2009 Tenth International Conference on Mobile Data Management: Systems, Services and Middleware

Adaptive Query Routing on Distributed Context - The COSINE Framework *

Lukasz Juszczyk, Harald Psaier, Atif Manzoor, Schahram Dustdar
Distributed Systems Group, Information Systems Institute
Vienna University of Technology, Austria
{juszczyk,hpsaier,manzoor,dustdar}@infosys.tuwien.ac.at

Abstract

Context-awareness has become a desired key feature of today's mobile systems, yet, its realization still remains a challenge. On the one hand, mobile computing provides great potential for adaptations based on sensed contextual information. On the other hand, the lack of dependability in mobile networks hampers an efficient provision of this information to requesting clients. In this paper we present COSINE, a context management framework for mobile environments. COSINE has been developed on the principles of peer-to-peer computing and establishes context sharing infrastructures consisting of loosely coupled Web services. The services represent modular entities of the applied context model and manage the retrieval, aggregation, query, and provision of context data. Clients access the distributed information transparently via proxies and a self-adaptive routing of queries provides increased fault-tolerance, which is essential in mobile environments.

1 Introduction

Since context-awareness was introduced by Schilit et al. in 1994 [16], it has been regarded as a key feature of systems operating in dynamic environments. Today, this comprises domains such as domotics [13], collaborative working environments [8], and many more [3] of which most fall into the category of mobile and pervasive computing. A particularly challenging domain is the support of emergency teams that operate directly in the fields of disaster and are being coordinated via portable computing devices capable of wireless communication [5, 14]. In these teams, human operators execute response processes in which maximum effectiveness is of paramount importance, as it affects the number of saved lives after natural disasters. Without doubt, the execution of the processes can be significantly improved by having up-to-date knowledge about the status of entities within the crisis situation, such as human operators, victims, and infrastructure objects. Though, the unstable characteristics of the established mobile ad-hoc networks make it difficult to share this knowledge in a dependable manner. Generally, disconnections and packet loss pose challenges to any distributed system. However, these issues have a severe impact on emergency response systems as the instability hampers the provision of important context information and, therefore, hampers the execution of the response processes. Such problems originating from the underlying network cannot be fully overcome at a software level, but, there are ways to mitigate their negative effects.

In this paper we present COSINE[1], a service-oriented framework for supporting context-awareness in mobile environments. Our solution is based on dynamically established joint context structures composed of light-weight Web services which provide modular entities of the distributed context data. The main contribution of COSINE is the flexible handling of network dynamics and a transparent dispatching of client requests to the services, which allows to compensate temporal unavailabilities of context sources by re-routing requests to replicating substitutes.

This paper is structured as follows. In Section 2 we illustrate the motivation for our work. Section 3 explains the concepts of COSINE. Section 4 deals with the open-source prototype and the results of our experiments. In Sections 5 we compare our framework to existing works and we conclude this paper in Section 6.

2 Motivation

COSINE has been developed in the scope of the WORKPAD project [5] which deals with building an adaptive software infrastructure for supporting collaborative work of emergency teams. WORKPAD combines a back-end peer-to-peer (P2P) community of inter-organizational hosts, providing advanced services for knowledge integration, with a mobile front-end P2P community of operators working in the fields of disasters. In the front-end, the operators establish mobile ad-hoc networks (MANETs) and execute response processes which are coordinated in an adaptive man-

*This work is partially supported by the European Union through the STREP Projects inContext (FP6-034718) and WORKPAD (FP6-034749).

[1] **CO**ntext **S**haring **I**n u**N**reliable **E**nvironments

978-0-7695-3650-7/09 $25.00 © 2009 IEEE
DOI 10.1109/MDM.2009.101

ner. The adaptivity is achieved by monitoring execution states of the processes, by correlating them with dynamic context data, and by taking action in case of deviations [7]. Apart from this automated self-adaptivity, context data is used to support the workers by displaying the current situation (e.g., annotated objects in areas, location of team members) on maps at their GIS clients [12]. In both cases, the client application subscribes to relevant parts of the context model and is notified when context data changes.

