

CLAERA—Collaborative, Location Aware Emergency Response Application

SERRI Project: Mobile Computing and
Application Development
Initiative

Project Principal Investigator:
Edmon Begoli

This material is based upon work supported by the U.S. Department of Homeland Security under U.S. Department of Energy Interagency Agreement 43WT10301. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. Department of Homeland Security.

SERRI Project: Mobile Computing and Application Development Initiative—
Applications for First Responders

**CLAERA—COLLABORATIVE, LOCATION AWARE
EMERGENCY RESPONSE APPLICATION**

Authors:
Edmon Begoli
Christopher Tomkins-Tinch

Oak Ridge National Laboratory

Date Published:

August 2010

Prepared for
U.S. Department of Homeland Security
under U.S. Department of Energy Interagency Agreement 43WT10301

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, Tennessee 37831-6283
managed by
UT-BATTELLE, LLC
for the
U.S. DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

ACKNOWLEDGEMENTS

We want to thank the SERRI Program for funding this research and development project.

CONTENTS

LIST OF FIGURES	vii
LIST OF TABLES	vii
ACRONYMS	ix
SOUTHEAST REGION RESEARCH INITIATIVE	xi
EXECUTIVE SUMMARY	xiii
1. GENERAL INFORMATION	1
2. PROJECT VISION AND REQUIREMENTS.....	1
2.1 CLAERA Use Cases	2
2.2 Architecturally Significant Requirements	3
2.2.1 Cross-Platform Portability	3
2.2.2 Ease of Use in Conditions of Constrained Input	3
2.2.3 Support for Maps and Global Positioning System.....	4
2.2.4 Intermittent Communication	4
2.2.5 Client-Server Scalability.....	4
3. APPLICATION ARCHITECTURE AND IMPLEMENTATION.....	4
3.1 Application Setup and Configuration.....	5
3.2 Receiving Updates and Maintaining Situational Awareness	5
3.3 Situational Awareness.....	6
3.4 Providing Updates.....	6
3.5 Implementation.....	7
3.6 CLAERA Server Component.....	8
3.6.1 Application Architecture	8
3.6.2 Performance and Scalability of the Server Side Component.....	9
3.7 Performance Optimizations.....	11
3.7.1 Caching Optimizations to Alerts	11
3.7.2 Optimizations to User Statistics Lookup	11
3.7.3 Optimizations to User Location Information.....	11
3.7.4 Performance Improvements Results	11
4. APPLICATION PACKAGING AND DEPLOYMENT.....	13
5. SUMMARY	14
6. REFERENCES.....	14
APPENDIX A. CLAERA WEB SERVICES.....	A-1

LIST OF FIGURES

1	Incident management interaction through CLAERA.....	2
2	CLAERA use cases diagram.....	3
3	CLAERA component architecture.....	5
4	Initial setup.....	5
5	Emergency alert.....	6
6	Map of the event and location of responders.....	6
7	Reporting status.....	7
8	CLAERA on the Android platform (Nexus One simulator).....	7
9	CLAERA on iPhone (field deployment).....	13

LIST OF TABLES

1.	CLAERA Use Cases.....	2
2.	Initial Test Results.....	9
3.	Test Results After Optimization.....	11

ACRONYMS

API	application program interface
CLAERA	Collaborative, Location Aware Emergency Response Application
CSS	Cascading Style Sheets
GPS	global positioning systems
HTTP	hypertext transfer protocol
JSON	JavaScript Object Notation
REST	Representational State Transfer (web services protocol)
SDK	software development kit
SMS	Short Message Service
TCP	Transmission Control Protocol
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language

SOUTHEAST REGION RESEARCH INITIATIVE

In 2006, the U.S. Department of Homeland Security commissioned UT-Battelle at the Oak Ridge National Laboratory (ORNL) to establish and manage a program to develop regional systems and solutions to address homeland security issues that can have national implications. The project, called the Southeast Region Research Initiative (SERRI), is intended to combine science and technology with validated operational approaches to address regionally unique requirements and suggest regional solutions with potential national implications. As a principal activity, SERRI will sponsor university research directed toward important homeland security problems of regional and national interest.

SERRI's regional approach capitalizes on the inherent power resident in the southeastern United States. The project partners, ORNL, the Y-12 National Security Complex, the Savannah River National Laboratory, and a host of regional research universities and industrial partners, are all tightly linked to the full spectrum of regional and national research universities and organizations, thus providing a gateway to cutting-edge science and technology unmatched by any other homeland security organization.

As part of its mission, SERRI supports technology transfer and implementation of innovations based upon SERRI-sponsored research to ensure research results are transitioned to useful products and services available to homeland security responders and practitioners.

For more information on SERRI, go to the SERRI Web site: www.serri.org.

EXECUTIVE SUMMARY

CLAERA—Collaborative, Location Aware Emergency Response Application—is a technology demonstrator that leverages widely available mobile devices to support self-organized, timely, economical, and efficient response to local and national emergencies. CLAERA demonstrates how emergency responders—either professionals or authorized citizens—can use general purpose equipment (e.g., smartphones), enhanced with specialized, cross-platform, portable, and location aware mobile applications, for a more organized response to emergency situations.

CLAERA offers the following benefits to emergency responders and homeland security officials.

- Use of a cross-platform software framework that enables an economical “write once, deploy anywhere” approach to mobile application development.
- Reduction of cognitive overload and communicational congestion at the emergency operations centers.
- Rapid information exchange and situational awareness about emergency situations.
- Use of standard, widely available, and personally owned mobile platforms by first responders and authorized civilians.

1. GENERAL INFORMATION

This project was inspired by the major shift in computing toward multifunctional smartphones. These inexpensive, powerful portable devices have gained massive popularity and provide everyday users with unprecedented access to information and the means of communication.

We recognized the technical, organizational, and economic potential of these devices for national security—and specifically for emergency management needs.

CLAERA was developed to augment and optimize existing emergency management operations and to introduce previously unavailable or economically infeasible situational awareness functionality, such as location aware and peer-to-peer information exchange. CLAERA is a software framework intended for use on low cost consumer mobile devices.

