.848	-77.206
☐	Nguyen	38.848	-77.331
☐	Phong	38.818	-77.289
☐	Huy	38.984	-77.392
☐	Jody	38.771	-77.145
☐	Paul	38.729	-77.452
☐	Vuong	38.859	-77.106
☐	Hung	38.835	-77.165

Figure 5.9: ERSimMon Simulation Selection

We now shows that if we choose to take the alternative route as suggested by the ERSimMon as shown in Figure 5.11. Figure 5.14 shows that the user is taking a different route, Route 50 instead of Route 66 with the driving speed of 29.08 miles per hour.

Figure 5.15 shows that the alternative route helps the user avoid the affected area of the ongoing tornado.

5.4 Related Works

This section discusses briefly other significant works aimed at improving or providing the mobile users' navigation assistance.

Although we haven't found any publications that are in this research area, there are other related works on the navigation assistance. However, their targets have been for different groups of audiences such as indoor users and blind audiences. One of which is the Guiding Light system [64], which uses projections based augmented reality from the hand-held projectors to provide way-finding information. This system uses a combination

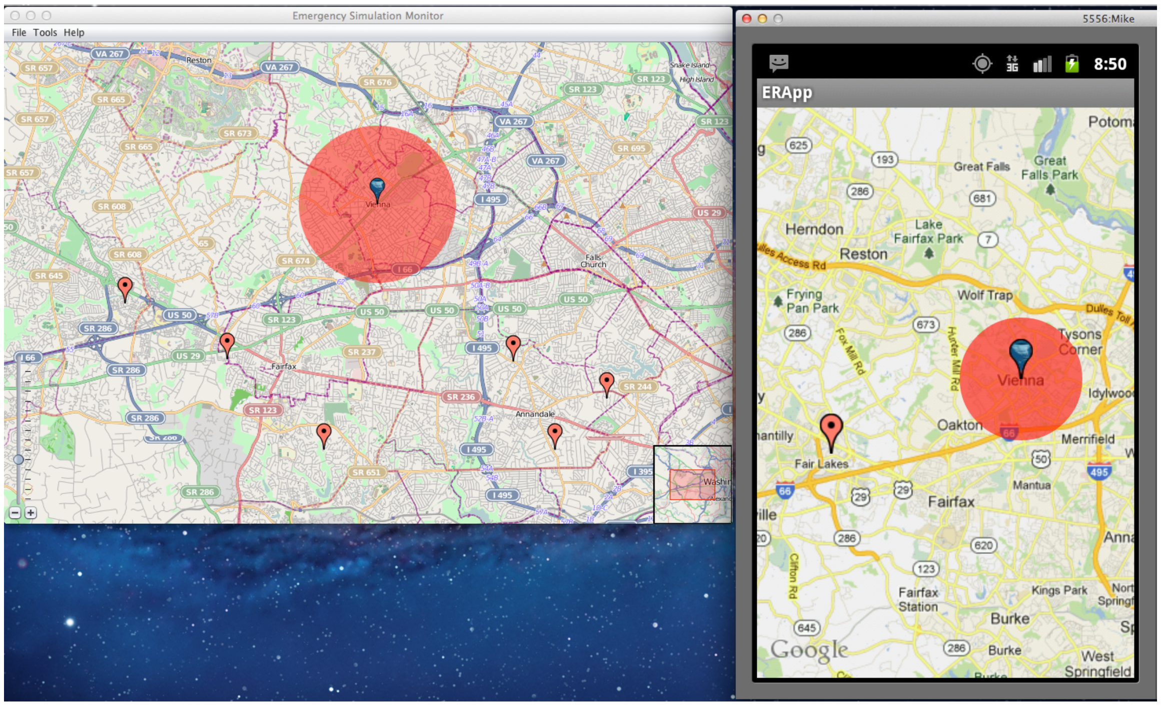
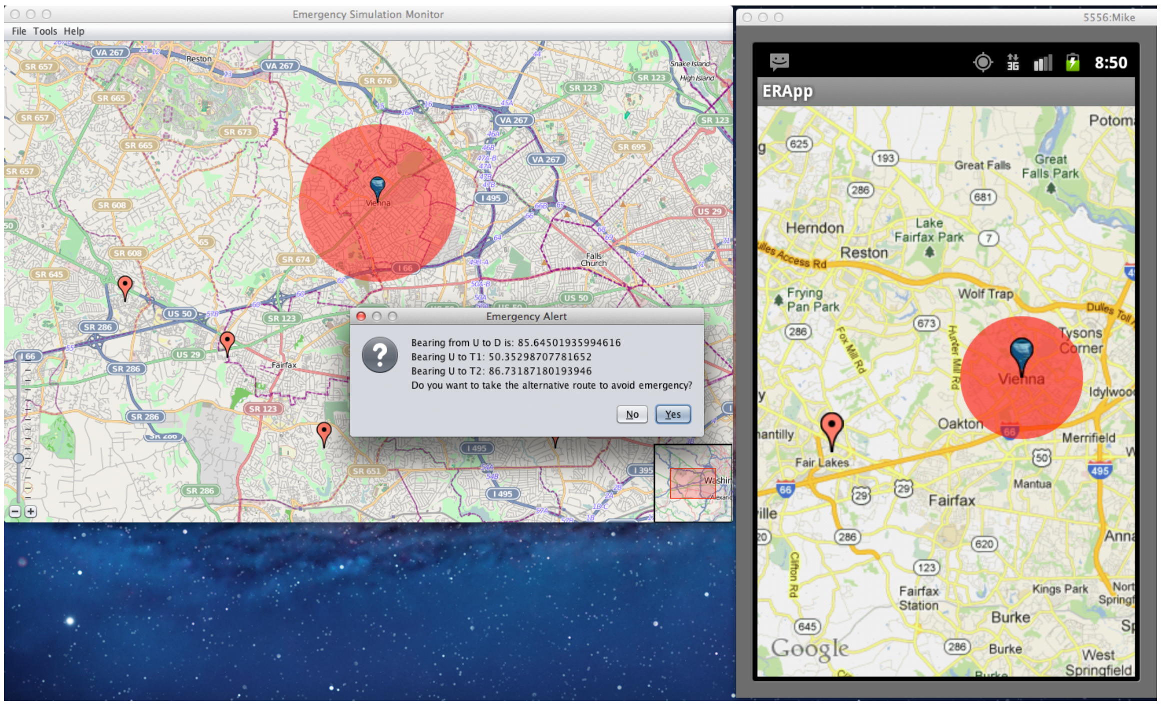