Evidently, context-awareness is of high importance for a disaster response system like WORKPAD. However, it is difficult to achieve if one considers the dynamics of MANETs [17]. The main problem is that distribution of context data automatically implies that parts of the shared data will be inaccessible if their hosting devices become unavailable (of course, unless the share is fully replicated which is, however, not feasible due to bandwidth constraints). As a consequence, WORKPAD requires the context management system to cope with a volatile availability of participants and, in addition, to detect changes in the environment quickly and to compensate failures, if possible. Hence, the concepts of COSINE have been designed to fulfill the requirements in distributed context sharing with the restrictions of MANETs. Usually, the requirements differ significantly depending on the area of application, however, for COSINE we have identified these:

- *Context model-independence:* The system must be customizable to various application domains.
- *Subscription & notification:* In addition to querying, the system must support subscription and notification.
- *Context freshness:* Context data must be propagated quickly to requesting clients.
- *Decentralization:* A decentralized peer-to-peer approach is required, avoiding single points of failure.
- *Location-transparency:* Access to distributed context must be transparent and the system must take care of locating proper context sources.
- *Fault-tolerance:* A volatile availability of nodes must not harm the operation of the context system.

3 COSINE

The COSINE framework provides a basis for context-awareness in small to medium-scale mobile environments. This comprises the storage and provision of XML context data to peers, a modeling of the share's structure in XSD, and access to the distributed context data via the XPath language. Moreover the framework provides a flexible handling of network dynamics, grounded on (a) decoupled service-oriented communication, (b) a flexible establishment P2P-based context structures, and (c) an adaptive routing of query requests, which can compensate temporal unavailability of context sources.

3.1 Service-oriented Context Sharing

Service-orientation provides various advantages to distributed systems which require high flexibility. The most outstanding feature is the loose coupling between services and clients, which reduces dependencies and allows clients to bind dynamically to the "best" service among equivalent ones. This flexibility is especially useful in dynamic environments where participants can appear and disappear at any time and, therefore, dependencies must be kept low. COSINE makes advantage of this to achieve tolerance on a volatile availability of context providers and to be able to invoke substitute services which provide replicated context.

For supporting these features, we have developed RESCUE [11], a service-oriented middleware for MANETs. In RESCUE, nodes advertise their services and listen to advertisement messages of other peers. Knowledge about the environment is stored and contains published descriptions about all discovered services. Client applications can search for services by either querying the database or by placing subscriptions to be notified when matching services become available or unavailable. Due to the active advertisement protocol, the middleware can notify clients almost immediately after the appearing of a service in the network. Eventually, remote services can be invoked via persisted asynchronous messages. In [11] we presented an evaluation which stated that, due to an efficient discovery protocol, RESCUE can usually detect changes in the environment in less than 2 seconds while keeping network traffic low.

COSINE derives a significant part of its flexibility from the RESCUE middleware. This comprises (a) increased scalability, due to a light-weight and incremental discovery protocol, (b) up-to-date awareness about services in the environment, due to quick notifications, (c) increased communication reliability, due to optimized invocations, and (d) interoperability, due to SOAP Web service interfaces.

3.1.1 COSINE Web services

In COSINE, joint context shares are established by discovering context services in the network, analyzing the type of context data they provide, and incorporating them dynamically into the structures of the shares. The type analysis is performed by checking the service's published descriptions and interfaces. From the descriptions, the system determines the type of context data and the service's designated position inside the share's structure (explained in the next section). By checking which interfaces are provided by the service, COSINE knows which interaction types it supports. In general, services can be context providers, context consumers, or both, as illustrated in Figure 1. At the provider side, the sensor can support querying and subscription. At the consumer side, context data can be updated directly or come as a notification to a performed subscription. With COSINE, we provide 3 basic types of services:

Sensor Service: Retrieves context data, such as GPS coordinates or user generated data, from a (virtual) sensor and provides it via a Web service interface.

Aggregator Service: Apart from sensor data extraction, this type of service handles the aggregation of context into a higher level structure, in order to support queries which correlate multiple types of data. It analyzes the model description to determine which context should be aggregated and retrieves it from the providing services.

Context Management Service: The CMS keeps track of the availability of context services. However, it does not collect data but merely acts as a proxy, routing requests to proper services. For this, it maintains an up-to-date view on the context structure and compares it with the used model. The CMS is running on every COSINE instance and provides transparent access to the distributed context share, hiding all complexity.

These services constitute the grounding of COSINE by providing basic functionality for managing and accessing context data. The sensor and aggregator services are provided as abstract classes which can extended with customized functionality. It is possible to extend the infrastructure with additional types of services which, however, must implement the corresponding Web service interfaces.