Our framework enables local, state, and federal first responders—as well as authorized civilians—to be active, well-informed, fully integrated, and valuable members of an organized emergency response. Through our application, any emergency responder with a standard location aware smartphone and this special purpose application will be equipped with a mobile emergency information platform, a reconnaissance sensor, and a tool for the local organization of team-based emergency response. CLAERA was developed as a proof-of-concept to demonstrate technologies that enable affordable cross-platform applications for emergency first responders. The project was conceived and executed as an applied research project.

2. PROJECT VISION AND REQUIREMENTS

The purpose of CLAERA is to provide decentralized, scalable, and economical incident awareness for first responders who own mobile phones equipped with global positioning systems (GPSs) and data service. For a mobile phone with CLAERA installed, response becomes a tool for coordinating emergency response and gaining situational awareness.

The CLAERA architecture includes mobile phones as components of a broader information infrastructure (Fig. 1). This infrastructure includes client side and server side components. The mobile application is one such client application and runs on a first responder's personal or government-issued mobile phone. This application displays alerts on a map with descriptions and associations of an incident, sends a user's own location and status as related to the incident, is flexible about if and when to receive alerts, and displays a situational map of other responders and their statuses. The server component is responsible for aggregation of responder status information and amortization of emergency information to involved parties. Given the typical size of an incident (fewer than 100 responders), an instance of the CLAERA server component must support concurrent connections of up to 100 CLAERA clients without visible performance degradation. In addition, CLAERA has to support connections from emergency management coordinators at emergency management centers accessing a CLAERA server for situational and status updates.

CLAERA runs without any modifications and without the rewrite of the client software on any Android 1.3+, Blackberry 4.0+, and iPhone 1.0+ platforms.

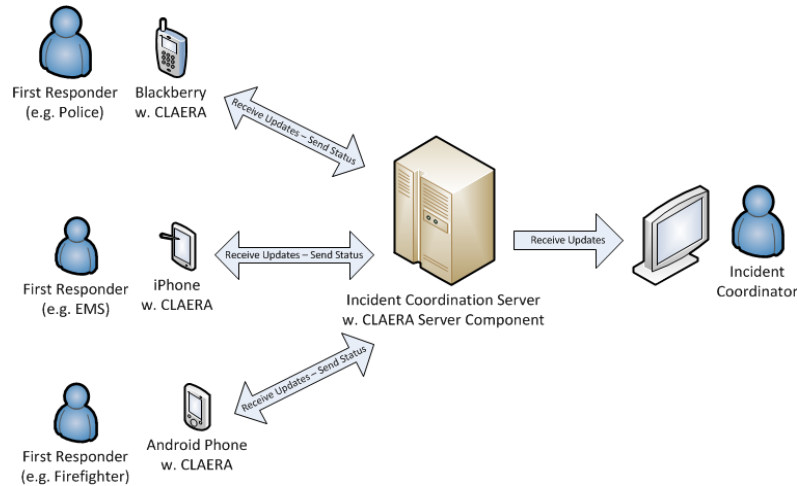


Fig. 1. Incident management interaction through CLAERA.

2.1 CLAERA Use Cases

The following use cases (Table 1 and Fig. 2) capture user-centric functional requirements for the CLAERA application with an outline of interactions between the “actors” (users or machines external to CLAERA) and the CLAERA system as a whole (client and server components).

Table 1. CLAERA Use Cases

ID	Use Case	Description
1	Configure Application	Allows a user to set up preferences such as how frequently he/she will receive alerts (including a choice to receive no alerts at all), the proximity of alerts to be included, and preferences for the type of alerts to be received.
2	Receive Alert	User receives textual and location-specific alerts at his/her device according to the preconfigured preferences. These preferences are stored on a user’s device and are also issued to the CLAERA incident coordination server component.
3	Respond to Alert	User responds to alert by issuing his/her intention to intervene and his/her current location with respect to the incident to the CLAERA incident coordination server component.
4	Send Status and Location Updates	User manually issues updates on his/her status and location and/or application automatically sends the location updates to the CLAERA incident coordination server component.
5	Get Situational Updates	User receives on his/her device situation updates that include locations of other first responders and their statuses related to the incident.

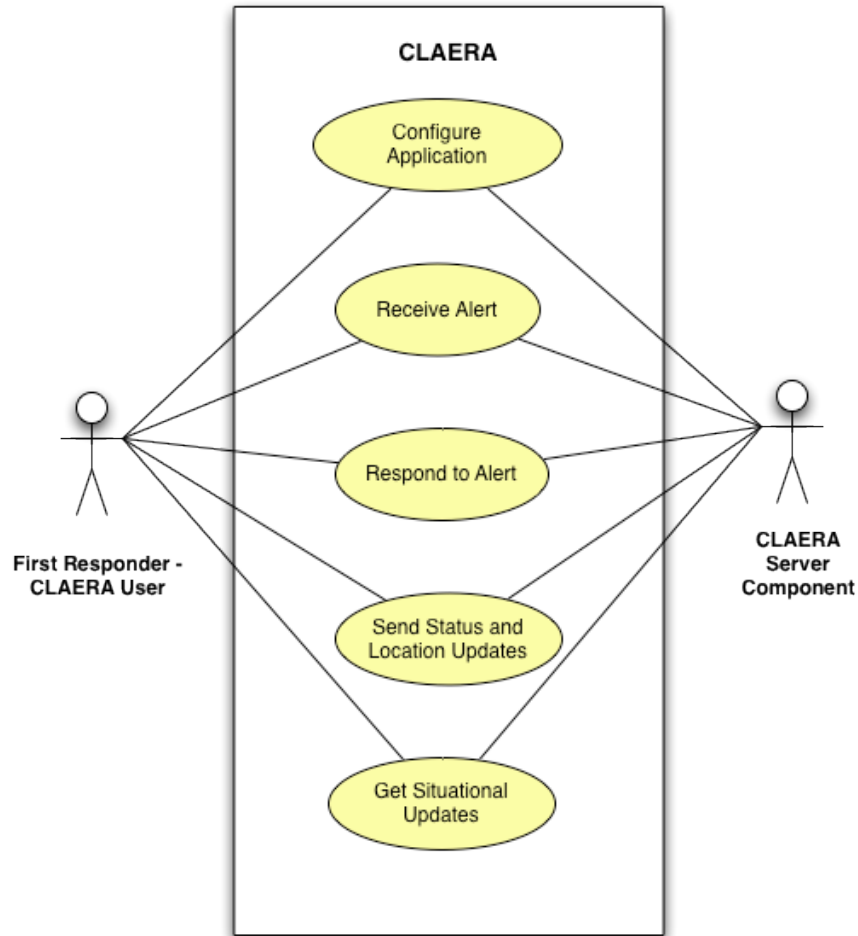


Fig. 2. CLAERA use cases diagram.

