2009 International Conference on Advanced Information Networking and Applications Workshops

Quality Aware Context Information Aggregation System for Pervasive Environments

Atif Manzoor, Hong-Linh Truong, Schahram Dustdar
Distributed Systems Group, Vienna University of Technology
{manzoor, truong, dustdar}@infosys.tuwien.ac.at

Abstract

Sensing context information and making it available to the people, involved in coordinating a collaborative task, is a preliminary phase in making a system adaptable to the prevailing situation in pervasive environments. However, the diversity of the sources of context information, the characteristics of pervasive environments, and the nature of collaborative tasks pose a stern challenge to the efficient management of context information by sensing a lot of redundant and conflicting information. Quality of Context parameters can be used to resolve the conflicts in context information. In this paper, we present a context aggregation system that detects and removes the duplicates and conflicts from context information by using the policies based on Quality of Context parameters. This system effectively aggregates the continuously evolving context information and efficiently uses the scare resources in pervasive environments.

1. Introduction

Context information plays a vital role in coordinating the collaborative tasks and adapting them to the prevailing situation in mobile and pervasive environments. Such collaborative tasks include performing rescue activities in disaster response, coordinating a sport festival, or arranging a carnival. In these situations, information is typically gathered from a variety of sources ranging from the sensors pre-installed in the environment to the sensors embedded in the applications on the devices carried by the participants of these tasks [1]. In most cases, information is evolved continuously and uninterrupted updates are received from the sensors [2]. We consider the cases in which more than one sensors collect the information about the same entity in the environment, giving rise to the redundant and conflicting information. For example, sensors embedded in the user interface application of the mobile devices of different disaster response workers that perform activities on the same site can send inconsistent information about that site. Because of these circumstances context information is not only voluminous but also most likely contains duplicate and conflicting context objects that unnecessarily use the limited resources of mobile devices.

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

The Nature of task in these emerging situations poses more challenges on devising a conflict resolution strategy. Different types of information have different significance in different circumstances, e.g., the information about the victims and the infrastructure is more critical than the information about the profile of the personnel participating in the rescue activities to carry on the relief work and to make decisions in coordinating disaster response. Single technique to resolve the conflict among all types of context information is insufficient in those cases. Simple conflict resolution techniques such as add, min, and average can also decrease the quality of context information. So far, there has not been enough research to address these issues. A mechanism that effectively resolves the conflicts in continuously evolving context information to make it consistent and coherent and efficiently uses the scare resources in pervasive environments is indispensable in such situations.

Quality of Context(QoC), defined as “any information that describes the quality of information that is used as context information” [3], can play an important role in resolving the conflicts in the context information [4]. In this paper, we present a mechanism that will detect the duplicate and conflicting information from the continuously updating context information. This system will also dynamically select and use a conflict resolving policy, based on QoC parameters, to resolve the conflicts. We have also described the implementation of the system and discussed its evaluation.

The rest of the paper is organized as follows. Section 2 gives an overview of related work. Section 3 gives an account of our context monitoring and management framework. Section 4 discusses our context information aggregation system. Section 5 describes the implementation and the evaluation of our system. Finally Section 6 presents the conclusion and the future work.

2. Related Work

Along with smart and pervasive environments, data aggregation has also been a popular research topic in sensor networks. Typical applications for the sensor networks are object tracking, surveillance, and environmental monitoring [5]. Data in these applications is usually handled using a simple data structure, e.g., numerical value. The main emphasis is on finding different data routing protocols for in-network data aggregation and very simple aggregation functions such as count, min, max, sum, average, and median are applied on data [6]. Comparatively, pervasive environments have a wider range of applications such as performing collaborative work. Hence, complex data structures are used to gather data from sources ranging from the simple senors to user interfaces and applications in mobile devices. Therefore data aggregation techniques from sensor networks are not sufficient for context aggregation in mobile and pervasive environments. In remaining section, we present a comparison of existing techniques for context aggregation in pervasive environments with our work.