3.2 Establishment of a Modular Context Share

COSINE must cope with a volatile availability of nodes, must detect changes quickly, and must incorporate new context services to and remove unavailable ones from the share's structure. The maintenance of the structure requires the knowledge of the relationships between the context types, which is referred to as a context model. As outlined in [4], numerous context models have been proposed, ranging from primitive ones, e.g., based on name-value tuples, to highly expressive but complex ones, e.g., based on ontologies. However, as COSINE was required to be built upon well-accepted standards, it was obvious to model the context structure in XSD and to represent the shared data as a (virtual) XML document which can be queried with XPath. Even though this implies restricted reasoning capabilities, compared to more sophisticated models, such as ontology-based ones, we followed the directive of keeping things simple and we underline the convenience for developers of using XML-based standards with rich tool support.

The context model itself consists of definitions of context types. Relationships between them are specified either directly via the tree structure of XML (parent-child relation) or via references inside the context data. For identification, each context type has a *contextID* element and the following snippet shows a simple instance of a worker's *contextID*. It contains the UUID of the service providing the worker's context and of its parent in the hierarchy, which might be a team context service aggregating its workers' context.

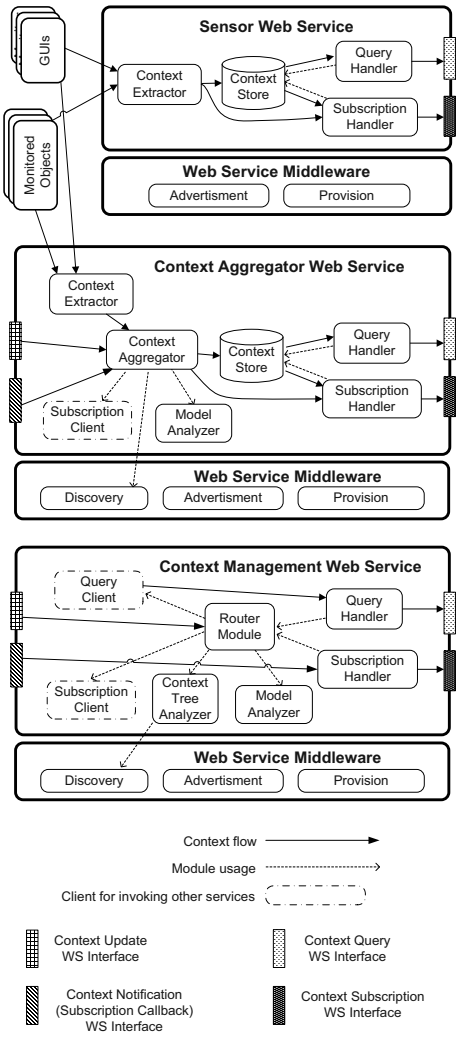


Figure 1: COSINE Web Services

```
<contextID firstName="Joe" lastName="thePlumber" ...
    uuid="bc97a9f0-84..." belongsTo="b63ceca0-84..." />
```

During the initialization of a context service, which provides context data of a specific type, COSINE analyzes annotations inside the type definition and populates the *contextID* fields accordingly. For example, a model designer can make COSINE auto-generate values, such as UUIDs, or he/she can force the user to enter data manually (e.g., his name). Furthermore, he/she can instruct COSINE to set up relationships automatically, e.g., binding a GPS sensor service to the local worker service, or let the user choose, for instance if a worker must be bound to one of the detected team services in the network. Eventually, the initialized service publishes its description, including the *contextID* field, via the RESCUE middleware and this way announces its

presence and the type of provided context data. These announcements are monitored and analyzed by all remote COSINE instances and their consumer services.

The aggregator services process them as illustrated in the following algorithm. If the announced service is verified to be a designated child of it, the aggregator starts retrieving its context data and merging it into its own model. (Due to space constraints we have omitted the merging algorithm.)

```
NEWCONTEXTSERVICEDETECTED @ AGGREGATOR(service)
 1  belongsTo ← service.getContextID().getBelongsTo()
 2  if ctxModel.isChildOf(service, this) and
 3     belongsTo = this.getUUID()
 4     then if service.supportsSubscription()
 5             then subscriptionClient.subscribe(service)
 6             else if service.supportsQuery()
 7                     then queryClient.retrieve(service)
```