Figure 5.10: Initial ERSimMon Simulation

of hand-held sensors such as proximity, accelerometer, compass, and vision to gather and places information on the surrounding spaces. It then compiles all the reference walls, paths, and other stationary objects in its repository. The system then presents the fast-forward clip of the paths and objects that they will encounter when moving from one place to the other in the building.

The second is the General Framework for a Collaborative Mobile Indoor Navigation Assistance System [65]. This system is to provide a cost-effective method to effectively transfer what the user is seeing to a remote expert who is familiar with the area (e.g., providing museum tours, guiding a lost pedestrian, and providing guided emergency response to an area struck by hurricane), such that interactive assistance can be provided to the local user using augmented reality techniques.

Third, Treuillet et al. [66] presents a new approach for localizing a person by using a



```
Bearing from U to D is: 85.64501935994616
Bearing U to T1: 50.35298707781652
Bearing U to T2: 86.73187180193946
Do you want to take the alternative route to avoid emergency?
```

Figure 5.11: ERSimMon Simulation

single-body-mounted camera and computer vision techniques to guide and maintain the blind person within a navigation corridor less than 1 meter wide along the intended path.

Clearly, the works that have been done so far have addressed the indoor navigation assistance or to the specific groups of users. To the best of our knowledge, there has been little or no research done in this area.

