ESCAPE – An Adaptive Framework for Managing and Providing Context Information in Emergency Situations*

Hong-Linh Truong, Lukasz Juszczyk, Atif Manzoor, and Schahram Dustdar

Vitalab, Distributed Systems Group, Vienna University of Technology
{truong,juszczyk,atif.manzoor,dustdar}@infosys.tuwien.ac.at

Abstract. Supporting adaptive processes in tackling emergency situations, such as disasters, is a key issue for any emergency management system. In such situations, various teams are deployed in many sites, at the front-end or back-end of the situations, to conduct tasks handling the emergency. Context information is of paramount importance in assisting the adaptation of these tasks. However, there is a lack of middleware supporting the management of context information in emergency situations. This paper presents a novel framework that manages and provides various types of context information required for adapting processes in emergency management systems.

1 Introduction

Supporting emergency situations (e.g., disaster responses) could benefit a lot from the recent increasing availability and capability of computer devices and networks. Pervasive devices have been widely used to capture data for and to coordinate the management of tasks at disaster fields. Communication networks help bringing collected data to the back-end center, and at the back-end, high performance computing/Grid systems and data centers can conduct advanced disaster management tasks. Still, there are many challenging issues in order to utilize advantages brought by pervasive computing, mobile networks, and Grid computing for supporting the management of emergency situations.

Take the scenarios being supported by the EU WORKPAD project [29] as examples. Aiming at supporting natural disaster responses, the front-end of the WORKPAD system will assist teams of human workers by providing services that are able to adaptively enact processes used in disaster responses. On the one hand, these services are executed on mobile ad-hoc networks whose availability and performance are changed frequently. On the other hand, processes carried out in disaster scenarios are normally established on demand and changed accordingly to meet the current status of the disaster scenarios. It means that depending on the context of specific scenarios, tasks conducted during the scenarios can be altered, either in an automatic fashion or a manual one controlled by team leaders. Therefore, it is of paramount importance that these services are able

* This research is partially supported by the European Union through the FP6-2005-IST-5-034749 project WORKPAD.

G. Kortuem et al. (Eds.): EuroSSC 2007, LNCS 4793, pp. 207–222, 2007.

to use context information to adjust the enactment and collaboration of processes. This leads to a great demand for frameworks that can be used to provide and manage various types of context information. However, most of existing context information management middleware are targeted to indoor and small scale environments, e.g., as discussed in [8]. There is a lack of similar middleware for emergency situations. The environment in which supporting tasks for emergency situations are performed is highly dynamic and unstructured.

In this paper, we discuss ESCAPE, a peer-to-peer based context-aware framework for emergency situations with the focus on crisis situations, such as disasters, in pervasive environments. We present the design and implementation of the context management services within ESCAPE that can support multiple teams working at different sites within many responses for emergency situations to collect and share various types of context information. ESCAPE is able to manage relevant context information which is described by arbitrary XML schema and required for emergency responses, and to provide the context information to any clients. In this paper, we also illustrate early experiments of the current prototype of the framework. The ESCAPE framework is an ongoing development. Therefore, in this paper, we only focus on the discussion of the management and provisioning of context information in emergency situations at middleware layer, instead of presenting adaptation techniques and applications.

The rest of this paper is organized as follows: Section 2 discusses the requirement and motivation. Section 3 presents the related work. The architecture of the context management services is described in Section 4. We describe the management of context information in Section 5. Prototype implementation is outlined in Section 6. Experiments are illustrated in Section 7. We discuss existing issues in the framework in Section 8. We summarize the paper and discuss about the future work in Section 9.

2 Requirements and Motivation

Effective responses to an emergency situation (e.g, a natural disaster) require key information at sites where the situation occurs in order to optimize the decision making and the collaborative work of teams handling the emergency. Context information can substantially impact on responses to the situation. The key to the success of responses to an emergency is to have effective response processes which are actually established on-demand and changed rapidly, depending on the context of the situation. Such effective response processes cannot be achieved without understanding the context associated with entities inherent in the situation. Required context information related to entities at each site in the emergency situation is related to not only teams performing responses, for example, information about team member tasks, status of devices, networks, etc., but also entities affected by the emergency situation, such as victims and infrastructure. To date, context information is widely used in context-aware systems but most of them are not targeted to emergency scenarios.

In our work, we consider the support to the management of various kinds of emergency *situations* such as disasters (e.g., earthquake and forest fire). In such situations, many support *teams* of *individuals* will be deployed, as soon as possible, at *sites*

(e.g., a village) where the situations occur in order to conduct situation *responses* (e.g., to rescue victims). All members of the teams will collect various types of data and perform different tasks, such as relief works or information gathering, for supporting response tasks. Within emergency situations, teams are equipped with diverse types of devices with different capabilities, such as PDAs and laptops. These devices have limited processing capacity, memory and lifetime. Moreover, the underlying network that connects these devices together is established as a mobile ad-hoc network in which usually a few devices can be able to connect to the back-end services. In addition to professional teams, teams of non-professional volunteers can also be established in a dynamic fashion. Given the current trend in mobile devices consumption, many people have their own networked PDAs and smart phones which can be easily used to support emergency situations.