OASIS [7] is a framework for context information integration and presentation based on ontologies. But it had not considered the scenario of resolving the conflicts in context information. In [8], they used the conflict resolving policy to delete the conflicting objects with smaller value of a measure, relative frequency, that is based on the time of the generation of that context object. In our work, we have used more sophisticated policies to resolve the conflict among context objects. These policies are based on trustworthiness, completeness, and significance of context object along with its up-to-datedness. In XMIDDLE [9], simple conflict resolution techniques, such as, add, last, random, first, and greatest are considered to reconcile the data entered about the same entity by different hosts. As compared to conflict resolution techniques used in this work, we will also be using the trustworthiness of a specific source to report about an event and QoC parameters to resolve the conflict in information.

MoGATU [10] was presented as a framework for data management in pervasive computing environments. Main emphasis is on processing and routing the queries to get the information. This system has not discussed the scenarios in which duplicate or conflicting information can be provided by more than one sources in the environment. In [11], a layered model for using context management has been presented. This model has used the age of context information to decide about the validity of the context information. However, age of context information is not enough for context data management in more dynamic pervasive environments, e.g., if a new context information is received from a source that have lower value of trustworthiness as compared to the source of already available information, this system will delete the context information from more trusted source. Such situations are considered in our system by using more dynamic conflict resolving policies.

3. Context Monitoring and Management Framework

In this section, we give an overview of context monitoring and management framework(CMMF) as shown in Figure 1. We have extended our CMMF presented in Escape [12] by adding context aggregation system based on QoC information. We consider three type of nodes in the environment. These nodes are context producer, context consumer and the nodes that act as both context consumer and context producer. Context producer directly gathers the context information from the environment, e.g., temperature sensors, device sensors, and software sensors embedded in the user interface and applications on mobile devices. Context consumers are the higher level applications, systems, and personnel involved in decision making that uses the context information. The third type of node act as an intermediary between context producers and context consumers, by collecting the context from low level context producers and providing it to high level context consumers. Context Monitoring and Management Framework(CMMF) works on these nodes to provide the mechanism to discover, query, subscribe, update, and provide the context information to the middleware. Attributes of the context information services are used to describe the information presented by that service in the middleware. Context consumers can discover, query, and subscribe to the context information of their interest. They are also notified when new information is gathered at the context producer. Context consumer can also query the required information in environment. Our context aggregation service works as the part of CMMF to effectively combine the context information provided by different sources. Removing the redundant and conflicting context information enables CMMF to efficiently use the scare resources of mobile devices and increase its efficacy to provide context information to interested applications and systems.

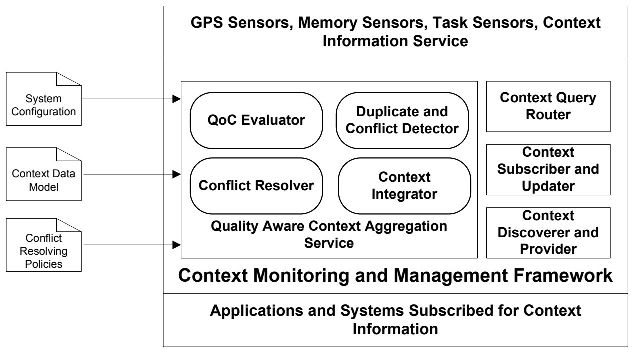
GPS Sensors, Memory Sensors, Task Sensors, Context Information Service
System Configuration
Context Data Model
Conflict Resolving Policies
QoC Evaluator
Duplicate and Conflict Detector
Conflict Resolver
Context Integrator
Quality Aware Context Aggregation Service
Context Query Router
Context Subscriber and Updater
Context Discoverer and Provider
Context Monitoring and Management Framework
Applications and Systems Subscribed for Context Information

Figure 1. System Components of Context Monitoring and Management Framework

4. Context Aggregation System

Context aggregation system, shown as a part of CMMF in Figure 1, works both locally and globally on a node. Locally, it aggregates all context information received at a node and globally it aggregates the context information from lower level nodes before forwarding it to the higher level nodes,

```
node.role=teamLeader
hierarchicalValue.max=4
hierarchicalValue.min=1
node.hierarchicalValue=2
device.type=pda
criticalValue.max=5
criticalValue.min=1
infrastructure.criticalValue=3
Infrastructure.totalAttributeCount=4
infrastructure.lifetime=60 minutes
infrastructure.source=userInterface
```

Figure 2. System configuration file sample

in case of context information queries from those higher level nodes. This system takes the *System Configuration*, *Context Data Model*, and *Conflict Resolving Policies* as the input to the system. *QoC Evaluator*, *Duplicate and Conflict Detector*, *Conflict Resolver*, and *Context Integrator* are the components of the system. In the following section, we will be describing them in detail.

