Exploring Development of Service-Oriented C2 Systems for Emergency Response

Magnus Ingmarsson
Dept. of Computer and Information Science,
Linköping University, SWEDEN
magin@ida.liu.se

Henrik Eriksson
Dept. of Computer and Information Science,
Linköping University, SWEDEN
her@ida.liu.se

Niklas Hallberg
FOI Swedish Defence Research Agency, SWEDEN
niklas.hallberg@foi.se

ABSTRACT

Local emergency-response organizations must maximize their use of existing resources. Therefore, emergency-response organizations need appropriate command-and-control (C2) systems to coordinate not only their own resources, but also to take advantages of other local actors. The local nature of response coordination imposes additional challenges for the development of C2 systems. In particular, the C2 systems must support coordination across organizational boundaries at the local level.

Service-oriented architectures (SOA) provide new technologies for the development of C2 systems. This approach is based on a set of loosely-coupled services offered by multiple actors rather than a single monolithic system. This work reports the result of a prototype SOA implementation that builds on a previous requirements engineering study for service-oriented C2 systems for local emergency response. The results illustrate how it is possible to develop lightweight C2 systems using state-or-the art Web and SOA technologies. However, there are still remaining organizational and maintainability challenges.

Keywords

Service orientation, SOA, Emergency Response, C2-systems, Crisis Management

INTRODUCTION

Citizens have high demand on well-functioning emergency responses when crisis occur. However, it is difficult to be fully prepared due to that time, location, and nature of crises are hard to predict. Neither, is it financially possible to have staff and equipment recourses exclusively standing by for all conceivable crises. In reality, it is necessary to make use of resources at hand from different actors and to create improvised solutions (Mendonca, Jefferson and Harrald, 2007). The local community is commonly the level on which crises must be handled (Haddow and Bullock, 2006).

The complexity of major response operations involving multiple actors combined with the inherent intricacy of a multitude of different organizations and organizational levels acting according to different mandates and regulations imposes serious challenges for emergency management (Jungert and Hallberg, 2008). It is essential that these actors can allocate resources and synchronize activities in an efficient manner (Shen and Shaw, 2004).

Command-and-control (C2) systems can assist response commanders in situation awareness, planning, and resource allocation (Jungert, Hallberg and Hunstad, 2006). However, traditional C2 systems are difficult and time-consuming to develop, especially because they must accommodate for collaboration with multiple actors and their diverse information infrastructure. Furthermore, the development of C2 systems requires a deep understanding of the activities, objectives, and actors involved as well as their information needs. For local communities, it is important

to integrate local and regional resources from, e.g. rescue services, police and healthcare, into a lightweight C2 system that facilitates cooperation.

Recently, service orientation has emerged as a new approach to system development and to system architectures. Such service-oriented architectures (SOA) are now used to implement information systems for a wide range of business applications. By organizing C2 systems as a set of services, commanders can have access to a multitude of resources and assistance from different organizations. In this model, each actor can make services available to others my publishing them on a network. Client systems can then call services to perform overall tasks, and sometimes even discover new services and accommodate for changes in service availability. One of the goals in service orientation in the emergency-response context is to increase the flexibility of resource allocation, for instance by engage resources normally used for other tasks.

This work builds on the results of a previous study that focused on exploring the possibilities for service orientation in C2 system for emergency management at the local level (Pilemalm and Hallberg, 2008). The previous study identified requirements through interviews and scenario studies and developed an initial mock-up, which was used for scenario-based evaluation. Based on the resulting requirements, we have developed a prototype system for further exploring the use of SOA in development of C2 systems for local emergency management.

The objective of this paper is to present the service-oriented prototype system, Responsoria, and to discuses the use of SOA to realize C2 systems for Crisis management. Responsoria, illustrates the use of current Web and SOA technologies for C2 systems. The goal for the Responsoria prototype is to (1) further explore the potential of SOA in the emergency-management context; (2) illustrate the concept of SOA to practitioners, decision makers, and other actors; (3) identify potential weaknesses of the SOA approach and its supporting technologies; and (4) pinpoint obstacles in the development process. Although this prototype by all means does not implement all the services required for a complete C2 system it uses the relevant SOA technologies and communication protocols as the underlying information infrastructure. For example, Responsoria is based on a state-of-the-art Web-based front end combined with current SOA frameworks and application servers.