Moreover, as teams in the site may perform tasks in a dangerous environment and the teams lack a strong processing power and necessary data, the front-end teams may need support from teams at the back-end. This requires us to store context information at the back-end due to several reasons. For example, teams at the back-end can use context information to perform other tasks that could not be done by the front-end teams. Furthermore, as people at the front-end are working in dangerous environments, latest location information (one type of context information) of people who are in dangerous environment can be used for, e.g., to rescue them in case they are in danger.

To support the above-mentioned scenarios, context-aware support systems for emergency situations using mobile devices and ad-hoc networks have to be developed. An indispensable part of these systems is a context management middleware which must be able to collect various context information related to the emergency situation. Such context information will be utilized at the site by multiple teams and by the back-end support teams. As the middleware has to be deployed in constrained devices, various design issues must be considered. Since devices do not have a strong communication and processing capability, the network of teams is unstructured and the operating environment is highly dynamic. As a result, context information is exchanged between various peers in a dynamic and volatile environment. Therefore, a P2P (peer-to-peer) data exchange model is more suitable.

Context management services have to exchange context information with many supporting tools, such as GIS (Geographic Information Systems) and multimedia emergency management applications. Moreover, we have to make the front-end services interoperable with the back-end (Grid-based) services which might belong to different organizations. As a result, SOA-based models and techniques will be employed in the management and dissemination of context information. In this aspect, the middleware operates on Pervasive Grid environments [17]. However, the context management framework for emergency scenarios should be flexible or be easily adapted to handle context information specified by different models at multiple levels of abstraction since context information in emergency situations is not known in advance. In this respect, the framework should support an extensible data model, e.g. XML, and query mechanism, e.g. XQuery [31] and XUpdate [33]. Then, various plug-ins used to handle specific cases, such as sensors, event notification, event condition action, etc., could be seamlessly integrated into the framework to support situation-specific scenarios.

3 Related Work

Several studies of context-aware systems have already been conducted, such as in [8]. In this paper, we concentrate our study only on existing middleware for managing context data. RCSM [35,5] is a middleware supporting context sensitive applications based on an object model. The JCAF (Java Context Awareness Framework) supports both the infrastructure and the programming framework for developing context-aware applications in Java [4,10]. JCAF is based on the peer-to-peer model but it does not support automatic discovery of peers or a super-peer. Moreover, the communication is based on Java RMI (Remote Method Invocation). The AWARENESS project [1] provides an infrastructure for developing context-aware and pro-active applications. It targets to applications in mobile networks for the health care domain. The PACE middleware [15] provides context and preference managements together with a programming toolkit and tools for assisting context-aware applications to store, access, and utilize context information managed by the middleware. PACE supports context-aware applications to make decisions based on user preferences. The GAIA project is a CORBA-based middleware supporting active space applications [26]. GAIA active space has limited and well-defined physical boundaries so GAIA is not suitable for emergency management. It is targeted to small and constrained environment such as smart homes and meeting rooms. The CARMEN middleware [11] uses CC/PP for describing metadata of user/device profiles while the Mercury middleware prototype [28] describes user, terminal, network, and service profiles using CC/PP. We observed that most of abovementioned middleware do not support a variety of context data inherent in emergency situations. Furthermore, most of them are targeted to in door environment and do not support scenarios of collaborative teamwork in emergency situations in which front-end teams and back-end systems are connected and exchange context information through a large scale and highly dynamic environment.

Relational databases are widely used to store context information. For example, [24] stores context information about geography, people and equipments in a relational database. In [20], Location historical information is stored in a database that can be accessed using SQL. The PACE middleware provides a context management whose backend is a relational database [15]. The Aura context information service is a distributed infrastructure which includes different information providers [22]. Context-aware applications can subscribe to middleware in order to obtain context information through subscription/notification [24,10] or can query information stored in persistent databases, e.g., in [15]. We take a different approach in which we rely on XML-based information and XQuery/XPath for accessing context information. It helps to facilitate the integration with different types of application and support the extensibility and generalization of the middleware in handling different types of context information.

Context information is also widely utilized in personal applications in pervasive computing such as electronic communication [25], in office and education use, e.g., for monitoring user interactions in the office [30], for managing presentations in meeting rooms [16], and for office assistants [34]. In [9], context information in hospitals is used in context-aware pill containers and hospital beds. [21] describes how context information can be used in logistics applications. Our work is different as we aim at supporting emergency management applications which require much diverse context

information and operate in a highly dynamic environment. The ORCHESTRA [3] and OASIS [2] projects focus on disaster/crisis management, however, they do aim at supporting context-awareness.

4 Architecture of ESCAPE Context Management Services

4.1 Architectural Overview

Figure 1 presents the architecture of the ESCAPE context management framework which includes context information management services (CIMSs) and the back-end support system. Each individual's device will host an instance of the CIMS used by the individual who is responsible for collecting and managing context information related to the individual. As individuals are organized into teams, each team will establish a network of CIMSs. For each team, one CIMS whose hosting device has more powerful capability, such as the device of the team leader, will act as a super peer. This super peer will periodically gather context information available in CIMSs within its team and then push the context information to the back-end systems. Within a team, we use a simple peer structure: all peers are equal and there is no forwarding mechanism among peers[1]. Any peer which wants to obtain context information provided by another peer just directly queries or subscribes context information from the provider.