4.1. System Configuration

As it have been described in the previous section, our system works on three type of nodes. These nodes range from the handheld devices with very limited resources to high power back end systems. Therefore, we need to change the behavior of system according to the requirements and the resources of the node. This information is provided to the system by the configuration file. For example, if our system is working on a PDA that is receiving context information from the sensors installed on that device only, it can be mentioned in the configuration file that the system should only search for duplicates. It should not unnecessarily consume the processing power in looking for the conflicting context objects, as it is rare chance if information is being gathered only from the same sensors. Information related to the concepts in the context data model that is used to evaluate the QoC parameters is also provided in the system configuration. Figure 2 shows a sample configuration file that describes that the person on this node is working as the team leader and is carrying a PDA device.

4.2. Context Information Model

Our context aggregation system takes the context information model as an input to the system. In context information model, every context object contains the information about the identification of the source that gathered the context object, identification of the entity about which that context object is gathered, time at which that information is gathered, and the context information itself. The information about the identity of source, entity, and timestamp will be used to find duplicate and conflicting context objects. Figure 3 shows the XML representation of a context object of type

```
<Infrastructure sourceID = "UIAX00065"
                entityID = "Square000X38"
                timestamp = "1219668617937"
                name = "MainSquare"
                Usability = "70%">
                <QoCParameters  uptodatedness="0.83311665"
                    trustworthiness="0.6333333"
                    completeness="0.94446003"
                    significance="1.0" />
</Infrastructure>
```

Figure 3. XML representation of a context object of type Infrastructure

infrastructure, named *MainSquare*, that has the value of *sourceID*, *entityID*, *timestamp* and the context information about the *usability* of that *infrastructure*. This information is sensed by a user interface sensor on a rescue worker's device performing task in response to a flood in the city. Context object can also be a simple value gathered from a low level sensor or it can consist of other context objects. For example in a disaster response, context object representing the context information about a site of disaster will consist of the context object representing workers, buildings, and victims. So that context object $\mathcal{O}$ can also be represented as the collections of constituent context objects such as $\mathcal{O} = \{\mathcal{O}_1 \cup \mathcal{O}_2 \cup \mathcal{O}_3 \cup .... \cup \mathcal{O}_n\}$. Context information services in the environment will be providing context objects according to this model to the context aggregation system to combine these newly arrived context objects with existing information.

4.3. QoC Evaluator

QoC Evaluator evaluates the QoC parameters for the context information that is contained by a context object and annotates that context object with those evaluated QoC parameters. We have described the mechanism of the evaluation of QoC parameters from QoC sources in our previous work [4]. Context object is passed to this component for the evaluation of QoC parameters. This component also gathers the necessary information about the source of information and entity about which that context information is gathered from the system configuration file. This system configuration file is provided by the user of the

```
infrastructure.reslovingPolicy=combinedQuality
infrastructure.threshold=.8
rescueWorker.reslovingPolicy=trustworthiness
rescueWorker.threshold=1.0
team.reslovingPolicy=trustworthiness
team.threshold=1.0
resource.reslovingPolicy=uptodatedness
resource.threshold=.9
victims.reslovingPolicy=significance
victims.threshold=.7
```

Figure 4. Conflict resolving policies file sample

system along with the context information model file. QoC sources such as *SourceLocation*, *InformationEntitLocation*, *MeasurementTime*, *SourceState* are used to evaluate the QoC parameters: *up-to-datedness*, *trust-worthiness*, *completeness*, and *significance* of context object. These QoC parameters have values in the range [0..1] and play an important role in resolving the conflict in context objects. Figure 3 shows the context object of type infrastructure annotated with QoC parameters. Configuration file in Figure 2 is also showing the information about life time, critical value, total number of attributes, and source of context object of type *infrastructure* that is used to evaluate these QoC parameters.