2.2 Architecturally Significant Requirements

2.2.1 Cross-Platform Portability

At the time of the design and implementation of the prototype (2009), the Blackberry was the most popular platform for business use, and virtually all federal, state, and local officials had Blackberry devices as business issued phones. The iPhone platform continues to be the most popular smartphone platform for personal use, but still emerging Android based devices are seen by many market research agencies as the upcoming major mobile platform [1]. We therefore established that our solution—to be available on all of the most popular business and personal mobile platforms—had to run on these three platforms, Blackberry, iPhone, and Android, without modification.

2.2.2 Ease of Use in Conditions of Constrained Input

First responders operate in unpredictable and often severely constraining conditions which make use of small form factor devices difficult and cumbersome. For this reason, the design of the user interface of CLAERA mandates minimal user input. Presented

information is clear, easily accessible, and understandable. Specifically, most menus and functions are easily accessible—both in physical appearance and if measured by the “click” distance from the momentary location of the user on the user interface. Minimizing this distance reduces the number and magnitude of visual saccades needed to navigate the user interface and helps to increase efficiency of use [2].

2.2.3 Support for Maps and Global Positioning System

Location aware services—a popular feature of mobile computing that incorporates interactive maps, GPS sensor feeds, and real time location reporting—are also great examples of assistive technologies that reduce cognitive overload of the first responder in situations of high consequences. For that reason, CLAERA features strong integration of location aware services with the incident-specific situational map.

2.2.4 Intermittent Communication

The application supports operations in the degraded state of communication typical of high consequences situations where devices can experience lasting losses of connectivity. The application should be able to maximally leverage available means of connectivity and seamlessly pull and push the information during the windows of opportunity without interruption to the user’s experience with the application.

2.2.5 Client-Server Scalability

CLAERA is a client-server application that, as it is estimated, supports loads of about 30 concurrent clients connecting to a single server and up to 100 at the maximum load. These requirements are based on the estimated and projected numbers of crews at incident sites [3].

3. APPLICATION ARCHITECTURE AND IMPLEMENTATION

The CLAERA application runs as a PhoneGap [4] application either in a foreground fully active mode when the specific alert is being acted upon or in a background mode where the application is only monitoring for alerts (on the iPhone, background mode is available on versions of iOS 4 or greater).

On the server side, CLAERA clients interact with a CLAERA server component that is implemented as a collection of RESTful web services [5]. RESTful web services are implemented in a Java Jersey [6] framework running inside the Apache Tomcat application server and using the open source Postgres database as a relational backend via iBatis [7] persistence framework (Fig. 3).

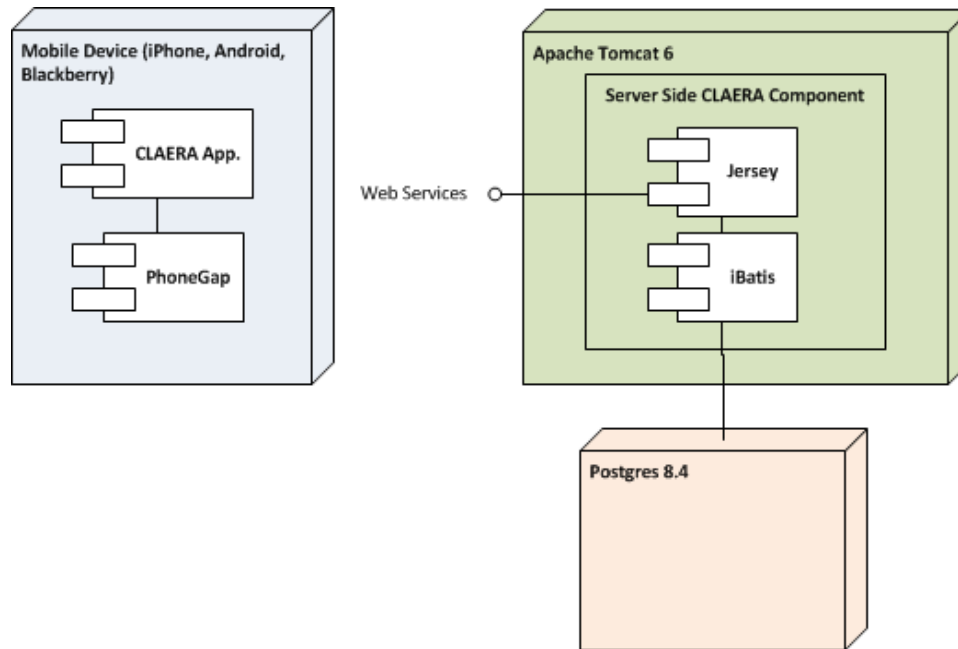


Fig. 3. CLAERA component architecture.

3.1 Application Setup and Configuration

The application is deployed as a PhoneGap-based application. PhoneGap is an open source application development framework based on XHTML-Cascading Style Sheets (CSS) technology. PhoneGap is platform agnostic and supported on all three platforms of interest, Android, iPhone, and Blackberry.

The CLAERA client application is distributed as a PhoneGap project template. Its implementation closely follows the use cases for the application outlined in the requirements section (Sect. 2.2).

One installed user has to initially configure the application for the frequency of updates, the types of updates, and the effective distance from the event as the geographic filter for notifications (Fig. 4).