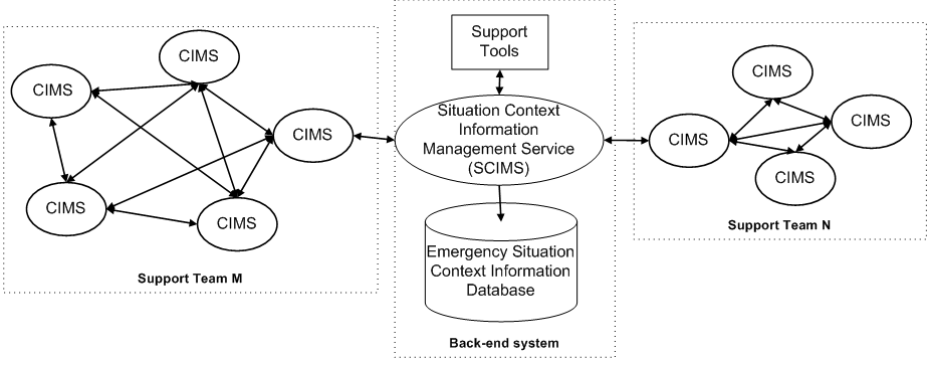

Fig. 1. Architecture of the ESCAPE context management framework for emergency situations

At the back-end, we store context information into a situation context information management service (SCIMS) which keeps all the context data related to a situation. By using this information, various support teams at the back-end can utilize rich data sources and computing services to support teams at sites. Furthermore, the context information managed by SCIMS can be used for post-situation studies. Two different teams can exchange context information by either using the back-end systems, e.g., in case the teams are not located in the same site or the network connection among

[1] This is only at context management middleware level. The underlying network can support multi-hop communication, depending on mobile ad-hoc network techniques employed.

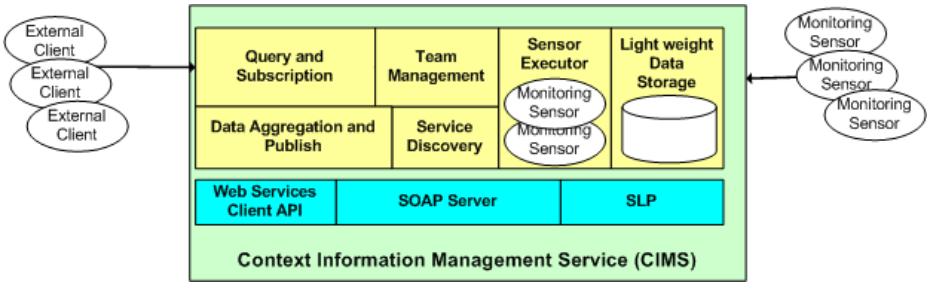

Fig. 2. Overview of the context information management service

two teams is not available, or utilizing the team leader CIMS by searching team leader devices in the network.

Figure 2 presents the architecture of a context information management service (CIMS) which is used to manage and provide context information at an individual's device. The *Web Services Client API* is used to communicate with other Web services. The *SOAP server* provides features for building services based on SOAP. *SLP* (Service Location Protocol) component supports team service advertisements and discovery. The two components *Service Discovery* and *Team Management* will be responsible for discovering other CIMSs and for managing CIMSs belonging to the same team, respectively. The *Query and Subscription* component is responsible for processing data query and subscription requests from any clients. The *Data Aggregation and Publish* component gathers context information from various peers and publishes the information to the back-end. The *Sensor Executor* is responsible for controlling internal sensors collecting context information. These sensors are considered as plug-ins of CIMS. The context information gathered at each CIMS will be stored in the *Lightweight Data Storage*.

4.2 Service Discovery and Team Management

A CIMS will publish information about itself by exploiting multicast service discovery based on SLP (Service Location Protocol)[27]. Each CIMS is described mainly by a triple `(teamID, individualID, serviceURI)` in which `teamID` is used to identify the team; all instances of CIMS within a team will have the same `teamID`. The element `individualID` identifies the individual whose device manages the CIMS whereas `serviceURI` specifies the URI of the CIMS. This triple information will be mapped into SLP advertisements. Based on that, service discovery can be performed.

A CIMS will publish its service information periodically and will keep a record of this information of all CIMSs in its team. A CIMS will check its team members regularly by pinging them. By doing so, each CIMS has an up-to-date record of all its team members. Currently, we use a configuration file to specify the intervals based on which CIMS should publish its information and check its team members presence.

4.3 Publishing and Querying Context Information

In our framework, context information will be collected by different clients and monitoring sensors (software- and hardware-based sensors). CIMSs will provide interfaces

for these clients and sensors to publish and query context information. Being able to handle different kinds of context information, CIMSs will accept any type of context information that is described in XML format, without knowing the detailed representation of context information.

CIMS provides two mechanisms for sensors/clients to publish context information. We distinguish two cases: sensors execution will be and will not be controlled by CIMS. In the first case, sensors will be invoked directly by CIMS as a plugin. To support this, we develop a generic interface that sensors should implement. This interface includes three main methods named `setParameters, execute` and `getContextInfo`, allowing CIMS to initialize sensors, invoke sensors instances and obtain XML-based context data without considering how the sensors are implemented. In the second case, CIMS provides Web services operations for any sensors/clients to publish context information. To retrieve context information, clients have to use Web services operations provided by CIMS. Clients can also specify XQuery-based requests in searching for context information from a CIMS.