4.4. Resolving Policies

Our system uses the conflict resolving policies that are defined on the basis of QoC parameters. These policies can depend on a single QoC parameter such as *up-to-datedness*, *trust-worthiness*, *completeness*, and *significance* or a combination of them such as *combinedQuality*. If the resolving policy for a specific type of context object is specified as *trust-worthiness*, it will mean that if we have to resolve the conflict between the two context objects of this type, we will discard the context object with lower value of *trust-worthiness* and store the context object with higher value. Similarly if the resolving policy for a type of context object is defined as *combinedQuality*, it will mean that we will take the average of all QoC parameters to resolve the conflict between conflicting objects of that type. Sometimes for very critical information it is useful to keep the conflicting information in the system and let the user decide about it. In that case a threshold value can also be set, so that, all the context objects having the quality higher than that threshold value are kept in the context store. In case of numerical data, e.g., room temperature, conflict resolving policies like addition, minimum, maximum, or average can also be mentioned.

Policies related to all the concepts in the context information model are mentioned in conflict resolving policies file and passed to the system. Figure 4 shows a sample of conflict resolving policies file that have been provided to the system to resolve the conflicts in the context objects in the case of disaster response activities. When the system detects a conflict in a pair of context objects, first of all system checks the type of context information that is presented by that context information. Secondly, it looks into the conflict resolving policies file to find the policy that is defined in that file to resolve the conflicts for that type of context objects. After that system gets the values of QoC parameters that have been used to enforce that policy. For example, in Figure 4 *combinedQuality* with the threshold value of .8 has been specified as the conflict resolving policy for context objects presenting the context information of type *infrastructure*. When the system detects a pair of conflicting context objects of type *infrastructure*, it will get the value of *combinedQuality*, i.e., the average of all QoC parameters, for that context object. Context objects having the value of *combinedQuality* more than .8 are only kept in the context store. All other context objects are discarded from the context store.

Algorithm 1 Algorithm to detect duplicate and conflicting context objects

INPUT: New arrived context object CO

```
1: get the entityID of CO
2: **if** There exists a context object with same entityID **then**
3:    **if** sourceID of both context objects match **then**
4:       **if** timestamp of both context objects match **then**
5:          $Discard\ context\ object$
6:       **else**
7:          $call\ function\ ResolveConflict\ by\ passing$
            $both\ context\ objects$
8:       **end if**
9:    **else**
10:      $call\ function\ ResolveConflict\ by\ passing$
         $both\ context\ objects$
11:   **end if**
12: **else**
13:   $Add\ the\ context\ object\ to\ contextstore$
14: **end if**
```

4.5. Duplicate and Conflict Detector

Newly arrived context object is passed to Algorithm 1 to check if it is the duplicate of or in conflict with any existing context object. First we get the identifier of the entity represented by this newly arrived context object and check if there is any context object in the existing data representing the same entity. If we do not find any context object representing the same entity, it means that we do not have any duplicate or conflicting context object and newly arrived context object is added to existing context store. If there is some context object representing the same entity, then we check the sources of context objects. If they have different sources then they are identified as the conflicting context objects and are passed to Algorithm 2, to resolve the conflict. If these two context objects are from the same source then we check the time when these context objects are generated. If they have the same timestamp then it means that they are the exact duplicate of each other and anyone of them can be discarded and the other one is kept in the context store. If they have the different timestamps it means that these are the conflicting context objects that are generated at different instance of time. This type of pair is also passed to the Algorithm 2.

4.6. Conflict Resolver and Integrator

Algorithm 2 receives the pair of conflicting context objects and is responsible for resolving the conflict among those conflict context objects and add them to the context store. Pairs of conflicting context objects are passed to this algorithm in two cases. First, when the conflicting objects are generated from the same source and second, when the context objects are generated from different sources. Conflict resolving policies, discussed in Section 4.4, for each type of context object are specified in the configuration file. For example, Figure 2 shows the sample configuration file in which conflict resolving policy for the concept of type *infrastructure* is specified as *combinedQuality*. Finally, after discarding duplicate and conflicting context objects, context objects are added in context store by the context integrator. This context information is provided to the applications or systems that have subscribed for that information.