3.2 Receiving Updates and Maintaining Situational Awareness

Once the application is active, a user will receive any emergency alerts that fall within the boundaries of the user's preferences—geographic

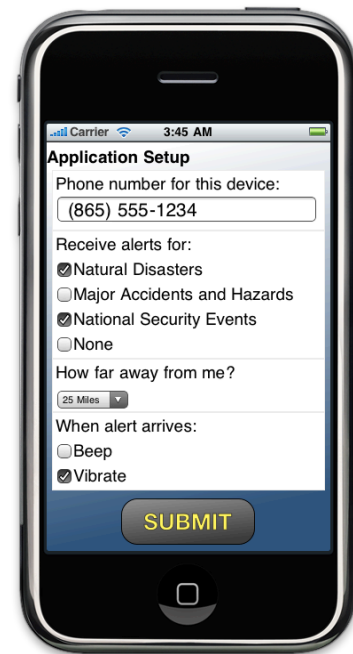


Fig. 4. Initial setup.

proximity and type of emergency. If there is more than one event, the user has an option to page through events (Fig. 5).

3.3 Situational Awareness

An event is projected on the map along with the locations, status, and appropriate icons representing the roles of emergency management crews or individuals responding to the event (police, fire crews, medical emergency crews). This map provides the user of the application with a condensed view, status, and conditions of the emergency response to the emergency event (Fig. 6). This instant situational awareness enables first responders to make the best choices regarding responses to the situation. Note that this map and the associated mechanism are not replacements for the centralized emergency operations center based emergency response mapping process, but serve to augment it.

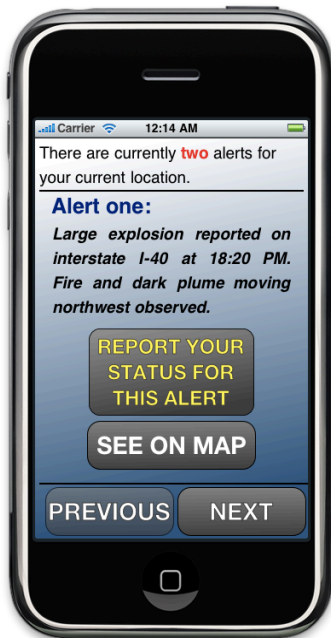


Fig. 5. Emergency alert.

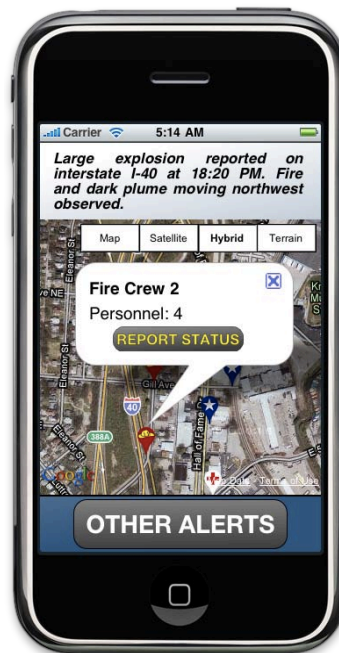


Fig. 6. Map of the event and location of responders.

3.4 Providing Updates

CLAERA supports a two-way information exchange model. An application user may not only receive updates but may also send updates about his or her own situation and the nature of the response related to the emergency event. A set of predefined options makes sending updates very simple and easy to perform, even in deteriorating conditions (Fig. 7). Each update consists of an identifier representing the responder, status, and last known location. CLAERA continuously records data about the last known location, which, in the event of the loss of the GPS signal, will be sent as part of the update. Once the user decides to respond to the emergency CLAERA will automatically send updates of the last known location and the current status.

CLAERA allows the user to provide more detailed updates about his or her status, if needed, and to manually specify the location if, for example, the last automatically recorded location is inaccurate or significantly out of date. Manual entry of location may also be useful for including human-relevant location cues such as landmarks.

3.5 Implementation

One of the architectural requirements for the CLAERA application was for it to be available on as many of the mainstream platforms as possible. This requirement was driven by practical, economic reasons. Making the CLAERA application available on as many platforms as possible would make better use of the existing mobile phones already owned by first responders and therefore reduce the overall cost of the platform deployment.

PhoneGap, an open source project initiated by the Nitobi Software Corporation, was selected as the cross-platform framework used in the development of the CLAERA application.

At the time of this writing, PhoneGap provides a JavaScript application program interface (API) on the iPhone, Android, and BlackBerry platforms for accessing many device-specific native features. Native binary packages are being developed to extend the PhoneGap API to the Palm WebOS, Symbian OS, and Windows Mobile platforms.

On the iPhone, PhoneGap permits access to the device accelerometer, geographic location, user contacts, audio, camera, network, file system, Short Message Service (SMS), compass, telephony, local SQLite database, and vibration functionality.

On the Android platform (Fig. 8), PhoneGap permits access to the device accelerometer, camera, compass, contact list, file system, geographic location, audio, network, and local SQLite database.

On the BlackBerry platform, PhoneGap permits access to the device accelerometer, camera, compass, contact list, file system, geographic location, audio, network, SMS, and telephony.

While the API to access native device features may be common across platforms, physical device attributes may differ. The pixel dimensions of a display may differ among platforms, and deployment may need to account for this if applications are not using native controls. For uniformity of user experience across platforms, CLAERA makes use of common controls. The application was written for a display with horizontal dimension of 320 pixels and vertical

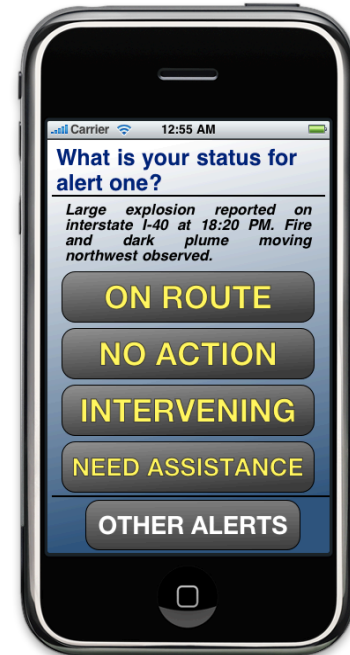


Fig. 7. Reporting status.