4.4 Customized Middleware Components

Supporting reconfigurable middleware components is important in pervasive environments since specific platforms, e.g., PDA or laptop, have a multitude of varying capabilities and support technologies that should be exploited differently. We identify two main components within the CIMS that should be reconfigured according to underlying platforms: *Query and Subscription* and *Lightweight Data Storage*. As we aim at supporting different kinds of context information, the context management framework does not bind to any specific representations of context information as long as the representations are in XML. Using existing tools like Xerces [7] and kSOAP2 [19], we are able to generate and process XML data in constrained devices. Nevertheless, supporting advanced XML processing functions, such as query and update of XML data with XQuery [31] and XUpdate [33], in constrained capability devices is very limited. Our *Lightweight Data Storage* is based on eXist XML database [12] when a CIMS is deployed in normal devices supporting Java SE (e.g., laptop). Otherwise, the storage will be based on a round-robin model in which context information is stored into XML files, and XQuery processing is based on MXQuery [23] which is a lightweight XQuery engine for mobile devices.

5 Management of Context Information

5.1 Context Information Level

In emergency situations, context information will be collected by and exchanged among individuals and teams within different scopes of knowledge, e.g. within the knowledge of an individual or a team at a site during a response or the whole situation. Therefore, we support five levels of knowledge in which context information is available: *individual, team, site, response* and *situation*. The *individual* level indicates the context information within the knowledge of an individual. It means that context information is either associated with or collected by an individual. The *team* level indicates context

information gathered from all individuals of a team. The *site* level specifies context information collected from all teams working on the same site whereas the *response* level indicates context information gathered within a response. The *situation* level specifies all context information gathered during the situation. The five levels of knowledge provide a detailed structure of context information. Based on that, context information can be efficiently shared among teams and utilized for different purposes.

5.2 Storage and Aggregation of Context Information

A single level/place in storing context information, such as only at the team leader device, is not suitable for emergency situations. In such situations, teams at sites are equipped with devices which are not always highly capable and the network is normally not reliable. Therefore, rather than relying on centralized managers, we believe that context information should be provided by and exchanged among individuals in a dynamic fashion. Context information in emergency situation must be widely shared among different teams at different places and be stored for post-situation studies. Thus, context information exchange should not be limited in a single team/place.

Addressing two different storage mechanisms, distributed storage information in mobile devices and in high-end systems is a challenging issue. Our context management services support any kind of context information modeled in XML. At each CIMS, context information sent by sensors to the service is managed in records. Each record r is represented as a triple `(individualID, timestamp, contextDataURI)` where `individualID` is the unique identifier of the team member, `timestamp` is the time at which the context information was collected, and `contextDataURI` is the URI specifying the location from which detailed context information can be retrieved. The detailed context information can be stored into files or XML databases, depending on the capability of the hosting device. As mobile devices have a limited resource and context information will partially be stored in the back-end services, not all context information will be kept in a device. Instead, we employ a round robin mechanism to maintain existing context information stored in each CIMS. Depending on the device capability, the round robin database mechanism may be relaxed.

As mentioned before, context information will be collected at CIMSs which hold the context information at the individual level. Aggregation of context information with in a team will be conducted by the team leader. Context information will be available from different places, and the information has to be stored over the time. At a time t, the newest instance of context information collected is called a *snapshot* of context information. We consider the management of context information at five levels: within a device managed by an individual, within a team, within a site, within a response, and within the whole situation. Consequently, we have five different types of snapshots in the scopes of the above-mentioned levels. Spanning the timeline of situation responses, various teams are deployed and context information associated with the situation is collected by the teams. Since response time is a critical issue in the management of emergency situation and the devices used have limited capabilities, we employ simple mechanisms to manage context information. Let $ctx_s(level, time)$ where $level \in \{individual, team, site, response, situation\}$ denote a context snapshot within a $level$ at a given $time$. Within a team, each member

monitors and collects context information which will be stored and updated locally. A snapshot of context information stored in a device is associated with a timestamp t and is denoted as $ctx_s(individual, t)$. Depending on the capabilities of the device, the number of snapshots kept in a device could be limited to a pre-defined value n.

The context information collected by a team will be stored temporarily at the team leader device or pushed back to the back-end service periodically by CIMS of the team leader. At a given time t, $ctx_s(team, t)$ will be a fusion of $\{ctx_s(individual, t)\}$ for all $individual$ belonging to the team. Similarly, $ctx_s(site, t)$ is a fusion of $\{ctx_s(team, t)\}$ for all teams within the site, $ctx_s(response, t)$ is a fusion of $\{ctx_s(site, t)\}$ for all sites involved in the response. The snapshot of the situation, $ctx_s(situation, t)$, is defined as a fusion of $\{ctx_s(response, t)\}$ for all responses conducted in the situation. While context information at *individual* and *team* levels is available at the front-end, the information of the other levels is available at the back-end system only.

5.3 Provenance of Context Information

All context information gathered could be tracked through our provenance support. We design a generic XML schema based on that provenance information of context information can be described. Figure 3 describes main elements of the schema used to describe provenance information. This representation allows us to specify detailed information about the five levels of knowledge by using elements `situation, site, response, team` and `individual`. The detailed content of provenance information about these levels can be described in XML and encoded by using `<![CDATA[ ]]>` section. The element `collectedAt` indicates the time at which the context information is collected while element `contextDataURI` specifies the URI through which context information can be retrieved.