In the remainder of this paper, we explain the technical background, outline the methods used, and present the result in terms of the system developed and its components. Finally, we discuss lessons learned from the prototype development as well as advantages and disadvantages of the service-oriented approach for C2 development.

METHOD

This work has been conducted in six major stages (1) a literature study, (2) interviews with emergency response representatives, (3) design of a crisis scenario, (4) design of prototype that supports service-oriented C2 system for emergency response, (5) a scenario based evaluation of the concept, and (6) creation of a demonstrator system. Stages 1–5 have been reported previously in (Pilemalm and Hallberg, 2008) while this paper focuses on Stage 6. However, as a background, we will start by briefly outlining stages 1–5.

Stage 1 was a literature study comprising of the structure of the emergency response in Sweden and crisis organizations responsibilities. Stage 2 involved interviews with representatives from the six major crisis organizations at local level. In Stage 3, the results from the literature study, visits, and interviews were used to develop a scenario of an incident in local community. Stage 4 identified needed services that corresponded to the scenario and to design of a mock-up prototype of service-oriented C 2 systems. In Stage 5, the prototype was evaluated of the user representatives a scenario-based session. The result of the evaluation was used to improve the concept.

Stage 6, which is reported in this paper, started by assessing the requirements from the previous five stages including the result of the evaluation. From this assessment, it was concluded that the architecture of the system needs to permit: (1) deployment with a minimum of effort. (2) High standards compliancy. (3) High modularly. (4) Update and maintenance of parts and the whole system while the system simultaneously is operating.

One of the more prevalent and widely used SOA technologies is Java EE. Java EE brings with it a fault-tolerant, distributed, multi-tier, structure that employs an Application Server that allows the building of a highly modular system. It also provides persistence, transaction processing, concurrency control, etcetera through Java Enterprise beans. Thus, a large amount of work regarding the surrounding system is handled automatically. Furthermore, Java EE has good support by a multitude of tools, for administering, and controlling large systems.

After selecting Java EE as our platform for the backend we concentrated on meeting the requirements for the frontend. From the requirements it was clear that the desired look and feel was that of a desktop application.

However, it was also desirable to combine this look and feel with the other requirements of easy administration, deployment and so on. These requirements are not easy to satisfy simultaneously. Traditionally, a web-based solution will have ease of deployment, but may not provide a desktop application feel. We believe we have found a useful compromise in using Google Web Toolkit which enables both.

We verified the support for Business Process Execution Language (BPEL) in different Integrated Development Environments (IDE). (A more in depth discussion of BPEL is available in the Discussion part of this paper.) Netbeans (Boudreau 2006) became our IDE of choice for this project due to the built in support for both Java Enterprise Beans and BPEL.

Scenario

Systems are not understandable devoid from their context. Therefore, the context of the scenario is briefly presented. In a Swedish community, with about 150,000 inhabitants, on the highway, cutting through the community a truck carrying propane overturns. Shortly after, a bus carrying 25 people run into the truck and as a result another 10 cars are involved in a multiple collision. It is close to Christmas. The incident occurs close to an overcrowded shopping mall and a large event arena where a Christmas concert is ongoing. From this point, the scenario presents how the local crisis response organizations, by using a service-oriented C2 system, initiate the response and collaborate by sharing information and resources. The scenario assumes that there exists a service-oriented infrastructure, which allows the actors to provide and make used of services for crisis management.

RESULTS

This section presents the system architecture, communication, and outlines services implemented. RESPONSORIA is a SOA application with a Web-based user-interface front end.

Architecture

Responsoria is a SOA application with a Web-based user-interface front end, which comprises of a number of subsystems (Figure 1). The core of the system consists of a web server, which delivers Web pages to the browser client. This approach enables a rich desktop application feel while retaining the advantages of a web-based solution for the users. The web server has a proxy that enables communication with other parts of Responsoria. On the application-server side, there is a pluggable structure as well. In the basic configuration, Responsoria uses web services in the form of Enterprise Java Beans (EJBs) for Logging, ID, Mapping, Note-taking, and so on. Due to the pluggable structure, it is easy to extend the system by adding references to other web services.