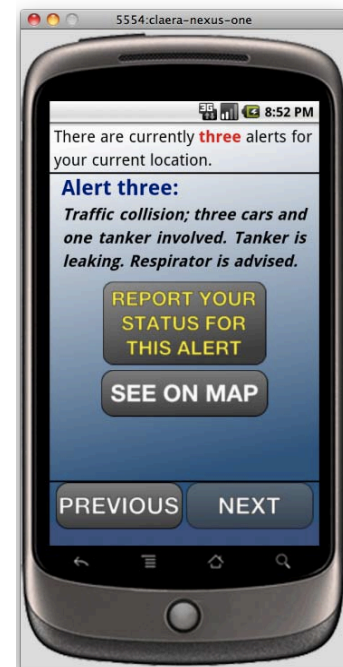


Fig. 8. CLAERA on the Android platform (Nexus One simulator).

dimension of 480 pixels. Adaptations to the CSS used in rendering the application web views permit view elements to adjust in size and spacing to suit displays of differing dimensions.

CLAERA has been successfully deployed to the iPhone and Android platforms. In each case, PhoneGap provides a platform-specific application directory structure and files used in creating the hardware JavaScript API.

Initially, the CLAERA application package was developed, compiled, and deployed using Apple XCode software and the iPhone software development kit (SDK). The web view subdirectory of the XCode project, containing all of the CLAERA-specific application code, was then copied to an equivalent web view directory in a PhoneGap Android project file. The Android project file was built using the Android SDK and the Android Development Tools plug-in for the Eclipse integrated development environment.

Application debugging was performed using platform-specific emulators.

Applications written with web views can be developed more quickly than those written with native code and can be easily adapted to other platforms; however, such applications are not without caveats. As application code is written in JavaScript, compilation to machine code does not occur. Application code is interpreted by the mobile web browser engine and executes more slowly than native code.

Before deployment, application code can be optimized for mobile deployment. This is typically performed by automated tools and often involves removing code comments, shortening variable and function names, and removing unnecessary whitespace. Further, to reduce network overhead, data requested from a remote server may be compressed using gzip or other algorithms before transmission.

3.6 CLAERA Server Component

3.6.1 Application Architecture

The server side application, CLAERA Server Component, is implemented as a Java web application. It exposes its services to the CLAERA client as a set of “RESTful” web services using Jersey framework. The web services accept data in XML or JavaScript Object Notation (JSON) format and can return data as XML, JSON, or JSONP (JSON “with padding”). The Jersey framework handles the mapping from any of the formats into custom, internally defined object models specific to CLAERA. Data are stored and retrieved from the database via Apache iBatis to perform the object to SQL mapping.

CLAERA web services correspond to CLAERA’s use cases. Details of each web service are provided in Appendix A.

The CLAERA Server Component is deployed into Apache Tomcat 6.0.20 running on Java 1.6.0_16. The physical server is a Dell server with dual 2.5 GHz Intel Xeon processors and 4 GB of RAM running the Windows Server 2008 Standard 64-bit operating system. HTTP traffic is routed to Tomcat through Apache HTTP Server 2.2 via mod_proxy. Data are persisted in a PostgreSQL 8.4 database running on the same machine.

3.6.2 Performance and Scalability of the Server Side Component

The following sections describe quality assurance and optimization tests to ensure the satisfactory performance of the CLAERA server side component and the overall application and the initial results of testing.

3.6.2.1 Test setup

Performance tests were created and executed using Apache JMeter [8].

The tests exercised and examined performance, reliability, and robustness of CLAERA web services by invoking the following web services.

1. Get active alerts within range—checks if there are any alerts in the user’s area of interest.
2. Set user status—enables CLAERA user to respond to the alert with his/her current status related to the emergency event (e.g., “on route,” “no action,” “intervening”).
3. Set user location (executed 10 times)—enables tracking of the responding first responder/CLAERA user.
4. Get user statistics for alert (executed 10 times)—enables CLAERA user to see the locations and status of other responders.

The configuration for each test was as follows.

- Ten concurrent, simulated users executing each web service once.
- Ten concurrent, simulated users executing each web service 10 times consecutively.
- One hundred concurrent, simulated users executing each web service 10 times consecutively.

The maximal number of 100 concurrent users was chosen to satisfy the scalability requirement of a maximum of 100 concurrent users interacting with the application per incident.

3.6.2.2 Initial test results

The initial results of the test are presented in Table 2.

Table 2. Initial Test Results

Operation	# Samples	Avg.	Min.	Max.	Std. Dev.	Error (%)	Throughput (ops./s)	KB/s	Avg. Bytes
Get Alerts within range	10	38	16	65	16.11	0	10.1	1.64	167
Set User Status	10	41	9	64	19.23	0	9.8	0	0
Set User Location	100	39	11	77	15.87	0	55.7	0	0
User Statistics for Alert	100	39	9	71	15.47	0	55.7	12.08	222
TOTAL	220	39	9	77	15.88	0	118.5	12.55	108.5

Users 10
 # Iterations 10

Operation	# Samples	Avg.	Min.	Max.	Std. Dev.	Error (%)	Throughput (ops./s)	KB/s	Avg. Bytes
Get Alerts within range	100	72	12	184	25.72	0	6	0.98	167
Set User Status	100	73	14	153	23.51	0	6	0	0
Set User Location	1,000	76	10	220	25.15	0	57.5	0	0
User Statistics for Alert	1,000	74	12	216	24.32	0	57.6	12.48	222
TOTAL	2,200	74	10	220	24.77	0	126.1	13.36	108.5

Users 100
 # Iterations 1

Operation	# Samples	Avg.	Min.	Max.	Std. Dev.	Error (%)	Throughput (ops./s)	KB/s	Avg. Bytes
Get Alerts within range	100	2,014	14	4,890	1,434.79	0	18.6	3.03	167
Set User Status	100	1,633	23	9,819	1,370.12	0	8.6	0	0
Set User Location	1,000	1,141	32	4,178	574.18	0	34.7	0	0
User Statistics for Alert	1,000	1,088	18	5,887	505.94	0	34.9	7.57	222
TOTAL	2,200	1,179	14	9,819	700.46	0	76.1	9.06	108.5

Users 100
 # Iterations 10

Operation	# Samples	Avg.	Min.	Max.	Std. Dev.	Error (%)	Throughput (ops./s)	KB/s	Avg. Bytes
Get Alerts within range	1,000	1,573	10	188,996	7,298.93	0.1	2.6	0.43	168.4
Set User Status	1,000	949	10	46,128	1,998.63	0	2.6	0	0
Set User Location	10,000	999	8	189,005	4,397.17	0.03	26	0.01	0.5
User Statistics for Alert	10,000	1,011	7	189,006	4,990.81	0.04	26.1	5.66	222.5
TOTAL	22,000	1,028	7	189,006	4,771.2	0.04	57.2	6.09	109

While initial results indicated satisfactory performance for the specified load, we determined—based on the traffic patterns and the database persistence interactions—that we could further improve the performance.

