# Navigation Assistance Framework for Emergencies

Paul Ngo
Department of Computer Science
George Mason University
4400 University Drive, MS 4A4
Fairfax, Virginia 22030
Email: pngo1@gmu.edu

Duminda Wijesekera
Department of Computer Science
George Mason University
4400 University Drive, MS 4A4
Fairfax, Virginia 22030
Email: dwijesek@gmu.edu

***Abstract*—Emergencies occur every day at unexpected times and impact our lives in unimaginable ways. In any emergency situation, there are two type of victims: direct victims and indirect victims. Both will have their current plans disrupted in order to deal with the emergency. Federal, State, and Local governments have established a 911 system to assist direct victims. However, there is still lack of assistance provided to the indirect victims. In this paper, we propose a Navigation Assistance Framework that allows emergency organizations to provide emergency information that can assist victims navigating out of the emergency area and reaching their intended destinations in a reasonable amount of time. We develop an emergency prototype ERSimMon to simulate this capability in a small scale to show the effectiveness of the proposed solution. In addition, we develop the Emergency Response Application (ERApp) for a smart phone platform, which intercepts the enhanced Commercial Mobile Alert System (CMAS) broadcast message, displays the user's location with respect to the emergency location on the map and provides navigational assistance and recommend actions to help the user navigate out of ongoing emergencies.**

## I. Introduction

According to the Out-of-State and Long Commutes Survey 2011 [12], 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. Americans spend significant amounts of time, on the average of 25 minutes [13] in their vehicles on the road to go from home to work on a normal working day.

Added to average commute time, local emergencies such as car accidents, road construction, inclement weather, etc. may add extra delays 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 delays are unavoidable. According to a poll conducted by ABCNews on traffic in the United States [14], 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 impending 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.

In 2006, the Federal Government established a Worker Adjustment and Retraining Notification (WARN) Act that supported the research and development of Common Mobile Alert System (CMAS) [15]. The proposed CMAS system utilizes existing commercial telecommunication infrastructures to broadcast emergency alerts and warnings to a specified geographic area. We have extended the usability of CMAS to broadcast alerts to small-scale local emergencies [2]. 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 [1]. Both of these CMAS enhancements allow local emergency information to be broadcast to mobile users more effectively.

To provide relevant navigational 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 possible future works. The NAF acts as a central hub that all relevant information regarding to emergencies flow through and distribute to registered ERApps, which are smart phone applications. This development is aligned with the Dynamic Mobile Application initiative from the US Department of Transportation [20], which can be adapted to cars to alert drivers when approaching work-zones or construction sites [21].

The rest of the paper is organized as follows: section II discusses NAF requirements and some supporting use cases. Section III discusses the NAF design and implementation. Section IV discusses results of our experiments. Section V describes related works and we conclude in section VI.

## II. Navigation Assistance Framework Requirements and Use Cases

In this section, we specify some requirements and objectives that organizations may implement in their processes and operations in order to provide emergency information to other trusted organizations. A requirement contains the word "shall" and is identified by the letters "**R**". An *Objective* is a feature

or function that is desirable, but not mandatory. An Objective contains the words "it is desirable" and is identified by the letters "**O**".

**R#1:** There shall be a way to provide current information about any impending emergency.

**R#2:** There shall be a way to provide directions to avoid the impending emergency.

**O#1:** It is desirable that users provide daily events in their calendars and expose them to the trusted entity in order to provide relevant and immediate actions during time of crisis.

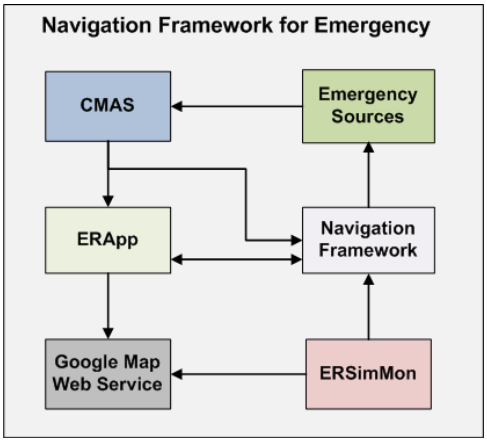


Fig. 1. Navigation Assistance Framework High-Level Components

### A. Use Cases

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

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 an emergency 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 impacted 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 caused by these emergencies can be up to hours. The second and third use cases illustrate that an emergency has occurred in the area. In this case, we illustrate the impending tornado because the tornado is a medium scale emergency and its movement can be tracked by the National Weather Center (NWC) [19]. 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.