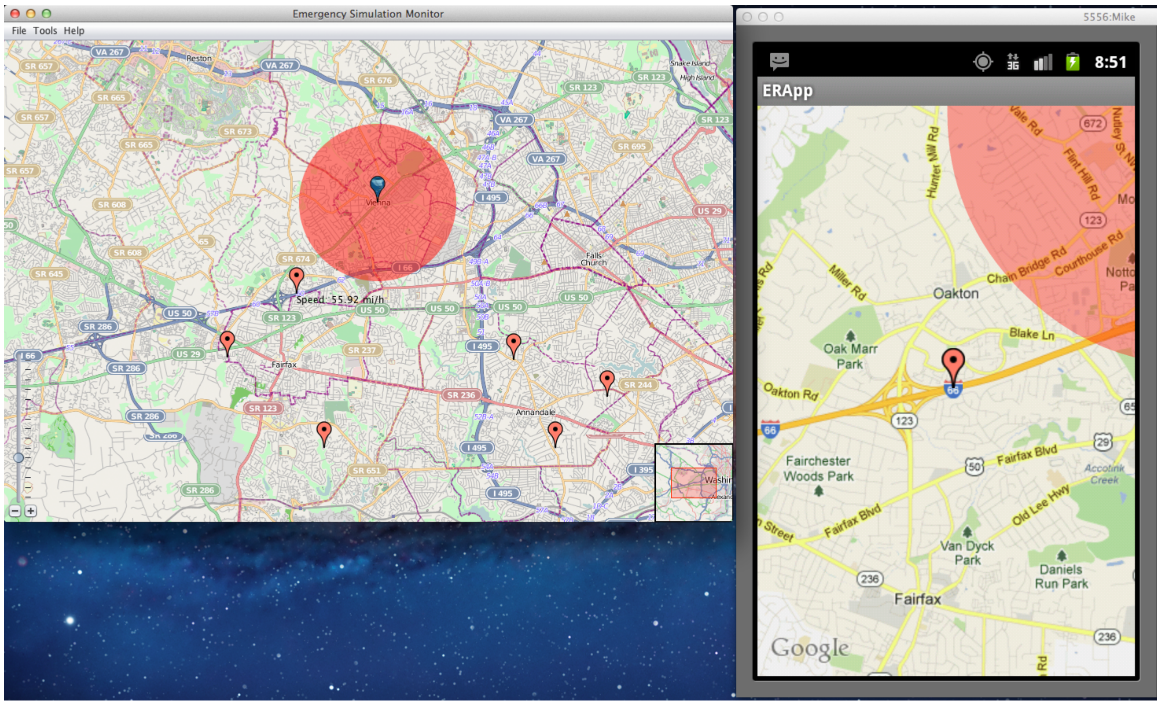
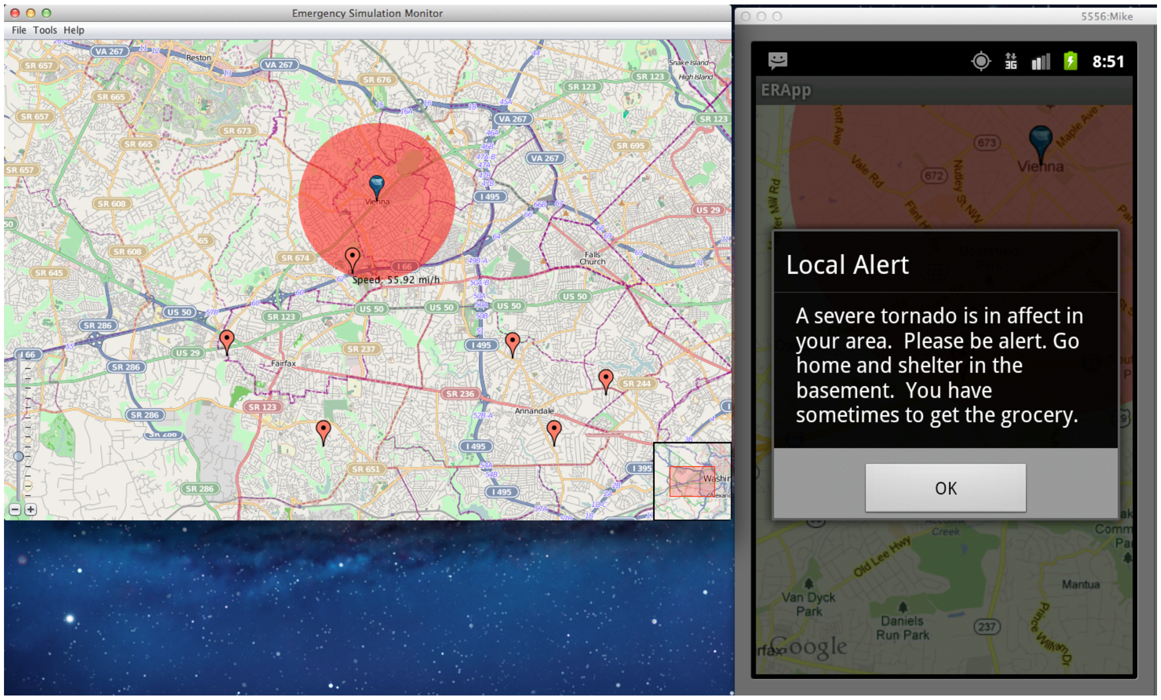