Point-to-point monitoring indicated that the database was being accessed for each web service request to serve a small proportion of data relative to the overall network traffic. Many of these requests were just polling inquiries to the server from the clients (e.g., queries as to whether there were any alerts in the user’s area or, if the user was responding to an event, queries as to the locations of others responding to the same event). Based on this realization about the client-server interaction, we decided to cache some of the requested, seldom changing, objects into memory, when possible, to reduce database traffic. We estimated that this approach would relieve some significant fraction of the database disk

access requests and reduce the TCP based inter-process communication between the application and the database.

3.7 Performance Optimizations

3.7.1 Caching Optimizations to Alerts

All active alerts are cached in memory. When the application starts, all the active alerts are loaded from the database. As new alerts are created, the memory cache is updated and the new alerts are added to the database. As alerts are deactivated, they are removed from the memory cache and deactivated in the database. When the client polls for active alerts within range, the server application accepts the client location and radius as the input parameter and searches through the active alert cache for matching active results and, if any are found, returns the results to the client.

3.7.2 Optimizations to User Statistics Lookup

User statistics for active alerts are also cached in memory. When the application starts, all the user statistics are loaded from the database into memory. As users submit updated status, the server application updates both the cache and the database. This allows for the establishment of the fast, in-memory lookup list of who is responding to each active alert and what his/her status is.

3.7.3 Optimizations to User Location Information

A user’s most recent location is cached in memory. This information is loaded into memory from the database at the start-up time. As users submit their current locations, the application replaces this record in the cache with the most up-to-date versions, and it updates the record on the database.

When the client application queries the current user statistics for a given alert, the server application searches the cached information for other responding users and their status respective to the incident and, if results are found, provides this information to the client application. This process is accomplished without ever accessing the database.

3.7.4 Performance Improvements Results

Table 2 displays the results of the same test as presented in Table 1 after optimization. Observe the significant gains in overall transactional bandwidth and speedup in response times.

Table 3. Test Results After Optimization

Operation	# Samples	Avg.	Min.	Max.	Std. Dev.	Error (%)	Throughput (ops./s)	KB/s	Avg. Bytes
# Users		10							
# Iterations		1							
Get Alerts within range	10	47	15	121	29.26	0	10	3.18	327
Set User Status	10	62	35	100	20.66	0	9.8	0	0

Operation	# Samples	Avg.	Min.	Max.	Std. Dev.	Error (%)	Throughput (ops./s)	KB/s	Avg. Bytes
Set User Location	100	60	27	153	22.53	0	49.9	0	0
User Statistics for Alert	100	51	12	136	23.07	0	50.2	10.88	222
TOTAL	220	55	12	153	23.51	0	101.7	11.49	115.8

Users 10
 # Iterations 10

Operation	# Samples	Avg.	Min.	Max.	Std. Dev.	Error (%)	Throughput (ops./s)	KB/s	Avg. Bytes
Get Alerts within range	100	119	15	3,170	310.35	0	3.88	1.24	327
Set User Status	100	131	29	3,054	297.74	0	3.88	0	0
Set User Location	1,000	115	20	3,115	195.64	0	37.95	0	0
User Statistics for Alert	1,000	99	12	3,054	170.13	0	37.98	8.23	222
TOTAL	2,200	109	12	3,170	197.60	0	83.09	9.39	115.77

Users 100
 # Iterations 1

Operation	# Samples	Avg.	Min.	Max.	Std. Dev.	Error (%)	Throughput (ops./s)	KB/s	Avg. Bytes
Get Alerts within range	100	136	6	3,107	517.98	0	29.50	9.42	327
Set User Status	100	549	23	2,974	665.59	0	27.37	0	0
Set User Location	1,000	376	14	3,313	589.06	0	83.27	0	0
User Statistics for Alert	1,000	185	8	9,019	594.30	0	83.32	18.06	222
TOTAL	2,200	286	6	9,019	602.57	0	182.12	20.59	115.77

Users 100
 # Iterations 10

Operation	# Samples	Avg.	Min.	Max.	Std. Dev.	Error (%)	Throughput (ops./s)	KB/s	Avg. Bytes
Get Alerts within range	1,000	200	8	9,092	694.33	0	12.381	3.95	327
Set User Status	1,000	396	12	9,148	681.18	0	12.38	0	0
Set User Location	10,000	451	11	45,437	1,183.51	0	123.32	0	0
User Statistics for Alert	10,000	203	7	9,553	682.36	0	123.32	26.74	222
TOTAL	22,000	325	7	45,437	951.97	0	270.97	30.64	115.77

4. APPLICATION PACKAGING AND DEPLOYMENT

Although the scope of this project was limited to applied research and development, we envision that this application could be developed and deployed at the larger scale and available for download through fusion centers, emergency operations centers, or dedicated application stores. We expect that emergency responders, depending on their role and the their organizational circumstances, would download the application on their own or through their organizations (Fig. 9), but that the application would get activated only upon receipt of the activation code from the server and that it would remain active only while resident on a registered device.

The mobile application was developed using platform-agnostic code and can be easily adapted for deployment on any of the mobile operating systems supported by the PhoneGap framework.

For deployment on a particular mobile platform, the application code is packaged within a project template supplied by PhoneGap. This template is specific to a mobile operating system and can be built into an executable binary program compatible with a particular device. Once executable binaries have been compiled, platform-specific deployment enterprise procedures may be followed.