The framework will automatically store provenance information into the back-end system whenever context information is pushed back to the back-end system. Provenance information is important because it allows us to correlate all gathered context information to its sources and creators, providing tracing capability and improving the understanding of actions performed within emergency situations.

```
<xsd:complexType name="ContexProvenance">
   <xsd:sequence>
     <xsd:element name="provenanceEntry" type="tns:ContextProvenanceEntry"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ContextProvenanceEntry">
   <xsd:sequence>
     <xsd:element name="situation" type="xsd:string"/>
     <xsd:element name="site" type="xsd:string"/>
     <xsd:element name="response" type="xsd:string"/>
     <xsd:element name="team" type="xsd:string"/>
     <xsd:element name="individual" type="xsd:string"/>
     <xsd:element name="collectedAt" type="xsd:dateTime"/>
     <xsd:element name="contextURI" type="xsd:anyURI"/>
   </xsd:sequence>
</xsd:complexType>
```

Fig. 3. Schema (simplified) for describing provenance of context information

6 Implementation

To implement the architecture mentioned in Section 4, we employ various libraries for handling Web services and XML on mobile devices such as kSOAP2 [19] and CDC-based Xerces[6]. XQuery/XUpdate supports are based on eXist [12] and MXQuery [23]. Our current prototype is implemented in Java ME CDC for PDA and can be customized with Java SE-based libraries for normal laptops to exploit advanced Web services and XML capabilities.

In our implementation, *Service Discovery* and *Team Management* are implemented on top of jSLP [18] which is a lightweight Java implementation of SLP for mobile devices. We use the *SOAP server* in Sliver BPEL [14] and implement our service-based components on top of that. Sliver supports both TCP socket-based and Jetty HTTP-based communications. Within a team, CIMSs can communicate with each other using SOAP by selecting one of those communications.

The back-end context information service is currently implemented based on eXist XML database [12]. CIMSs push data to back-end services by using the REST (Representational State Transfer) interface provided by eXist database.

7 Experiments

7.1 Experimental Application: Supporting Disaster Responses

One of the main motivations for developing this framework is to use it in supporting disaster responses in the WORKPAD project [29]. Being able to collect and provide context data relevant to disaster responses is the key issue to the WORKPAD adaptive process management systems used by team leaders to plan response activities. Furthermore, context information is required by the disaster management support based on geographic information systems (GIS). To this end, we have developed a novel context information model for disaster managements. Figure 4 describes main concepts of the

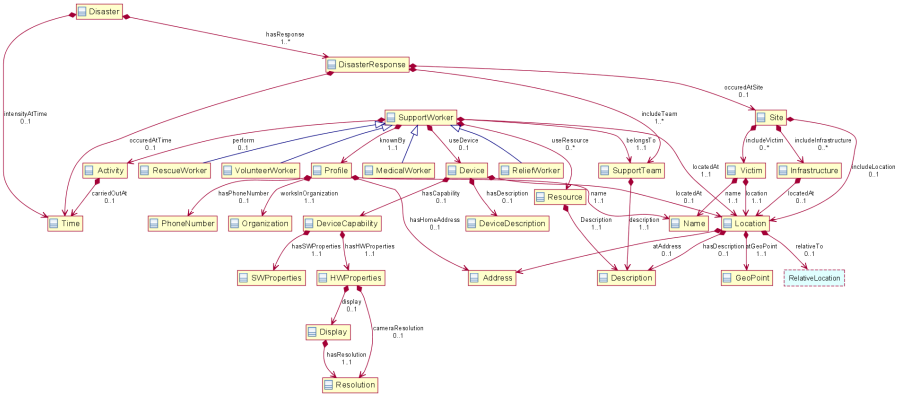

Fig. 4. WORKPAD context information model in disaster responses

first version of the WORKPAD context information model which can describe various context information inherent in a disaster response.

In the case of disaster responses, the five levels of knowledge about context information are *support worker, support team, disaster site, disaster response, disaster* which are mapped to *individual, team, site, response, situation*, respectively. In order to test the current prototype of our framework, we use many simulated sensors whose functionalities are exactly the same as that of real sensors, except that the context information is automatically generated from simulation configuration parameters.

7.2 Testbed

Figure 5 describes our current testbed. We setup a testbed which includes 3 iPAQ 6915 PDAs (Intel PXA 270 416 MHz, 64 MB RAM, Windows CE 5.0, 2GB external MiniSD, IBM J9 WebSphere Everyplace Micro Environment 6.1), a Dell XPS M1210 (Intel Centrino Duo Core 1.83 GHz, 2GB RAM, Windows XP) notebook, and a Dell D620 (Intel Core 2 Duo 2GHz, 2 GB RAM, Debian Linux). Devices in the testbed are connected through a mobile ad-hoc network based on 802.11b. A CIMS is deployed on each device and the Dell D620 laptop is designated as the gateway to the back-end. In our setting, the mobile ad-hoc network bandwidth is limited to 220 Kbits/s but we observed that the average bandwidth is around 150 Kbits/s. The back-end system is based on a Dell Blade (2 Xeon 3.2 GHz CPUs with Hyperthreading, 2GB RAM, and Ubuntu Linux). We use simulated sensors to produce context information according to the WORKPAD context information model. In our experiments, we focus on presenting some preliminary analyses of data transfers and examples of accessing context information using XQuery.