## III. Navigation Assistance Framework Architecture and Implementation

We describe major components and the functionality of the Navigation Assistance Framework (NAF) in this section. First, we provide a high-level description of each component and its function in the overall architecture.

### A. Architecture Components

Figure 1 shows the high-level architectural components for the Navigation Assistance Framework. It consists of Emergency Sources, Commercial Mobile Alert System (CMAS) [15], 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 progression of an emergency and to provide updates if needed by other systems. These emergency systems can expose their emergency information as a service. We provide a set of interfaces in table II that can be implemented by Emergency Sources. For example, the National Hurricane Center [18] is considered one of the Sources for emergency information. The Emergency Sources push the most up-to-date emergency data to the CMAS through a web service connection as indicated by the arrow going from the Emergency Sources to CMAS in Figure 1. The CMAS operator will generate the broadcast message based on the emergency data and broadcast it. We have proposed a few enhancements [1], [2] to improve the content of the broadcast message and the area effected by an emergency. The CMAS broadcasts 90-character text messages of emergencies to all mobile devices through ERApp, a mobile emergency application installed on the users' mobile devices (how the ERApp is certified and installed on mobile devices is beyond the scope of this paper). In addition, the CMAS pushes the emergency data to the NAF as indicated by the arrow going from the CMAS to the NAF through a web service connection in Figure 1.

The Navigation Assistance Framework provides a set of interfaces that Emergency Sources need to implement and acts as a listener to the emergency data and advice policies. Whenever needed, the Navigation Assistance Framework pulls the most up-to-date emergency data and advice policies from the Emergency Sources as indicated by the arrow going from the Navigation Framework to Emergency Sources in Figure 1. 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 as indicated by the arrow going both directions from the ERApp to the NAF in Figure 1. The ERApp uses the Google Map Web Services to display a user's

position with respect to the occurring emergency as indicated by the arrow going from the ERApp to Google Map Web Services in Figure 1. The ERApp applies the advice policies to see if the current status of users' behaviors satisfy the conditions on the policy and displays the emergency advice recommended by the policy. For example, the advice policy can say that if the user at rest is 3000 meters away from the center of an emergency, the user needs to consider teleworking for the day. Figure 2 shows such a sample policy written in XACML [17]. XACML policy language answers yes or no to the access control request based on some conditions stated in a policy. Consequently, XACML is not capable of providing emergency advice. Therefore, the NAF uses XACML to evaluate conditions in emergency advice policies given by Emergency Sources before advising users.

In general, these arrows presented in the figure 1 are defined by either push, pull or both push and pull web services. Implementation and the hosting of these web services are beyond the scope of this paper.

```
<?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>
```

Fig. 2. Emergency Advice XACML Policy

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 the list of emergency constraints and road congestion information in order to suggest the best routes that the user can take to reach his destinations. This pulling connection is indicated by the arrow going from the ERSimMon to the Google Map Web Services in Figure 1.

For these components to work 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 are a list of enhancements:

First, we propose an enhancement to the Google Maps API web services [7] 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 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 the directions to avoid emergency road blocks or help navigate around pending emergencies. At best *alternative* parameters provide several routes to the destination, without any guarantee that these routes will avoid emergency road blocks or the pending emergency.

Therefore, our enhancement adds 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: 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.

The NAF doesn't depend on the Google Map enhancement to provide alternative routes in order to avoid the area affected by an emergency. But in this paper, we show how Google could implement this enhancement as we describe below. The NAF can use major routes and intersections as preexisting points and build a directed path using the Shortest Path (in time and distance) algorithm [25] to determine the path to the destination. For every connecting point as a new temporary destination, the NAF uses the algorithm 2 to determine that the route to new temporary destination is out of the affected area.