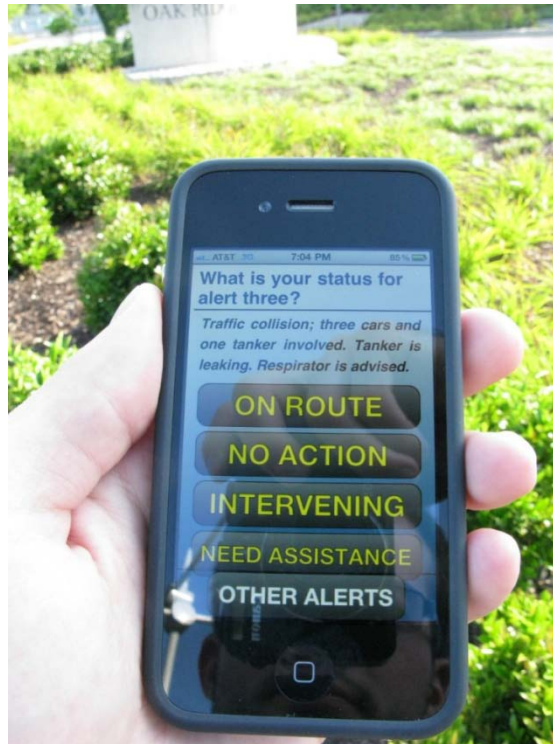


Fig. 9. CLAERA on iPhone (field deployment).

5. SUMMARY

Mobile applications; smartphones; location based services; and social, self-establishing networks are tomorrow's technologies. However, these concepts and applications have been readily embraced by the general population, and this is only expected to grow with the launch of faster, more user friendly devices and more efficient networks.

Our simple, flexible, and broad framework proposes a solution that is, while highly needed, novel and not yet uniformly available to this group of users. Our solution will enable members of the emergency response community to create a wide ranging, well connected, and well organized system capable of responding to a wide array of emergencies.

Furthermore, we hope that our framework will inspire federal authorities and commercial entities to see wireless networks and mobile devices as strategic national infrastructure and invest in their reach and resiliency.

6. REFERENCES

1. John Cox, "Forrester: Google Android smartphones to take 10% of market in 2010," *Network World*, December 24, 2009.
2. J. H. Goldberg and X. P. Kotval. "Computer interface evaluation using eye movements: methods and constructs," *International Journal of Industrial Ergonomics*, **24**(6), pp. 631–645, October 1999.
3. Federal Emergency Management Agency, Emergency Management Institute. IS-703.a NIMS Resource Management Course, December 2009.
4. PhoneGap Application Development Framework, Nitobi Software Co., December 2009, <http://www.phonegap.com>.
5. Sameer Tyagi. RESTful Web Services, August 2006, Sun Developer Network, <http://java.sun.com/developer/technicalArticles/WebServices/restful>.
6. Jersey REST Framework, Version 1.3, Project GlassFish, December 2009 <https://jersey.dev.java.net>.
7. Apache Software Foundation. Apache iBatis, Version 2, December 2009 <http://ibatis.apache.org>.
8. Apache Software Foundation. Apache Jakarta JMeter, Version 2.4, Dec. 2009, <http://jakarta.apache.org/jmeter>.

APPENDIX A. CLAERA WEB SERVICES

This is a list of web services that CLAERA mobile clients invoke remotely on the CLAERA Server Component to send or get situational updates.

1. Get Active Alerts Within Range

The idea is to get a list of active alerts based on your current location.

URL:

[http://www.\[hidden\].gov/rest/alerts](http://www.[hidden].gov/rest/alerts)

Query Parameters:

format - Return format {xml, json, jsonp}

callback - Callback name if using jsonp

lat - Users current latitude

long - Users current longitude

radius - integer distance to search for alerts in your area (units in miles)

type - (optional) filter on alert type

HTTP Method:

GET

Output Formats:

application/json, application/xml, application/x-javascript

Sample Request:

[http://www.\[hidden\].gov/rest/alerts?format=jsonp&lat=33.567&long=-84.789&radius=30](http://www.[hidden].gov/rest/alerts?format=jsonp&lat=33.567&long=-84.789&radius=30)

Sample Response:

```

{
  "alert": [
    {
      "active": "true",
      "description": "This is the alert description.",
      "id": "12345",
      "latitude": "36.02",
      "longitude": "-84.23",
      "name": "Doomsday",
      "timestamp": "2009-11-11T13:53:55.470-05:00",
      "types": ["FIRE", "POLICE"]
    },
    {
      "active": "true",
      "description": "We are just testing.",
      "id": "456",
      "latitude": "34.34",
      "longitude": "134.34",
      "name": "Nuke Test",
      "timestamp": "2009-11-11T13:53:55.470-05:00",
      "types": ["FIRE", "POLICE"]
    }
  ]
}

```

2. Get All Active Alerts

The idea is to get a list of all active alerts.

URL:

[http://www.\[hidden\].gov/rest/alerts/allactive](http://www.[hidden].gov/rest/alerts/allactive)

Query Parameters:

format - Return format {xml, json, jsonp}

callback - Callback name if using jsonp

HTTP Method:

GET

Output Formats:

application/json, application/xml, application/x-javascript

Sample Request:

http:// www.[hidden].gov/rest/alerts/allactive?format=json

Sample Response:

```
{
  "alert": [
    {
      "active": "true",
      "description": "This is the alert.",
      "id": "12345",
      "latitude": "36.02",
      "longitude": "-84.23",
      "name": "Doomsday",
      "timestamp": "2009-11-11T13:53:55.470-05:00",
      "types": ["FIRE", "POLICE"]
    },
    {
      "active": "true",
      "description": "We are just testing.",
      "id": "456",
      "latitude": "34.34",
      "longitude": "134.34",
      "name": "Nuke Test",
      "timestamp": "2009-11-11T13:53:55.470-05:00",
      "types": ["FIRE", "POLICE"]
    }
  ]
}
```

3. Get Alert By Id

The idea is to get an alert object given an alert id.