Currently, the system is running on a Glassfish 2 backend, which is an open-source application server implemented in Java. This approach allows the application server to run on virtually any available system. During development, we have successfully performed development and testing using a cross-platform approach in order to verify maximum functionality and cross-platform distribution ability.

The user-interface client is web-based and implemented using the Google Web Toolkit (1) (GWT). GWT is a framework for developing interactive Web applications. GWT allows developers to write the application in Java while still deploying on a standard web browser. The GWT compiler produces JavaScript code, which is deployable on Web servers. The advantage of GWT is that it enables a look-and-feel similar to desktop applications while providing straightforward development and deployment. We have successfully run the client on MS Windows XP, Apple OS X as well as iPhone, and we expect the system to work on a majority of high-end hand-held devices. Furthermore, we have verified the performance in Internet Explorer, Safari, and Firefox.

The services themselves are Java EE web services. This approach allows for the same type of cross-platform distribution ability as the main application. It also brings with it some significant features for security, quality of service, and portability.

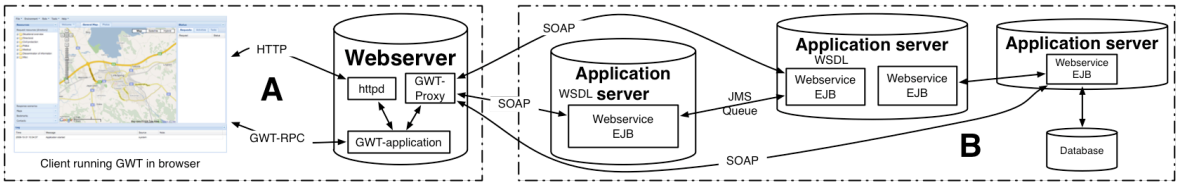

Figure 1: Architecture of the RESPONSORIA system. (A) Web-based user-interface client. (B) Server part consisting of a collection of implemented web services running on application servers.

Motivation for choice of GWT as frontend technology for this type of system

The previous study (Pilemalm and Hallberg, 2008) found that there is a need for systems that allow cooperation between multitudes of different organizations while enabling an open structure for plugging in available, and in some cases, transient services, while at the same time catering to personnel in the field as well as in the central command.

Getting an application-like feel has traditionally been a challenge when using the browser as an interface. Early on, JavaScript was the way to get closer to this than what was normally possible on a browser. However, JavaScript alone does not lend itself very well to activities that require continuous data transfer between the network and the client. We choose GWT because it provides advantages both for developers and end users. In terms of development, GWT allows to write the application in Java, which enables the developer to take advantage of a richer feature set than if the application was written in JavaScript.

Communication in the system

The desktop *feel* stems from a continuous link to the web-server (see Figure 1). This open link enables the web browser to stream data back and forth between itself and the web-server. By streaming data, the application is able to react to user interaction in the same manner that a desktop application does, without the need for the user to press "Submit" or similar type buttons/controls.

The communication in Responsoria follows web standards. Initially, the application is loaded into the browser. A GWT-application may be viewed as having three parts (see Figure 1). The first part is the part that gets loaded into the web browser. It is comprised of JavaScript. The web browser part then has the previously mentioned continuous contact with the second part of the application, which resides on the web-server. For security reasons, the second part of the GWT-application is restricted in what it can do on the server. To gain complete functionality there is the need for the third part, the GWT-proxy. This proxy is needed for the GWT-program to be able to communicate with the outside world and to utilize the services in it.

Let us consider the user of services by discussing the "Mobile phone positioning-service," which can be seen in Figure 2. When the user has started the application (by opening a webpage), the system first loads the application in the form of JavaScript onto the web-browser, draws the interface on the screen and then opens the connection to the webserver. At this point, the user may enter a phone number to be positioned, as well as a time interval. When the user clicks on "Position", the system uses the GWT-proxy to send a request to the proper web-service which returns a result to be presented by the GUI.