Algorithm 2 Algorithm to resolve conflicts

```
INPUT: Conflicting context objects
 1: if sourceID matches then
 2:     $get\ conflict\ resolving\ policy\ for\ that\ type$
 3:     $apply\ conflict\ resolving\ policy$
 4:     $add\ the\ context\ object\ that\ satify\ the\ policy\ to\ context\ store$
 5: else
 6:     $call\ QoCEvaluator\ to\ get\ QoCParameters$
 7:     $get\ conflict\ resolving\ policy\ for\ that\ type$
 8:     $apply\ conflict\ resolving\ policy$
 9:     $add\ the\ context\ object\ that\ satify\ the\ policy\ to\ context\ store$
10: end if
```

5. Implementation and Evaluation

We have used Java2 ME (CDC 1.1 profile) for the development of our quality aware context aggregation system prototype that has extended our Escape framework [12] as described in Section 3. We have also used RESCUE [13] as the underlying system to provide the support for the development of Web services middleware on mobile devices. System configuration and conflict resolving policies files are implemented as key-value pair as shown in Figure 2 and Figure 4. Context information model, designed to manage the context information in disaster response, provided to the system as XML schema. Context information was also stored as XML elements. Figure 3 shows a context object, annotated with QoC parameters, describing the usability of a square in the city. In the remaining section we will discuss the detail of our experiment and evaluation of our system.

We had simulated an environment that represented a team, consisting of five workers. That team was performing the rescue activities in a city that was affected by flood. Workers were also collecting the context information about the usability of an infrastructure named *MainSquare* as represented by the context object shown in Figure 3 and providing this information to the middleware as context information services. Team leader had subscribed to get this information and was receiving continuous updates from all the workers. Context aggregation system at the team leader received this information to effectively combine the recently received context objects with the existing context. The team leader had specified the conflict resolving policy as the *combinedQuality*. In this policy, quality is represented by the average value of all the QoC parameters. QoC parameters were normalized to have the value in the range [0..1] and average of all QoC parameters also lied in the same range. Team leader also set the threshold value for *conbinedQuality*. This threshold value was considered as the minimum value of *conbinedQuality* for a context object to add into the context store. All the context objects having value less than threshold value were discarded.

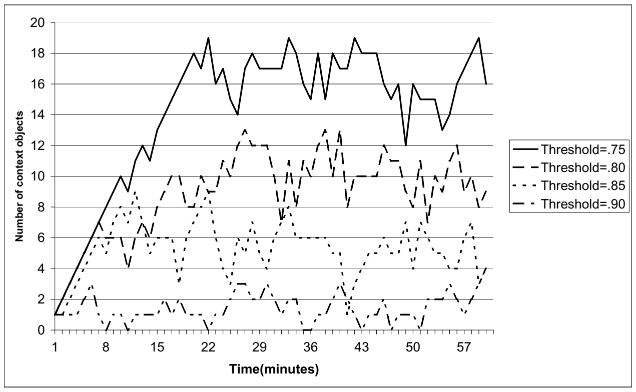

Figure 5. Number of context objects contained in context data store

Figure 5 shows the number of context objects stored in context store along with time line. Team leader had set the different threshold values of *combinedQuality* in four different cases as show in Figure 5. Number of context objects saved in context store were different with different values of threshold. It shows that the less number of context objects are kept in the context store when high value of quality threshold is specified. The users can select the value of quality threshold and he can receive the context objects according to the his requirements. Less number of context objects with higher quality decrease the burden on the team leader to make analysis of context information and take decision to adopt according to the current situation.

Figure 6 shows the number of context objects that have been deleted from the context store in a given period of time because they did not meet the quality criteria that have been set by the team leader. We had observed that

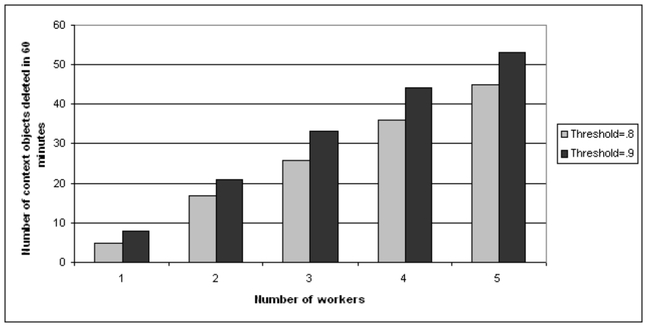

Figure 6. Number of context objects deleted

the number of deleted context objects, i.e., the context objects with context information of low quality, increased with increase in number of workers in a team for a given quality criteria. So teams with larger number of workers poses more difficulties for the team leader to decide about the context object providing correct context information. Our system improve the performance by deleting the context objects of lower quality and presenting the team leader with limited number of context objects of higher quality to make decisions.