URL:

http:// www.[hidden].gov/rest/alerts/ids/{alertId}

Path Parameters:

alertId - Id of the alert

Query Parameters:

format - Return format {xml, json, jsonp}

callback - Callback name if using jsonp

HTTP Method:

GET

Output Formats:

application/json, application/xml, application/x-javascript

Sample Request:

http:// www.[hidden].gov/rest/alerts/ids/456?format=json

Sample Response:

```
{
  "active": "true",
  "description": "This is the alert description.",
  "id": "456",
  "latitude": "36.02",
  "longitude": "-84.23",
  "name": "Doomsday",
  "timestamp": "2009-11-11T13:57:39.284-05:00",
  "types": ["FIRE", "POLICE"]
}
```

4. Insert Alert

Publish an alert.

URL:

http:// www.[hidden].gov /rest/alerts

HTTP Method:

POST

Input Formats:

application/json, application/xml

Sample Request:

http:// www.[hidden].gov /rest/alerts

And posting the following data

```
{
  "name": "test3,"
  "description": "This is a test3.",
  "latitude": "-84.66,"
  "longitude": "32.66",
  "timestamp": "2009-11-13T10:35:46.091-05:00,"
  "types": ["FIRE," "POLICE"]
}
```

Sample Response:

```
Response HTTP/1.1 201 Created
location header set Location: With
http://www.[hidden].gov/rest/alerts/ids/10
```

5. Delete Alert

Actually marks an alert as inactive; maybe we'll do an actual delete in the future.

URL:

http:// www.[hidden].gov /rest/alerts/ids/{alertId}

HTTP Method:

DELETE

Path Parameters:

alertId - Id of the alert

Sample Request:

http:// www.[hidden].gov/rest/alerts/ids/567

Sample Response:

```
Response HTTP/1.1 200
```

6. Get Current User Statistics For an Alert

The idea is to get a list of users responding to a given alert along with their current status and location.

URL:

http:// www.[hidden].gov /rest/alerts/userstatii/{alertId}

Path Parameters:

alertId - Id of the alert

Query Parameters:

format - Return format {xml, json, jsonp}

callback - Callback name if using jsonp

HTTP Method:

GET

Output Formats:

application/json, application/xml, application/x-javascript

Sample Request:

http:// www.[hidden].gov /rest/alerts/userstatii/1?format=json

Sample Response:

```
{
  "userStatus": [
    {
      "id": "18,"
      "alertId": "1,"
      "[obfuscated],"
      "statusTime": "2009-12-02T09:04:21.063-05:00,"
      "message": "On my way,"
      "latitude": "33.87765,"
      "longitude": "-84.3222"},
    {
      "id": "19,"
      "alertId": "1,"
      "[obfuscated],"
      "statusTime": "2009-12-02T09:13:26.478-05:00,"
      "message": "Almost there,"
      "latitude": "33.87799,"
      "longitude": "-84.3111"
    }
  ]
}
```

7. Get User (Config) By Id

The idea is to get a user object given their id (phoneNumber).

URL:

http:// www.[hidden].gov /rest/users/ids/{phoneNum}

Path Parameters:

phoneNum - phone number of the user

Query Parameters:

format - Return format {xml, json, jsonp}

callback - Callback name if using jsonp

HTTP Method:

GET

Output Formats:

application/json, application/xml, application/x-javascript

Sample Request:

http:// www.[hidden].gov/rest/users/ids/[obfuscated]?format=json

Sample Response:

```
{
  "type": "FIRE,"
  "distance": "25,"
  "phoneNumber": "[obfuscated],"
  "alertType": "all,"
  "notificationStyle": "RING"
}
```

8. Insert/Update User Config

Insert a user or update their configuration.

URL:

http:// www.[hidden].gov/rest/users

HTTP Method:

POST

Input Formats:

application/json, application/xml

Sample Request:

http:// www.[hidden].gov/rest/users

And posting the following data

```
{
  "phoneNum": "[obfuscated]",
  "alertType": "all",
  "notificationStyle": "RING",
  "type": "FIRE",
  "distance": "25"
}
```

Sample Response:

Response HTTP/1.1 200 OK

9. Submit User Status

User can submit his/her current status in responding to an alert. [alertId, phoneNum, statusTime]; must be unique.

URL:

http:// www.[hidden].gov/rest/userstatus

HTTP Method:

POST

Input Formats:

application/json, application/xml

Sample Request:

http:// www.[hidden].gov/rest/userstatus

And posting the following data

```
{
  "phoneNum": "[obfuscated]",
  "statusTime": "2009-11-20T09:58:22.924-05:00",
  "latitude": "33.2456",
  "longitude": "-84.78878",
  "alertId": "1",
  "status": "In Route",
  "message": "I'll be there soon",
  "locationText": ""
}
```

Sample Response:

Response HTTP/1.1 200 OK

10. Get Latest User Status (per alert)

Get the last submitted user status for a given user and alert.

URL:

http:// www.[hidden].gov/rest/userstatus/latest

Query Parameters:

phoneNum - User Phone Number
alertId - ID of the alert
format - Return format {xml, json, jsonp}
callback - Callback name if using jsonp

HTTP Method:

GET

Output Formats:

application/json, application/xml, application/x-javascript

Sample Request:

http://www.[hidden].gov/rest/userstatus/latest?alertId=1&phoneNum=[obfuscated]&format=json

Sample Response:

```
{
  "id": "6",
  "phoneNum": "[obfuscated]",
  "status": "In Route",
  "message": "I'll be there soon.",
  "statusTime": "2009-11-20T09:58:20.924-05:00",
  "latitude": "33.2456",
  "longitude": "-84.78878",
  "locationText": ""
}
```

11. Submit User Location

User can submit his/her current location for tracking purposes.

URL:

http:// www.[hidden].gov/rest/userlocation

HTTP Method:

POST

Input Formats:

application/json, application/xml

Sample Request:

http:// www.[hidden].gov/rest/userlocation

And posting the following data

```
{
  "phoneNum": "[hidden]",
  "statusTime": "2009-11-30T09:58:22.924-05:00",
  "latitude": "33.2456",
  "longitude": "-84.78878"
}
```

Sample Response:

Response HTTP/1.1 200 OK



Southeast Region Research Initiative

National Security Directorate

P.O. Box 6242

Oak Ridge National Laboratory

Oak Ridge, TN 37831-6252

www.serri.org