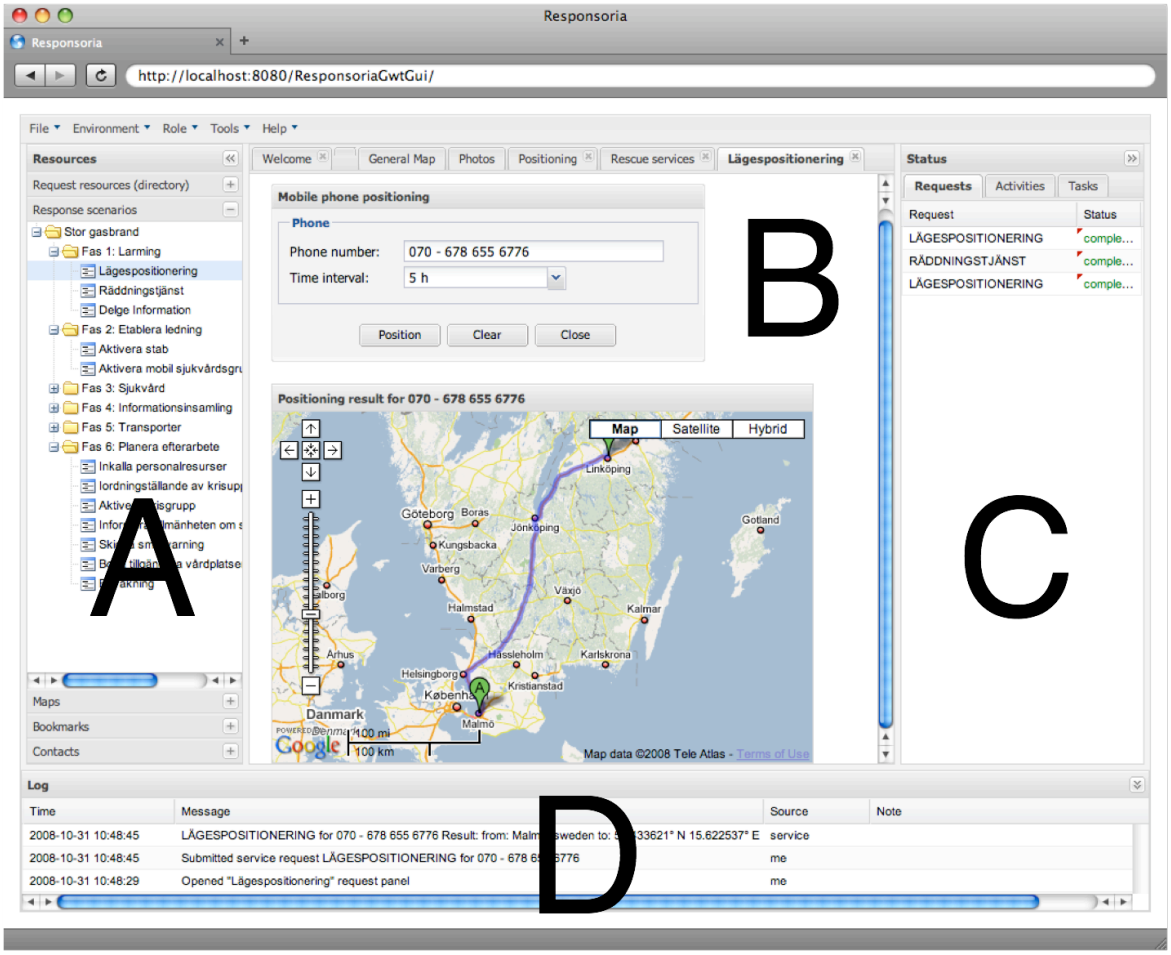

Figure 2: Responsoria main interface. (A) Service selection panel, with preconfigured service groups for different scenarios. (B) Service panel showing the 'Mobile phone positioning' service. In this case showing the trace of a mobile phone for the last five hours. (C) Status panel, showing progress for different requests as well as tasks and activities. (D) Log. Showing all activity in the system as well as provider of a note taking function.