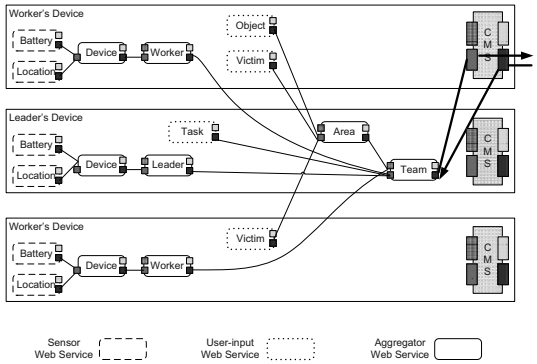
In contrast to the aggregator services, the CMS only observes the environment and maintains an view on the share, in order to know whereto route incoming requests.

```
NEWCONTEXTSERVICEDETECTED @ CMS(service)
 1  belongsTo ← service.getContextID().getBelongsTo()
 2  parent ← ctxTree.lookupUUID(belongsTo)
 3  if parent ≠ null
 4     then if ctxModel.isChildOf(service, parent)
 5             then appendChild(parent, service)
 6     else orphanList.add(service)
 7  orphans ← orphanList.getBelongingTo(service)
 8  for each o in orphans
 9     do if ctxModel.isChildOf(service, parent)
10             then appendChild(service, o)
11                  orphanList.remove(o)
```

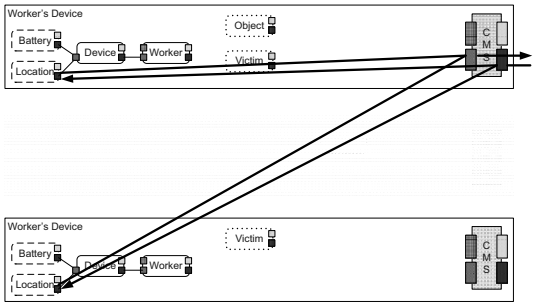
By applying these algorithms, COSINE establishes distributed structures, such as the one shown in Figure 2(a) in which links symbolize context aggregation and the CMS acts as a proxy for requests. The flexibility of this approach becomes evident if one considers the ability of the RESCUE middleware to detect changes in the network almost at realtime [11], which allows COSINE to react quickly to failures by adapting routes of requests.

3.3 Adaptive Query Routing

The usage of aggregator services has also a positive effect on the availability of data, as retrieved context is automatically replicated at the aggregators. This makes it possible to access desired context, even in case of unavailability of the original source. For providing this feature, COSINE expects clients to direct requests (query, subscription) to the local CMS instance which has an up-to-date view on the current structure of the context share and, in combination with the knowledge of the context model, knows to which services it can dispatch the request. Furthermore, due to acting as a proxy and being able to analyze the context



(a) Initial routing on context tree



(b) Adapted routing after disconnection of leader

Figure 2: Dynamic routing of requests

flow, the CMS uses pluggable rating modules for calculating the quality of the services, in order to optimize the dispatching, e.g., by preferring close-by nodes or ones which deliver context data of higher freshness. For every incoming request, the CMS creates a set of possible routings (direct context providers for requested data and their aggregating/replicating services), selects the best rated one, and dispatches the request to the services.

```
INITIALROUTING(request)
 1  routes ← getDirectSrc(ctxTree, ctxModel, request)
 2  routes.add(getAggregators(ctxTree, ctxModel, routes))
 3  for each r in routes
 4     do for each s in r.serviceSet
 5           do for each p in ratingPlugins
 6                 do r.costs ← r.costs + p.getCosts(s)
 7  best ← routingWithMinCosts(routings)
 8  for each s in best.serviceSet
 9     do query ← transformQueryString(request, s)
10        if request.isSubscription()
11           then subscriptionClient.subscribe(s, query)
12           else queryClient.retrieve(s, query)
13  forwardResponsesTo(request.client)
```

In case of a change in the context structure (relevant context services appear or old ones disappear), the algorithm is

rerun, in order to determine if a different route is required, and the dispatching of the request is adapted, if needed.

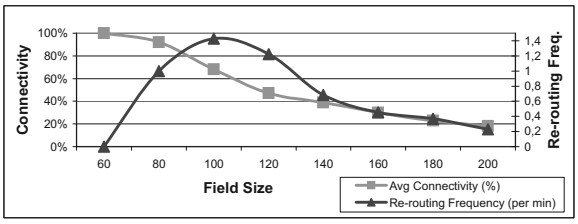
The following scenario demonstrates a re-routing of a request caused by changes in the environment. In Figure 2(a), a client on the first worker's device initiates a subscription about the location of all medics via the XPath request '`/Team[contextID/@teamName='Medics']/Worker//Location`'. The CMS analyzes the string for referenced context types and determines which currently available provider services are able to process this request. By correlating potential sources with their ratings, the CMS decides to route the subscription to the *Team* service of the leader, which has all location data aggregated and continuously updated. Additionally, it forwards all incoming notifications back to the requesting client. Yet, shortly after, a connection loss of the leader makes all his/her services unavailable (see Figure 2(b)), which is detected by the CMS. The CMS analyzes the new structure of the share and decides to split the request and to route it to the *Location* sensor services of the individual workers, which are known to belong to the desired team. Consequently, it transforms the XPath request to '`/Location`', subscribes with it at the *Location* services of the workers, and forwards the notifications again to the client. The client is kept unaware of the performed re-routing to substitute services, as the adaptation is performed transparently.