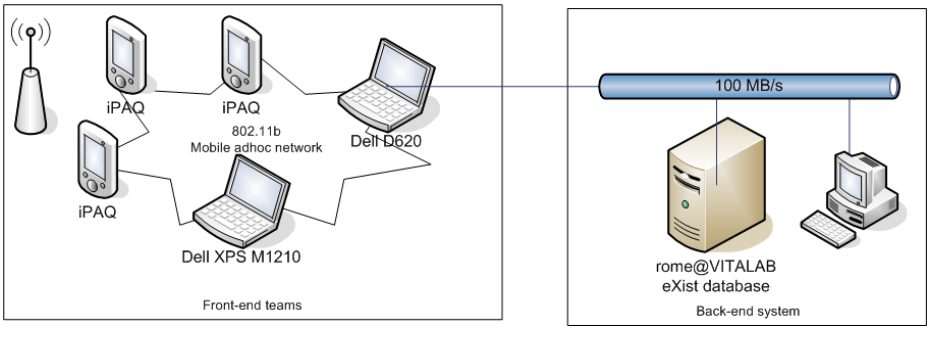

Fig. 5. Testbed deployment

7.3 Performance Analysis

In the first scenario, we consider a team including five members. Three members use PDAs and two members use laptops in our testbed. The Dell D620 is designated as the device of the team leader. Figure 6 shows Jetty HTTP-based and TCP socket-based data transfers between a CIMS deployed in a member and a CIMS deployed the team

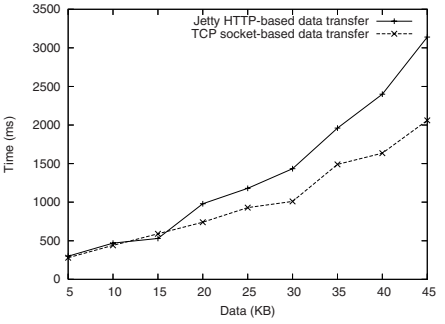

Fig. 6. Data transfer between a CIMS member (PDA) and CIMS team leader (laptop)

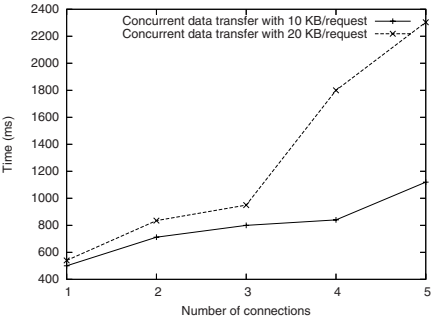

Fig. 7. Concurrent data transfer between a CIMS (PDA) and its client (laptop)

leader. Overall when the data size is smaller than 15KB the performance is almost the same. However, TCP socket-based data transfer outperforms Jetty HTTP-based when the transferred size is increased. We also found that the TCP socket-based communication in Sliver is not quite stable. Therefore, in the following experiments we relied on Jetty HTTP-based communication. Figure 7 presented the concurrent data transfer tests between a CIMS in a PDA and a client in the Dell D620. When doubling the data size from 10KB/request to 20KB/request, with the number of concurrent connections is smaller than 3, the transfer time increases but not substantial. However, with more than 3 connections, the transfer time increases significantly. This suggested that we should not use multiple concurrent connections to transfer a large data size to PDAs. Moreover, PDAs might not be used as team leader device in case the number of team member is large and there is a need to transfer large amount of data.

In the second scenario, we modeled a system including four teams. Two team leaders use PDAs and two team leaders use laptops. CIMSs running on devices of the four team leaders will gather information of teams and send the context information back to the back-end system. Non-leader members of a team are simulated through sensors that send data to the team leader. CIMS of each team leader made three concurrent connections to the back-end system and sent totally approximate 17KB every 5 seconds. We conducted the tests in which from one team to four teams send data simultaneously, and measured the average execution time in 5 minutes run. Still in this scenario, all devices connect to the back-end through the designated router Dell D620. Figure 8 shows the performance of transferring data from team leader devices to the back-end system. Overall, we observed that the performance of CIMSs in PDAs is different. These behaviors need to be examined in more detail by analyzing how the back-end system handles requests from teams. The transfer time also increased during the test because the eXist database had to handle more XML documents in the same collection.

All performance data presented are average values determined from various runs. We observed there is a high variation between different runs in PDAs. For example, when measuring parallel data transfer with 3 connections with 20 KB, the fastest transfer time is 190 ms whereas the slowest transfer time is 5170 ms. We also observed that storing the whole big XML document in SCIMS is much faster than updating small XML data

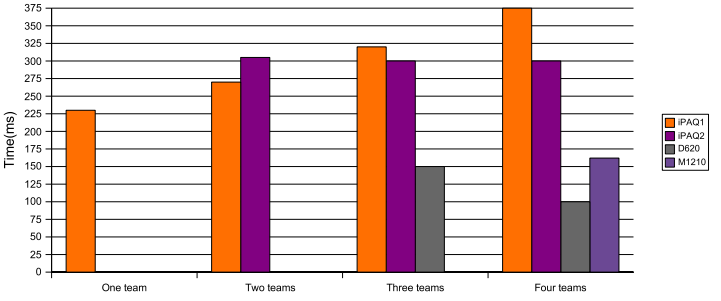

Fig. 8. Performance of transferring data to the back-end system

fragment into an existing XML document using XUpdate when the framework stored context information into separate XML documents and all provenance information into a single XML document. It means that it would be better to store multiple small XML documents than merge them into a big one. Since the framework focuses on bringing data to the back-end system quickly we changed the way we stored provenance information. Instead of storing all provenance information into a single document, we store provenance information in separate files.