Second, we enhance the emergency source service to provide 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 using web service.

### B. 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 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 [1], [2] that provides more details about the impending emergency incident. In addition, ERApp is equipped to receive frequent updates about the impending emergency status including tracking information such as GPS location, time, intensity, effected area, etc. This information is necessary for the ERApp to better advice and direct users to their destinations. The goal is to avoid possible road blocks and dangerous areas that are being affected by emergency incidents. We can achieve this goal if we have updated emergency information.

*1) Emergency Data:* Depending on the type of emergency, information may come from different sources. For example, tornado data may come from the National Weather Center. Hurricane data may be retrieved from the National Hurricane Center. Road closures in the local area may come from the local police department or the Department of Transportation. 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 assistance 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 in order 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 data set 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) [8].

For road closures such as a car accident, road construction, water main break, etc., we collect the following emergency data: starting date and time of the closure, and 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 this data is to provide the magnitude of the emergency, 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.

*2) Updating the Directions:* After the Navigation Assistance Framework receives the emergency data from the Emergency Source, it sends the updated data using text messaging to all the devices that have n installed ERApp and registered with the NAF. The ERApp determines the next step. ERApp will formulate a new query to get the updated routes to the destination, given that the impending 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 [11]. The ERApp will use this location address as the destination or allow users to enter their current destinations. 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 updated text message sent from the NAF to ERApps must be agreed upon and interpreted. The format is composed by the list of name-value pairs. The name must be abbreviated by 2 capitalized characters, where character is one byte, and the value must be a primitive type. These names and and associated types must be accessible by the NAF and the ERApp. Table I provides these names, associated types and short descriptions of each attribute.

TABLE I
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 |

According to the SMS Specification [9], an SMS Message has a limitation of 160 readable characters. If we represent this SMS message as readable characters, there may not be enough readable characters to hold values of all these attributes. Therefore, we represent these values as binary values and encode them using the Base 64 Encoding [10] in order to fit within 90 character as described in our previous work [1].

### C. 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 a prototype implementation of these interfaces working together.

*1) Emergency Data Collection:* Figure 1 shows that the NAF receives emergency data from two sources such as CMAS and Emergency Sources (including periodic updates). CMAS sends the CMAS message [1] to the NAF in its broadcast. Two important pieces of data are the GPS location of the impending emergency and the radius of the affected area. NAF uses this information to load the impending emergency details onto the map as shown in Figure 3. The Emergency Sources can push it to the NAF directly. Table II shows NAF interfaces with the Emergency Sources.

In order for NAF to interpret data, we use the following naming conventions for that purpose.

These suggested names and types are the binding agreements between the NAF and Emergency Sources, that we use to retrieve associated values and convert them.

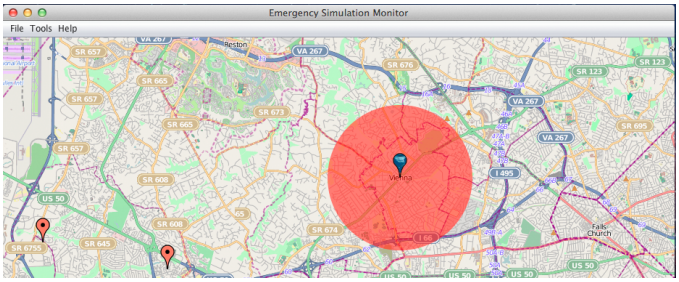


Fig. 3. Impending Tornado

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

TABLE II
INTERFACE FOR IESOURCE IMPLEMENTATION

| 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). |

TABLE III
DATA NAME SPACE

*2) Navigation Update:* The NAF sends an updated text message to all the registered ERApps installed on mobile devices. ERApps will have to decode the message and extract the emergency information.