4 Prototype and First Experiments

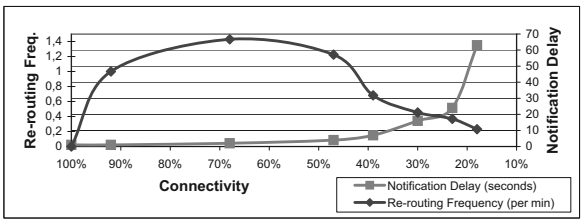
The COSINE prototype has been developed for Java ME and it reuses several open-source libraries for XML processing and SOAP communication. We believe that our framework will be also useful outside the scope of the WORKPAD project and, therefore, we have released the prototype under the GNU Lesser General Public License and provide it at our Web site [2].

For the first evaluation of COSINE, we have concentrated on measuring the average freshness of context data (how much time elapses until new context is propagated to a requesting client) depending on network dynamics. In general, the freshness depends on the connectivity of the participants and on how context is propagated. The effect of connectivity is obvious as disconnections cause delays. However, the propagation strategy is subject to trade-offs. On the one hand, notifications should be sent out immediately to achieve an optimal freshness, which, on the other hand, causes message showers and high network traffic. In our current approach, we handle this problem by adapting notification intervals depending on the sensor activity and combine multiple notifications into single messages.

In our experiments we simulated a MANET of workers performing random movements which affected the overall connectivity among them. We specified the field size, the workers wireless range, and speed and direction of their



(a) Re-routing frequency and connectivity depending on field size



(b) Notification delay and re-routing freq. depending on connectivity

Figure 3: Experiment Results

movements. A connection between two worker's devices was regarded as established if they were either within direct wireless range or were interconnected via forwarding nodes. On top on this testbed, we set up a small-scale environment of 5 workers establishing an extended version of the share depicted in Figure 2(a) and placed subscriptions about their locations. Each round was conducted with a fixed wireless range (50m) but with a different field size (60-200m) in order to achieve a varying connectivity among the workers.The CMS routed the requests either to the *Location* sensors of the workers and/or to the *Team* service which was aggregating the worker's context.

Figure 3(a) shows the effect of a growing field size on the connectivity and, consequently, on the frequency (per minute) of necessary re-routings of the subscriptions. Obviously, the connectivity was shrinking because the workers had more space to move and more likely lost connections. However, regarding the frequency of re-routings, which were performed to pick the best-rated context sources among the available ones, it becomes well visible that it followed a gamma distribution. This is due to the fact that a wider field reduces continuously the frequency of other workers joining ones connectivity range and, therefore, reduces the number of options to choose from. Figure 3(b), on the other hand, illustrates the results directly from the perspective of a decreasing connectivity. In this case, the re-routing frequency follows a beta distribution, as adaptations are not necessary on 100% (perfect availability) and not possible on 0% connectivity (isolation). Moreover, the notification delay follows an exponential distribution, due to the lowering probability of finding context sources at all.

Altogether, we want to point out the low notification de-

lay (high freshness). We believe that a delay of less than 5 seconds for networks with an average connectivity of 40% is a good result. This is mainly achieved due to the quick change notifications from the RESCUE middleware and the failure-compensating routing mechanism of COSINE. Detailed scalability tests, determining the limits of the COSINE approach, will be subject to future evaluations.

5 Related Work

Various middleware projects, such as the Context Toolkit [15], RCSM [18], and SOCAM [9] have been developed for supporting context-aware applications. Similar to COSINE, they manage sensor access through APIs, support distributed sharing of context data, and reasoning on this data. However, they use centralized discovery facilities for finding context sources. This approach is acceptable in environments, such as smart homes, which are rather static, however, is not reasonable in MANETs.

The CORTEX project [1] deals with co-operating mobile context-aware software components, called sentient objects. For this purpose a middleware was developed, which handles flexibly the retrieval and fusion of context data from surrounding sensors. In contrast to COSINE, the middleware does not provide any means for compensating failures caused by a volatile availability of context sources.