Figure 5.12: Driving toward the Affected Area



A severe tornado is in affect in your area. Please be alert. Go home and shelter in the basement. You have sometimes to get the grocery.

Figure 5.13: Upon entering the Affected Area

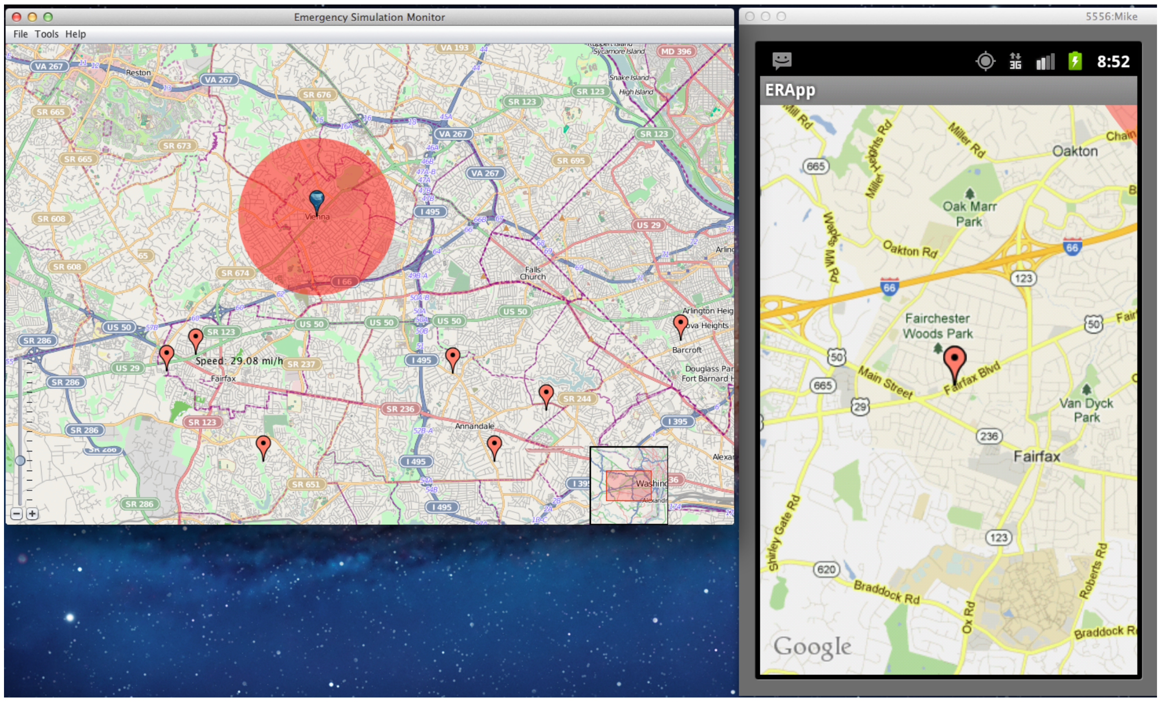


Figure 5.14: Alternative Route

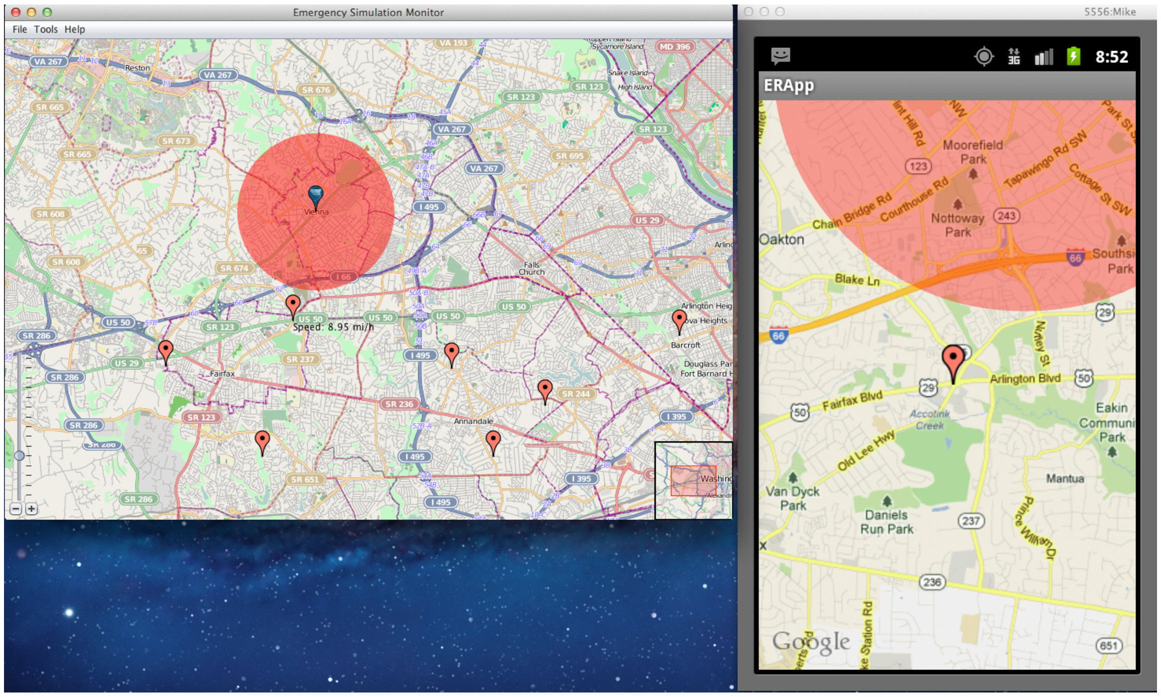


Figure 5.15: Avoiding the Affected Area

Chapter 6: Conclusion

We have taken a collection of standards for emergency management messages and proposed enhancements that would ensure that the messages are delivered to a set of recipients that are capable of responding to the needs at hand. Our proposal is based on a set of attributes that characterize the tasks needed for an external emergency handling entity. We have expressed these attributes by extending the proposed EDXL language. Our objective in doing so was to provide a 911-like pseudo name that is parameterized based on the organization, required responder's role and tasks he is expected to perform in order to satisfy the needs of the call. Our ongoing work addresses translating these pseudo names to addresses available on the telephone, email and pager services so that they can take advantage of PSTN-based and wireless-based priority calling services provided for specified actors of federal, state, local and tribal agencies.

We also investigate the current mechanisms that the federal government used to disseminate the emergency alert and the existing protocol standards to deliver to the public. We have found a couple of significant weaknesses. We proposed an enhancement to CMAS broadcast by implementing a prototype that sends an emergency alert to a confined area for small-scale emergencies. We also proposed an enhancement to the protocol standard to facilitate this communication effectively. We built the ERApp that can be easily installed and configured on the Android mobile phone platform. The ERApp serves as a filter of the broadcast alerts for the affected area and intelligently detects if the users are moving toward the affected area. It will display the alert message with some recommendations of what the users should do in dealing with the prevailing emergency. For future work, we will investigate the existing mobile device capabilities and address how the users would react in dealing with the emergency. In addition, we will extend the ERApp's capabilities by providing intelligent assistance to the users.

In addition, we further investigate different encoding mechanisms that can encode the emergency alert message that gives us more imformation about the nature of the emergency. We also comply with the 90 readable character message rule set in the CMAS specification. We implemented a working prototype of Emergency Response Alert System (ERAlert), which serves as the CMAS alert generation and encoding entity. We also built the ERApp that can be easily installed and configured on the Android mobile phone platform. The ERApp serves as the decoding entity of emergency alerts and displays them in the timely manner. It will provide, along with the alerts, some recommendations of what the users should do in dealing with the prevailing emergency.