7.4 Search for Relevant Context Information

As context information is described in XML, relevant context information can be searched by using XQuery. We outline few examples in the following.

Team A wants to reach to a place "P". Let's check unusable roads lead to "P": this can be done by the team leader when deciding to move the team to a new place. Another team might approach "P" before and notice unusable roads leading to be "P". The following query can be used to find out roads which are unusable.

```
for $infrastructure in collection('/db/contextinformation')//includeInfrastructure
where $infrastructure/category="ROAD" and $infrastructure/condition="UNUSABLE"
return $infrastructure
```

Let's send one worker to place "P" to take a photo: This can be decided by the team leader when she/he needs to send some support workers immediately to one area, e.g., for taking a photo needed for a further analysis. The team leader, for example, can obtain the location information of the support workers, and depending on this information, the leader can assign the task to those people who are near to that place. Similarly the team leader can also look for the activities that the support workers are performing and assign the new task to those who are doing activities with less priority. We can retrieve this information from team level context by using the following query:

```
for $worker in collection('db/contextinformation')//SupportWorker
where $worker//hasCamera and $worker/belongsTo/description="Team 1"
and $worker//Activity/status="LOW"
return $worker
```

8 Discussion of Existing Issues

One of the main issues about context information management is the quality aspects of context information such as incompleteness, duplication and inconsistency. For example, considering the case of context information aggregation. Since the team leader will pull and fuse the latest snapshot from members, it is possible that the team snapshots will miss some information when more information sent to a member than retrieved from the member. However, if we reduce the polling interval, the team snapshots may include duplicate context information. This issue is well known and many methods have been proposed to address it. However, context information is collected by using mobile devices which do not have enough capabilities to conduct these methods. Therefore, in our framework, checking quality of context information could be implemented as a plug-in for the CIMS.

Another issue is the data aggregation at CIMS. Each context management service retrieves various types of context information from different sensors and clients. The information may follow the same model (e.g., in the example of the WORKPAD project) but in practice, different sensors provide different information fragments at different times. Therefore, not only the context management service has to manage multiple fragments of information but also we cannot put the information fragments into a coherent view even they follow the same model. When context information does not follow the same model, CIMS cannot merge information fragments. However, in case context information follows the same model, we can merge information fragments into a single one. We can merge data fragments received in a predefined windows time into a single one, for example based on approximate XML joins [13] or using XUpdate/XSLT[32]. In doing so, we could define plug-ins for CIMSs. However, this might be applicable only to high-end mobile devices as we observed performance issues in updating XML documents in Section 7.3.

Since context information is gathered from various places by various teams, it is important to automatically process context information. We could define rules based on which context information can be evaluated and corresponding actions can be performed based on the evaluation of context information, e.g., inform relevant parties about new emerging issues. To this end, we can apply rule-based systems, event condition action (ECA) and complex event processing (CEP) concepts to the CIMS and the SCIMS.

9 Conclusion and Future Work

Adapting tasks in emergency situations is an important and required feature because both human teams and processes involved in disaster scenarios are established in a dynamic manner and are changed on-demand, depending on specific situations. In this paper, we have presented a novel, generic framework for managing and providing context information in emergency situations, such as natural disasters. We have developed a P2P context management framework that is able to support multiple levels of context information such as individual, team, site, response and situation. We have presented how context information management services in ESCAPE help managing and providing context information to teams at both front-end and back-end sites of the emergency

scenarios. We also presented performance analysis of the system and outlined ESCAPE functions for the disaster response management in the EU WORKPAD project. The ESCAPE context management services are based on a SOA model and support XML-based context information. Thus they can be easy used and integrated into emergency situation support systems.

We have not presented different application scenarios. Currently, applications utilizing our framework, such as adaptive process management and GIS-based disaster management systems, are being developed. Our future work is to fully achieve the prototype implementation by addressing issues mentioned in Section 8 and focusing on adaptive aspects within the framework. Moreover, we are working on utilizing context information for adapting processes in disaster responses.