Solar [6] is a sophisticated context management infrastructure for pervasive environments. It establishes P2P structures based on distributed hash tables and uses super peers for managing groups of context providers. Similar to COSINE, it supports aggregation, subscription, and dispatching of requests. Solar's focus lies on scalability, which is a in important aspect, however, the authors claim that system is not suited to dynamic environments.

The last related project is the PACE middleware [10] providing context aggregation, quality of context evaluation, and a simpler form of fault-tolerance relying on flexible discovery of context sources and a loose coupling among the components. COSINE, however, provides additional redundancy of context sources and due to the adaptive routing of requests is able to compensate faults to some extent.

6 Conclusion

In this paper we have presented the concepts of COSINE, a context sharing framework for mobile environments. Our work has been focused on handling challenges inherent in mobile communication and on providing a higher level of reliability to distributed context sharing. We have mitigated the impacts of volatile dependability inside MANETs by using decoupled service-oriented computing, by establishing flexible peer-to-peer structures of context providers, and by applying a self-adaptive routing of requests for compensating faults. In spite of the complex management of such infrastructures, COSINE allows a convenient usage of its features by hiding all complexity from the clients and by providing transparent access to the distributed context data.

References

[1] CORTEX Web site. `http://cortex.di.fc.ul.pt`.

[2] COSINE Prototype Web site. `http://www.infosys.tuwien.ac.at/prototypes/cosine`.

[3] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *IJAHUC*, 2(4):263–277, 2007.

[4] C. Bolchini, C. Curino, E. Quintarelli, F. A. Schreiber, and L. Tanca. A data-oriented survey of context models. *SIGMOD Record*, 36(4):19–26, 2007.

[5] T. Catarci, M. de Leoni, A. Marrella, M. Mecella, G. Vetere, B. Salvatore, Dustdar, L. Juszczyk, A. Manzoor, and H.-L. Truong. Pervasive software environments for supporting disaster responses. *IEEE Internet Computing*, 12(1):26–37, 2008.

[6] G. Chen, M. Li, and D. Kotz. Data-centric middleware for context-aware pervasive computing. *Pervasive and Mobile Computing*, 4(2):216–253, April 2008.

[7] M. de Leoni, M. Mecella, and G. D. Giacomo. Highly dynamic adaptation in process management systems through execution monitoring. In *BPM*, volume 4714 of *LNCS*, pages 182–197. Springer, 2007.

[8] D. Ejigu, V.-M. Scuturici, and L. Brunie. Coca: A collaborative context-aware service platform for pervasive computing. In *ITNG*, pages 297–302. IEEE, 2007.

[9] T. Gu, H. K. Pung, and D. Zhang. A service-oriented middleware for building context-aware services. *J. Network and Computer Applications*, 28(1):1–18, 2005.

[10] K. Henricksen, J. Indulska, T. McFadden, and S. Balasubramaniam. Middleware for distributed context-aware systems. In *OTM*, volume 3760 of *LNCS*, pages 846–863. Springer, 2005.

[11] L. Juszczyk and S. Dustdar. A middleware for service-oriented communication in mobile disaster response environments. In *MPAC*, pages 37–42. ACM, 2008.

[12] A. Marrella, R. Russo, A. Capata, M. Bortenschlager, and H. Rieser. A geo-based application for the management of mobile agents during crisis situations. In *ISCRAM*, 2008.

[13] F. Mastrogiovanni, A. Scalmato, A. Sgorbissa, and R. Zaccaria. An integrated approach to context specification and recognition in smart homes. In *ICOST*, volume 5120 of *LNCS*, pages 26–33. Springer, 2008.

[14] M. Portmann and A. A. Pirzada. Wireless mesh networks for public safety and crisis management applications. *IEEE Internet Computing*, 12(1):18–25, 2008.

[15] D. Salber, A. K. Dey, and G. D. Abowd. The context toolkit: Aiding the development of context-enabled applications. In *CHI*, pages 434–441, 1999.

[16] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *Workshop on Mobile Computing Systems and Applications*, pages 85–90. IEEE, 1994.

[17] A. P. Snow, U. Varshney, and A. D. Malloy. Reliability and survivability of wireless and mobile networks. *IEEE Computer*, 33(7):49–55, 2000.

[18] S. S. Yau, D. Huang, H. Gong, and S. Seth. Development and runtime support for situation-aware application software in ubiquitous computing environments. In *COMPSAC*, pages 452–457. IEEE, 2004.