Implemented services in the demonstrator

We have implemented a number of the services identified as necessary from the previous study. They are implemented as proof-of-concept services. We have also created a number of metaservices for internal use within the system: $ID^{Metaservice}$*:* Creates unique ID numbers for the requests in the system. *Mobile phone positioning:* Using Google Maps and geographic coordinates we plot the current position of a simulated mobile phone and/or its travel pattern for a selectable period of time. *Distribute information:* Sends information to designated receivers. *Request rescue services:* Enables ordering of rescue services for different tasks. *Logging:* Automatic logging of activities that take place in the system. *Note taking:* Possibility to augment the log with personal notes.

Some of the services enable the creation of a distributed staff team comprised of members from the different organizations. In particular, the Logging service and the Note taking service aids in this, by allowing the users of the system to implicitly see what others are doing as well as leave time-stamped notes to each other. C2-systems exist to support the staff's role by allowing information exchange. In this case, an automated operational picture is provided by the C2 system based on available information partly generated by the system and partly by its users.

DISCUSSION

Although prototype development is not quite comparable to development of a full-scale C2 system, there are still apparent advantages and disadvantages of the SOA approach already at the prototyping stage. Let us discuss some of the lessons learned from the development of the Responsoria prototype and from the prototype itself.

There are different ways to structure the services. The selection of services we implemented was based on the preceding requirements analysis and scenario description (Pilemalm and Hallberg, 2008). This model means that the services are structured according to the scenario and what the actors can deliver. However, it is possible to structure the services in alternative ways. It is particularly interesting to consider alterative ways to structure the Responsoria client, for instance by breaking up the client user interface and structure it as a set of services. Such services would not necessarily correspond directly to organizational activities, but rather to specific functionality in the user interface and end-user tasks. For example, we have experimented with making the list of services in the Responsoria user-interface client itself a service (i.e., the client then asks this service to provide a list of relevant services). In a similar way, it is achievable to develop the user-interface client as a set of services.

It is possible to create new services from aggregation of services. For example, the Business Process Execution Language (BPEL) is an XML-based language for describing business processes supported by an aggregation of primitive services that together support process flows. (Kloppman et al. 2004) One of the advantages of languages such as BPEL is that they are designed to be used by subject-matter experts (i.e., non-programmers) to implement services (e.g., in a graphical flowchart language). Initially, we attempted to use BPEL to describe services in the Responsoria prototype. For the service domain and the scenario in our C2 application, however, we found that it was not meaningful to describe services as BPEL processes. In addition, it is unclear whether it is meaningful for non-programmers to use process-definition languages directly in the emergency-management C2 domain.

The software development tools are important for the implementation of services and service-oriented systems. Integrated Development Environments (IDEs) can support developers in creating and maintaining source code and definition files for SOA (e.g., classes, interfaces, WSDL, and BPEL). In our practical development work, we found that the IDE to be both an indispensable tool and a limiting factor for SOA development. The IDE proved quite useful for creating source code templates and stubs and generating WSDL and BPEL documents. However, we found the refactoring support for SOA-related changes limited and sometimes incomplete or flawed. Furthermore, manual troubleshooting and repair of overly complex XML documents (e.g., WSDL and BPEL) is difficult and time consuming.

Security is an important issue for C2 applications. In general, it is difficult to secure SOA-based applications. For example, the change from traditional monolithic systems by networked services components creates new potential attack vectors. Although SOA frameworks can provide support for encryption of network communication and authentication of services, there might still be issues in terms of potential denial-of-service attacks and analysis of communication patterns, which can expose the system operation although the actual communication content is encrypted.

Quality of Service (QoS), is a challenge in this type of SOA framework. Traditionally QoS is defined as having sufficient bandwidth to satisfy the requirements of a particular service, which also is true in this situation. As expected, the heterogeneous and multi-facetted amalgamation of services presents a challenge when it comes to deciding which service to use in a given setting. The quandary of selecting the right service is especially true when one wants to utilize services that perhaps are provided from other service providers than traditional ones. This challenge has been pointed out before, and we have developed tools (Magubi) that are designed to handle this type of situation (Ingmarsson 2007). This integration has not been carried out in the first phase of constructing Responsoria, but is under way in the current second phase.