Furthermore, we address the availability aspect of emergency responders. We have built an Availability Emergency Response Framework (AERF) that can be easily adopted and further developed into a full-scale system for emergency organizations. We take a step toward ensuring the availability of the personnel in an organization by using the Role-based Access Control (RBAC) model and providing a capability to the organization to provide backup personnel for each of the organizational roles. We have prototyped this AERF framework for a local hospital to ensure the availability of all medical practitioners to respond to any emergency calls at any time. We also built a Timeout feature into our existing mobile application ERApp that can be easily used on Android smart phone devices. The Timeout feature allows users to send their availability status anywhere at anytime to the organization. Upon receiving the unavailable status from a user, the Organizational Monitor Control (OMC) will find a replacement based on the role of the user and the organization's policies.

Lastly, we have addressed the navigation problem during an emergency by building a Navigation Assistance Framework for Emergencies to provide the navigation assistance to mobile users or commuters. The NAF is collecting emergency data from Emergency Sources and disseminating it to all the registered mobile users. When there is a new update to emergency data, the Emergency Source pushes the new information to the NAF, which in turn updates all of its register ERApps via SMS message. ERApp sends a new query to

Google Maps for an alternative route to the destination in order to avoid the emergency path. We also build the ERSimMon to simulate a tornado event and the driving from one place to another to avoid the tornado and its affected area. It also spawns users' emulators in the simulation process to show what is being displayed on the user's ERApp. We suggest interfaces for the Emergency Sources to implement and to send the emergency data to the NAF. All parties, such as Emergency Sources, CMAS, and Mobile Users, have to do their parts to make the Navigation Assistance Framework work the way we design.

Bibliography

Bibliography

[1] NVRC - Emergency Preparedness and Emergency Communication Access *Lessons Learned Since 9/11 and Recommendations*. `http://tap.gallaudet.edu/emergency/nov05conference/EmergencyReports/DHHCANEmergencyReport.pdf`.

[2] Emergency Data Exchange Language (EDXL) Distribution Element, v. 1.0 OASIS Standard EDXL-DE v1.0, 1 May 2006.

[3] Emergency Data Exchange Language Resource Messaging (EDXL-RM) 1.0 OASIS Standard incorporating Approved Errata 22 December 2009.

[4] Common Alerting Protocol (CAP). `http://www.oasis-emergency.org/cap`.

[5] Common Alerting Protocol Version 1.1. `http://docs.oasis-open.org/emergency/cap/v1.1/errata/CAP-v1.1-errata.html`.

[6] Common Alerting Protocol Version 1.2. `http://docs.oasis-open.org/emergency/cap/v1.2/CAP-v1.2-os.html`.

[7] Government Emergency Telecommunications Service. `http://gets.ncs.gov/`.

[8] Wireless Priority Service. `http://wps.ncs.gov/`.

[9] ATIS-0700006, *CMAS via GSM/UMTS Cell Broadcast Service Specification;* March 2010.

[10] XACML. `https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml`

[11] Call Flow Scenarios for Calls Failed. `http://www.cisco.com/en/US/products/sw/iosswrel/ps1831/products_programming_reference_guide_chapter09186a0080087348.html`

[12] Jacqueline Yang, Duminda Wijesekera and Sushil Jajodia, Subject Switching Algorithms for Access Control in Federated Databases, in the proceedings of the 15th Annual IFIP Conference on Database Security, 2002. Pages 61-74.

[13] U. A. Kiser, UPDATE: Utility crews have contained a gas leak near Manassas `http://www2.insidenova.com/isn/news/local/article/gas_leak_forces_neighborhood_evacuations/60363/`

[14] AT&T Wireless. Packages & Deals List - Shop. `http://www.wireless.att.com/cell-phone-service/packages/packages-list.jsp`.

[15] Verizon Wireless. Individual Plans. http://www.verizonwireless.com/b2c/splash/plansingleline.jsp?lid=//global//plans//individual.

[16] Sprint Nextel. Find Popular Cell Phone Packages, Cell Phone Deals. http://nextelonline.nextel.com/NASApp/onlinestore/en/Action/DisplayPackages.

[17] Young "prefer texting to calls", BBC, Jun. 2003.http://news.bbc.co.uk/2/hi/business/2985072.stm.

[18] Commercial Mobile Alert System (CMAS) http://www.fema.gov/commercial-mobile-alert-system

[19] Commercial Mobile Alert Service Architecture and Requirements. http://www.npstc.org/documents/PMG-0035_Final_Recommendations_v0.6.pdf.