The ERApp then uses this 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 1 computes the direction that avoids the affected area of the impending 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 is checked against null value. The second parameter *up* is the user profile, which contains the email credentials to access the calendar. The third parameter *calURL* is the Google Calendar web service URL. On lines 1 and 2, the algorithm initializes 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 *cCal*. 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 *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 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 the emergency GPS location on line 23. The URL object is created from the *urlDir* on line 24. The input stream *inputStream* is created

**Algorithm 1** :**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: $cCal \leftarrow newCalendarService()$
4: $cCal.setCreds(up.getWEmail(), up.getWEmailPC())$
5: $calQuery \leftarrow newCalendarQuery(calURL)$
6: $resultEvents \leftarrow cCal.query(calQuery)$
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** $xmlDirDoc$

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.

In this algorithm, we only address the immediate event that requires a user's attention and participation during the emergency time. Any calendar events occurring thereafter are not addressed in this paper.

*3) Determine the Need for Alternative Routes:* This section discusses the algorithm to determine if commuters need to get an alternative route to their destination. If the travel 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 their current position to their destination. We retrieve the location, speed, direction, and the affected area (radius from the emergency location) of the impending emergency from the CMAS message as discussed in section III-C2.

**Algorithm 2** : **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 2 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 [16] *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 4 provides the visual map these locations.

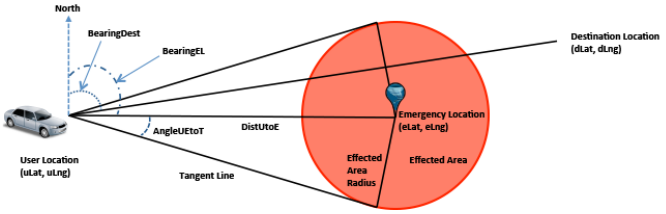


Fig. 4. Alternative Route Decision

*4) Providing Emergency Advice:* The ERApp uses the accelerometer sensor built in the hand-held devices to detect the user's movement. By comparing the (lat, long) acceleration components, the ERApp can estimate if the user is moving or at rest. The ERApp then compares 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 impending emergency. For example, if the user is inside the affected area of a tornado, relevant advice would be to drive to the nearest shelter immediately. ERApp 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.

As described in Figure 2, Emergency Sources sent the emergency advice to the NAF in XACML policies. The ERApp applies these policies to see if the user's behavior status satisfy the conditions on the policy. The ERApp displays the recommended advice to the user.

## IV. Experimentation

Before most, we need to generate a tornado alert informing the all people in the local area that the tornado is coming. In our experiment, we set the emergency location to be in Vienna, Virginia, the radius of the effected area to be 3200 meters from the center of the tornado, the expired time, category, certainty, status, urgency, and severity of the tornado. We broadcast the CMAS message to an emulator. Figure 5 shows the preparation of the tornado alert and the ERApp running on the emulator showing the user's location.

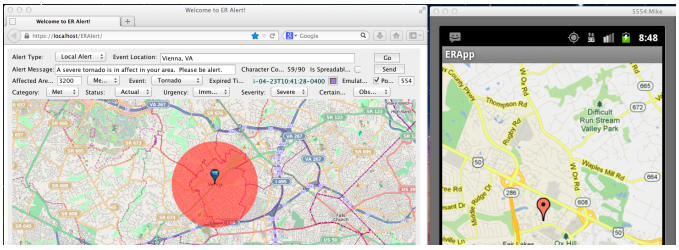


Fig. 5. Prepare the Tornado Alert

The CMAS authority is ready to send the broadcast tornado alert to the emulator by clicking on the Send button. Figure 6 shows the tornado with the effected area in red and the user's location.

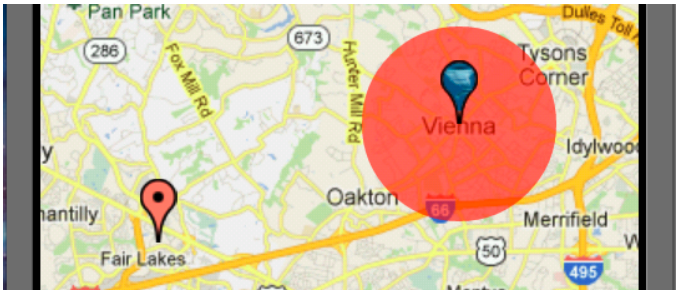


Fig. 6. Tornado Alert on ERApp

We built an Emergency Respond Simulation Monitor (ERSimMon) prototype to simulate an individual driving to work during an ongoing tornado. Figure 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. There are some 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 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.