The short-term and long-term maintainability of service-based C2 systems is a concern. Already during systems development, we found that it was sometimes difficult to maintain the prototype because seemingly small changes to service functionality could result in multiple changes throughout the system. For example, the addition of a new parameter to a service call resulted in changes to the service, the service description, the service call, and the Web client and its proxy. Sometimes, it was more efficient to create a new service with the added parameter from scratch than to modify the existing one. This problem would be even more difficult with an operational system in a multi-organizational environment. We believe that it is necessary to accommodate for service maintenance already during system development and to reach consensus among the organizations involved on service evolvement.

CONCLUSION

The Responsoria prototype illustrates the use of the SOA approach to development of C2 systems for emergency management at the municipal level. The prototype addresses two of the most important design goals set forth by Pilemalm and Hallberg (2008), namely to (1) make it possible for the emergency-management teams to keep working in a similar fashion as to what they are used to and (2) allow the use of existing resources from several actors in a crisis situation. The prototyping effort highlighted both the advantages and disadvantages of SOA. The combination of interactive Web technologies, such as GWT, and SOA technologies, such as the Java EE framework, makes it possible to develop lightweight C2 systems with web-based user-interface front ends. However, there are technical challenges in doing so. For example, the development environments are still immature, which makes iterative development difficult. Furthermore, there are remaining organizational issues, for instance in terms of agreeing on service interfaces, service functionality, service availability, and service maintenance. Nevertheless, we believe that the SOA approach is viable and that is can contribute to the development of C2 systems. In our continued effort, we shall explore the use of service-discovery techniques for assisting users in selecting appropriate services. Specifically, we plan to integrate Magubi (Ingmarsson 2007) into Responsoria to enhance the service-discovery aspect of the system.

ACKNOWLEDGMENTS

This work has become possible due to grants from the Swedish Emergency Management Agency (KBM).

REFERENCES

1. Boudreau, T., Tulach, J. and Unger, R. 2006 Decoupled design: building applications on the NetBeans platform, *Companion To the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications* (Portland, Oregon, USA, October 22 - 26, 2006). OOPSLA '06. ACM, New York, NY, 854-854. DOI= http://doi.acm.org/10.1145/1176617.1176734
2. Dewsbury, R. (2007) Google Web Toolkit Applications, *Addison-Wesley Professional*.
3. Haddow, G. D. and Bullock, J. A. (2006) Introduction to Emergency Management. Butterworth-Heinemann, Boston, MA.
4. Ingmarsson, M. (2007) Modelling User Tasks and Intentions for Service Discovery in Ubiquitous Computing, Licentiate thesis, Linköping University, Linköping.
5. Jungert, E. and Hallberg, N. (2008), An Operational Picture Systems Architecture for Crisis Management, Proceedings of the 14th International Conference on Distributed Multimedia Systems (DMS'2008), Boston, MA.
6. Jungert, E., Hallberg, N. and Hunstad, A. (2006) A Service-based Command and Control Systems Architecture for Crisis Management, The International Journal of Emergency Management, 3, 2, 131-148.
7. Kloppmann, M., König, D., Leymann F., Pfau, G. and Roller, D. (2004) Business process choreography in WebSphere: Combining the power of BPEL and J2EE, *IBM Systems Journal,* 43, 2.
8. Mendonca, D., Jefferson, T. and Harrald, J. (2007) Collaborative adhocracies and mix-and-match technologies in emergency management, *Communications of the ACM*, 50, 3, 44-49.
9. Pilemalm, S. and Hallberg, N. (2008) Exploring Service-Oriented C2 Support for Emergency Response for Local Communities, *Proceedings of ISCRAM 2008,* Washington DC.
10. Shen, S. Y. and Shaw, M. J. (2004) Managing coordination in emergency response systems with information technologies, *Proceedings of the Tenth Americas Conference on Information Systems*, New York, NY.