[20] American Time Use Survey - 2008 Results. http://www.bls.gov/news.release/pdf/atus.pdf. Bureau of Labor Statistics, United States Department of Labor.

[21] Public Radio Today How American Listens to Public Radio. http://internet.arbitron.com/downloads/PublicRadioToday07.pdf. Arbitron, 2007.

[22] The State of the News Media 2009. http://www.stateofthemedia.org/2009/narrative_audio_audience.php?cat=2&media=10.

[23] Wireless survey: 91% of Americans use cell phones. http://arstechnica.com/telecom/news/2010/03/wireless-survey-91-of-americans-have-cell-phones.ars.

[24] W. Enck, P. Traynor, P. McDaniel, and T.L. Porta. Exploiting open functionality in SMS-capable cellular networks. *Proceedings of the 12th ACM conference on Computer and communications security*, Alexandria, VA, USA: ACM, 2005, pp. 393-404.

[25] Broadcast Control Channel (BCCH). http://en.wikipedia.org/wiki/Broadcast_Control_Channel.

[26] AMBER Alert. http://www.amberalert.gov/.

[27] GSM handover or handoff. http://www.radio-electronics.com/info/cellulartelecomms/gsm_technical/handover-handoff.php.

[28] WBXML. http://www.w3.org/TR/wbxml/.

[29] Prime Power. http://en.wikipedia.org/wiki/Prime_power.

[30] Base64. http://en.wikipedia.org/wiki/Base64.

[31] Parsing XML. http://java.sun.com/developer/Books/xmljava/ch03.pdf.

[32] How to Calculate the size of Encrypted Data. http://www.obviex.com/Articles/CiphertextSize.aspx.

[33] Federal Information Processing Standards Publications. http://www.itl.nist.gov/fipspubs/index.htm.

[34] Emergency Management. `http://www.epa.gov/osweroe1/content/epcra/epcra_plan.htm#info`

[35] Public-key Infrastructure. `http://en.wikipedia.org/wiki/Public-key_infrastructure`

[36] Java Message Service. `http://en.wikipedia.org/wiki/Java_Message_Service`

[37] Min Xu, Duminda Wijesekera, and Xinwen Zhang, *Runtime Administration of an RBAC Profile for XACML*, Services Computing, IEEE Volumn 4(4), Oct 1, 2011.

[38] Min Xu and Duminda Wijesekera, *A Role-based XACML Administration and Delegation Profile and its Enforcement Architecture*, In Proceedings of the 2009 ACM Secure Web Services, Nov 13, 2009, Chicago, IL, USA.

[39] D. Haidar, N. Cuppens-Boulahia, F. Cuppens, and H. Debar, *An Extended RBAC Profile of XACML*, In Proceedings of the 2006 ACM Secure Web Services, Nov 3, 2006, Alexandria, VA, USA.

[40] XACML Profile for Role-based Access Control (RBAC), OASIS. `http://docs.oasis-open.org/xacml/cd-xacml-rbac-profile-01.pdf`

[41] Xiang Li, Gang Liu, Anhong Ling, Jian Zhan, Ning An, Lian Li, and Yongzhong Sha, *Building a Practical Ontology for Emergency Response Systems*, Computer Science and Software Engineering, 2008 International Conference on, 2008, pp. 222-225.

[42] Apache HTTP Server Project. `http://http://httpd.apache.org/`

[43] JBoss Community. `http://www.jboss.org/jbossas`

[44] MySQL. `http://dev.mysql.com/`

[45] Google Calendar API. `https://developers.google.com/google-apps/calendar/`

[46] Xiang Li, Gang Liu, Anhong Ling, Jian Zhan, Ning An, Lian Li, and Yongzhong Sha, Building a Practical Ontology for Emergency Response Systems, Computer Science and Software Engineering, 2008 International Conference on, 2008, pp. 222-225.

[47] Kai Yu, Qingquan Wang and Lili Rong; , "Emergency Ontology construction in emergency decision support system," Service Operations and Logistics, and Informatics, 2008. IEEE/SOLI 2008. IEEE International Conference on , vol.1, no., pp.801-805, 12-15 Oct. 2008.