In addition, the ERSimMon allows 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 nurses can be grouped together under "Nurse" 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 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.

The ERSimMon simulates the driving of the selected user and spawns the emulator for that user, showing the user's location and the ongoing tornado. The ERSimMon 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. Figure 7 shows the user driving at the speed of 55.92 miles per hour into the effected area.

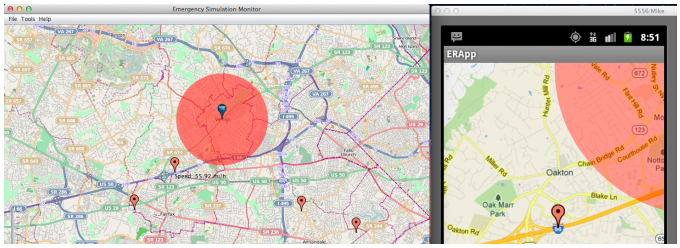


Fig. 7. Driving toward the Affected Area

If we choose to take the alternative route, the user is taking a different route, Route 50 instead of Route 66 with the driving speed of 29.08 miles per hour. Figure 8 shows that the alternative route helps the user avoid the affected area of the ongoing tornado.

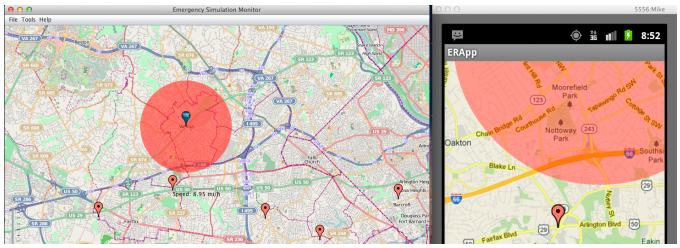


Fig. 8. Avoiding the Affected Area

## V. Related Works

This section briefly discusses 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 [22] that uses projections based augmented reality from the hand-held projectors to provide way-finding information. This system uses a combination 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 [23]. 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.

Treuillet et al. [24] presents a new approach for localizing a person by using a single-body-mounted camera and computer vision techniques to guide and nevigate a blind person within a navigation corridor less than 1 meter wide along the intended path.

Clearly, published works up to now 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 guiding drivers after a CMAS alert is emitted.

## VI. Conclusions

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.

## References


[1] Paul Ngo and Duminda Wijesekera, *Emergency Message In CMAS*, In Proceedings of the International Conference on Critial Infrastructure Protection - Sixth IFIP WG, Mar 2012.

[2] Paul Ngo and Duminda Wijesekera, *Enhancing CMAS Usability*, In Proceedings of the International Conference on Critial Infrastructure Protection - Fifth IFIP WG, Mar 2011.

[3] Paul Ngo and Duminda Wijesekera, *Using Ontological Information to Enhance Responder Availability in Emergency Response*, In Proceedings of the Semantic Technology for Intelligence, Defense, and Security Conference - STIDS, 2010.

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

[5] ATIS-0700007, *Implementation Guidelines and Best Practices for GSM/UMTS Cell Broadcast Service Specification*; October 2009.

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

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

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

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

[10] Base 64 http://en.wikipedia.org/wiki/Base64

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

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

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

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

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

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

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

[18] National Hurricane Center. http://www.nhc.noaa.gov/

[19] National Weather Center. http://nwc.ou.edu/

[20] Dynamic Mobile Application. http://www.its.dot.gov/dma/index.htm

[21] Imagine... http://www.its.dot.gov/imagine.htm#two

[22] 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.

[23] 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.

[24] 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

[25] Shortest Path http://www.cs.princeton.edu/~rs/AlgsDS07/15ShortestPaths.pdf