References

1. AWARENESS - Context AWARE mobile NEtworks and ServiceS, http://www.freeband.nl/project.cfm?id=494&language=en
2. EU OASIS project - Open Advanced System for dISaster and emergency management, http://www.oasis-fp6.org
3. EU ORCHESTRA project, http://www.eu-orchestra.org/
4. JCAF - The Java Context-Awareness Framework, http://www.daimi.au.dk/~bardram/jcaf/
5. RCSM Middleware Research Project, http://dpse.asu.edu/rcsm
6. Xerces CDC, http://imbert.matthieu.free.fr/jgroups-cdc/files/xerces-cdc.jar
7. Apache Xerces, http://xerces.apache.org/
8. Baldauf, M., Dustdar, S., Rosenberg, F.: A survey on context-aware systems. International Journal of Ad-Hoc and Ubiquitous Computing (January 2006)
9. Bardram, J.E.: Applications of context-aware computing in hospital work: examples and design principles. In: SAC 2004, pp. 1574–1579. ACM Press, New York (2004)
10. Bardram, J.E.: The java context awareness framework (jcaf) - a service infrastructure and programming framework for context-aware applications. In: Gellersen, H.-W., Want, R., Schmidt, A. (eds.) PERVASIVE 2005. LNCS, vol. 3468, pp. 98–115. Springer, Heidelberg (2005)
11. Bellavista, P., Corradi, A., Montanari, R., Stefanelli, C.: Context-aware middleware for resource management in the wireless internet. IEEE Trans. Software Eng. 29(12), 1086–1099 (2003)
12. eXist XML database, http://exist.sourceforge.net/
13. Guha, S., Jagadish, H.V., Koudas, N., Srivastava, D., Yu, T.: Approximate xml joins. In: SIGMOD 2002. Proceedings of the 2002 ACM SIGMOD international conference on Management of data, pp. 287–298. ACM Press, New York (2002)
14. Hackmann, G., Haitjema, M., Gill, C.D., Roman, G.-C.: Sliver: A bpel workflow process execution engine for mobile devices. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 503–508. Springer, Heidelberg (2006)
15. Henricksen, K., Indulska, J., McFadden, T., Balasubramaniam, S.: Middleware for distributed context-aware systems. In: Meersman, R., Tari, Z., Hacid, M.-S., Mylopoulos, J., Pernici, B., Babaoglu, Ö., Jacobsen, H.-A., Loyall, J.P., Kifer, M., Spaccapietra, S. (eds.) On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE. LNCS, vol. 3760, pp. 846–863. Springer, Heidelberg (2005)

16. Hess, C.K., Román, M., Campbell, R.H.: Building applications for ubiquitous computing environments. In: Mattern, F., Naghshineh, M. (eds.) Pervasive Computing. LNCS, vol. 2414, pp. 16–29. Springer, Heidelberg (2002)
17. Hingne, V., Joshi, A., Finin, T., Kargupta, H., Houstis, E.: Towards a pervasive grid. ipdps, 00:207b (2003)
18. jSLP, http://jslp.sourceforge.net/
19. kSOAP2, http://ksoap2.sourceforge.net/
20. Mantoro, T., Johnson, C.: Location history in a low-cost context awareness environment. In: ACSW Frontiers 2003: Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003, Darlinghurst, Australia. Australian Computer Society, Inc., pp. 153–158 (2003)
21. Meissen, U., Pfennigschmidt, S., Voisard, A., Wahnfried, T.: Context- and situation-awareness in information logistics. In: Lindner, W., Mesiti, M., Türker, C., Tzitzikas, Y., Vakali, A.I. (eds.) EDBT 2004. LNCS, vol. 3268, pp. 335–344. Springer, Heidelberg (2004)
22. Miller, N., Judd, G., Hengartner, U., Gandon, F., Steenkiste, P., Meng, I-H., Feng, M.-W., Sadeh, N.: Context-aware computing using a shared contextual information service. In: Ferscha, A., Mattern, F. (eds.) PERVASIVE 2004. LNCS, vol. 3001, Springer, Heidelberg (2004)
23. MXQuery, http://www.dbis.ethz.ch/research/current_projects/MXQuery
24. Naguib, H., Coulouris, G., Mitchell, S.: Middleware support for context-aware multimedia applications. In: Zielinski, K., Geihs, K., Laurentowski, A. (eds.) DAIS. IFIP Conference Proceedings, vol. 198, pp. 9–22. Kluwer, Dordrecht (2001)
25. Ranganathan, A., Campbell, R.H., Ravi, A., Mahajan, A.: Conchat: A context-aware chat program. IEEE Pervasive Computing 1(3), 51–57 (2002)
26. Román, M., Hess, C.K., Cerqueira, R., Ranganathan, A., Campbell, R.H., Nahrstedt, K.: Gaia: A middleware infrastructure to enable active spaces. In: Mattern, F., Naghshineh, M. (eds.) Pervasive Computing. LNCS, vol. 2414, pp. 74–83. Springer, Heidelberg (2002)
27. Service Location Protocol (SLP), http://tools.ietf.org/html/rfc2608
28. Solarski, M., Strick, L., Motonaga, K., Noda, C., Kellerer, W.: Flexible middleware support for future mobile services and their context-aware adaptation. In: Aagesen, F.A., Anutariya, C., Wuwongse, V. (eds.) INTELLCOMM 2004. LNCS, vol. 3283, pp. 281–292. Springer, Heidelberg (2004)
29. The EU WORKPAD Project, http://www.workpad-project.eu
30. Voida, S., Mynatt, E.D., MacIntyre, B., Corso, G.M.: Integrating virtual and physical context to support knowledge workers. IEEE Pervasive Computing 1(3), 73–79 (2002)
31. XQuery, http://www.w3.org/TR/xquery/
32. XSL Transformations (XSLT), http://www.w3.org/TR/xslt
33. XUpdate, http://xmldb-org.sourceforge.net/xupdate/
34. Yan, H., Selker, T.: Context-aware office assistant. In: IUI 2000: Proceedings of the 5th international conference on Intelligent user interfaces, pp. 276–279. ACM Press, New York (2000)
35. Yau, S.S., Karim, F.: A context-sensitive middleware for dynamic integration of mobile devices with network infrastructures. J. Parallel Distrib. Comput. 64(2), 301–317 (2004)