[48] R.Mizoguchi, M.Ikeda, K. Seta, J.Vanwelkenhuysen, "Ontology for Modeling the World from Problem Solving Perspectives," Proc. of IJCAI-95 Workshop on Basic Ontological Issues in Knowledge Sharing,1995, pp.1-12

[49] A. Malizia, T. Onorati, P. Diaz, I. Aedo, and F. Astorga-Paliza, SEMA4A: An ontology for emergency notification systems accessibility, Expert Systems with Applications, vol. 37, Apr. 2010, pp. 3380-3391.

[50] P. Di Maio, An Open Ontology for Open Source Emergency Response System, `http://opensource.mit.edu/papers/TOWARDS_AN_OPEN_ONTOLOGY_FOR_ER.pdf`

[51] ULEX http://www.lexs.gov/content/ulex

[52] Universal Core (UCore) https://ucore.gov/ucore/

[53] Integrated Public Alert and Warning System (IPAWS). http://www.fema.gov/emergency/ipaws/index.shtm.

[54] PAWS: Subscribing to the LSU Emergency Text Message System. http://grok.lsu.edu/mobile/article.aspx?articleid=4884.

[55] Mason Alert. https://alert.gmu.edu/index.php?CCheck=1.

[56] National Response Framework. http://www.fema.gov/national-response-framework

[57] Tran T, *RFID Based Secure Mobile Communication Framework for Emergency Response Management*, Wireless Communications and Networking Conference, 2010 IEEE.

[58] SIP: Session Initiation Protocol. http://www.ietf.org/rfc/rfc3261.txt

[59] Internet Calendaring and Scheduling Core Object Specification http://tools.ietf.org/html/rfc5545

[60] Out-of-State and Long Commutes: 2011 http://www.census.gov/hhes/commuting/files/2012/ACS-20.pdf

[61] Commuting in the United States: 2009 http://www.census.gov/prod/2011pubs/acs-15.pdf

[62] Poll: Traffic in the United States http://abcnews.go.com/Technology/Traffic/story?id=485098&page=2#.UVDUaBkbqL8

[63] Calculate distance, bearing and more between Latitude/Longitude points http://www.movable-type.co.uk/scripts/latlong.html

[64] J. Chung, I. Kim, and C. Schmandt, Guiding light: navigation assistance system using projection based augmented reality, in Proceedings of the IEEE International Conference on Consumer Electronics (ICCE11), IEEE, 2011, pp. 881882, doi: 10.1109/ICCE.2011.5722917.

[65] Rao, H. and Fu, W.-T. "A General Framework for a Collaborative Mobile Indoor Navigation Assistance System." In Proceedings of the ACM International Conference on Intelligent User Interfaces (IUI), Santa Monica, CA. 2013.

[66] S. Treuillet and E. Royer, "Outdoor/Indoor Vision-Based Localization For Blind Pedestrian Navigation Assistance", International Journal of Image and Graphics Vol. 10, No. 4 (2010) 481496. DOI: 10.1142/S0219467810003937

[67] The Google Directions API https://developers.google.com/maps/documentation/directions/

[68] Tornado Data http://www.srh.noaa.gov/oun/?n=tornadodata-county

[69] Technical realization of the Short Message Service (SMS) `http://www.3gpp.org/ftp/Specs/html-info/23040.htm`

Curriculum Vitae

My name is Paul Ngo. I migrated to the United States in December, 1991 after the escape from Vietnam in 1989. I had lived in Indonesia for two years and in the Philippines for 6 months prior to the United States. I started my High School education at Chantilly HS, Virginia in Feb, 1992 and graduated with Summa Cum Laude in 1994. I then entered Virginia Tech for my undergraduate in August, 1994. I graduated with Cum Laude with dual Bachelor degrees in Computer Science and Applied Mathematics in December, 1997. I began my professional career working for Accenture for two years in Virginia and then moved to San Jose, California to join a start-up company 2Roam, Inc. I then moved back home in Virginia right after the September 11, 2001 and worked as a software engineer consultant for a couple of companies before I joined the Defense Information Systems Agency (DISA) in September, 2002. While working at DISA, I entered George Mason University for a master program in Computer Science in August, 2006 and graduated in May, 2008. I immediately continued with my education pursuing for a doctorate degree in Computer Science. In Feb, 2009, I moved to Department of Homeland Security working as the Next Generation Network (NGN) Security Lead. After the internal reorganization, I am now working as the Enterprise Architecture Lead.