6. Conclusion and Future Work

In this paper we presented the context aggregation system that uses QoC parameters to resolve the conflict among the context objects representing the same context entity. We have also presented an algorithm to detect duplicate and conflicting context objects. Our context aggregation system presents context objects which have the data quality more than specified and saves the people from unnecessary burden to decide about quality of data and they can concentrate on decision making process. Our context aggregation system makes an efficient use of scare resources by deleting the duplicate and redundant context objects.

For future work, we plan to emphasize on defining and evaluating more sophisticated QoC parameters and framing the conflict resolving policies on the basis of those QoC parameters. We will try to minimize the role of user profiled values in the evaluation of QoC parameters and to enhance the quality of context information by combining the context information and QoC parameters from more than one context object. Along with context aggregation, we also plan to use QoC parameters in other tasks needed to perform for monitoring and management of context information in mobile and pervasive environments , e.g., context query routing and decision making.

References

[1] K. Henricksen and J. Indulska, "Modelling and using imperfect context information," in *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops(PERCOMW '04)*, March 2004, pp. 33–37.

[2] M. Balazinska, A. Deshpande, M. Franklin, P. Gibbons, J. Gray, S. Nath, M. Hansen, M. Liebhold, A. Szalay, and V. Tao, "Data management in the worldwide sensor web," *Pervasive Computing, IEEE*, vol. 6, no. 2, pp. 30–40, April-June 2007.

[3] T. Buchholz, A. Küpper, and M. Schiffers, "Quality of context: What it is and why we need it," in *Proceedings of the 10th International Workshop of the HP OpenView University Association(HPOVUA)*, vol. 2003. Hewlet-Packard OpenView University Association, 2003.

[4] A. Manzoor, H. L. Truong, and S. Dustdar, "On the evaluation of quality of context," in *EuroSSC*, ser. Lecture Notes in Computer Science, vol. 5279. Springer, 2008, pp. 140–153.

[5] K. Romer and F. Mattern, "The design space of wireless sensor networks," *Wireless Communications, IEEE*, vol. 11, no. 6, pp. 54–61, Dec 2004.

[6] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi, "In-network aggregation techniques for wireless sensor networks: a survey," *Wireless Communications, IEEE [see also IEEE Personal Communications]*, vol. 14, no. 2, pp. 70–87, April 2007.

[7] J. M. Serrano, J. Serrat, S. van der Meer, and M. O. Foghlu, "Ontology-based management for context integration in pervasive services operations," in *Inter-Domain Management*, vol. 4543/2007. Springer Berlin / Heidelberg, 2007, pp. 35–48.

[8] Y. Bu, T. Gu, X. Tao, J. Li, S. Chen, and J. Lu, "Managing quality of context in pervasive computing," in *QSIC '06: Proceedings of the Sixth International Conference on Quality Software*. IEEE Computer Society, 2006, pp. 193–200.

[9] C. Mascolo, L. Capra, S. Zachariadis, and W. Emmerich, "Xmiddle: a data-sharing middleware for mobile computing," *Int. Journal on Personal and Wireless Communications*, vol. 21, pp. 77–103, 2002.

[10] F. Perich, A. Joshi, T. Finin, and Y. Yesha, "On data management in pervasive computing environments," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 16, no. 5, pp. 621–634, May 2004.

[11] A. Schmidt, "A layered model for user context management with controlled aging and imperfection handling," in *Modeling and Retrieval of Context. Proceedings of the 2nd International Workshop on Modeling and Retrieval of Context MRC 2005*. Springer, 2005, pp. 86–100.

[12] H. L. Truong, L. Juszczyk, A. Manzoor, and S. Dustdar, "Escape - an adaptive framework for managing and providing context information in emergency situations," in *EuroSSC*, ser. Lecture Notes in Computer Science, vol. 4793. Springer, 2007, pp. 207–222.

[13] L. Juszczyk and S. Dustdar, "A middleware for service-oriented communication in mobile disaster response environments," in *6th International Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC). 9th Middleware Conference*. ACM/IFIP/USENIX, 2 December 2008.