Using Quality of Context to Resolve Conflicts in Context-Aware Systems *

Atif Manzoor, Hong-Linh Truong, and Schahram Dustdar
{manzoor, truong, dustdar}@infosys.tuwien.ac.at

Distributed Systems Group, Vienna University of Technology

Abstract. Context-aware systems in mobile and pervasive environments face many conflicting situations while collecting sensor data, processing sensor data to extract consistent and coherent high level context information, and disseminating that context information to assist in making decisions to adapt to the continuously evolving situations without diverting human attention. These conflicting situations pose stern challenges to the design and development of context-aware systems by making it extremely complicated and error-prone. Quality of Context parameters can be used to cope with these challenges. In this paper, we discuss the conflicting situations that a context-aware system may face at different layers of its conceptual design and present the conflict resolving policies that are defined on the basis of the Quality of Context parameters. We also illustrate how these policies can be used in different conflicting situations to improve the performance and effectiveness of context-aware systems.

1 Introduction

The vision of pervasive environments is characterized with a plethora of computation and communication enabled sensing devices that are embedded in our daily-life objects. The main objective of these devices is to facilitate user by working as a smart assistant for them. Many research efforts have been undertaken to fulfill these requirements and context management system frameworks are divided in different conceptual layers. These conceptual layers are assigned the task of collecting raw sensor data, extracting high level context information from this data, aggregating and storing context information after eliminating redundant and inconsistent context, providing this information to interested applications and users, and finally applications and users take actions to adapt themselves to this information as described in [1].

Different conflicting situations can arise during the execution of the aforementioned tasks. These conflicting situations strongly affect the capability of context-aware systems to adapt to the evolving situation in pervasive environments. Earlier systems have used some simple strategies such as drop all, drop last, drop first [24], involved user to resolve conflicts [7], or do the mediation on the basis of some predefined static policies [17]. These strategies may slow down the process of decision making, distract user, or discard some important context objects as well. Moreover, context conflicts cannot

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

Conceptual framework layers [1]	*Conflicts*	*Examples*
Context Acquisition	Conflicts in making selection among different sensors using different techniques to collect context [5, 6]	GPS and GSM collect location information of a mobile user with different level of accuracy
Processing	Conflicts in extracting high level context information [8, 21, 22]	Sensor data shows that someone is present in two different locations
Context Distribution	Conflicts in context information aggregation [18]	Redundant and inconsistent data reaching a node from different routs
Application	Conflicting interests of applications [17]	Different preferences set by two users present in living room

Table 1. Layers of the conceptual framework of a context management system and conflicts that can arise on those layers

be resolved at design time [4] and need a strategy that can dynamically handle them at runtime without distracting user.

Quality of context (QoC) is defined as "any information that describes the quality of information that is used as context information" [3] can be used to devise the policies to resolve the conflicts at different layers of a conceptual framework of context-aware system as shown in Table 1. Later QoC is also defined as "any inherent information that describes context information and can be used to determine the worth of information for a specific application" [13]. In [14], we have classified QoC in QoC parameters and QoC sources. QoC parameters, such as up-to-datedness, trust-worthiness, completeness, and significance, are used to indicate the quality of context information. QoC sources like source location, measurement time, source state, and source category are used to evaluate those QoC parameters. In this paper we analyze generic conflicting situations that can occur at different layers of a context-aware system and propose the conflict resolving policies based on the quality of context parameters. We also present how these conflict resolving policies can be used and describe the prototype implementation of our system that have used these policies. We have performed the experiments to evaluate these policies. We observed those policies that used the combination of different QoC parameters considering the perspective of the use of context information are more effective.

The rest of the paper is organized as follows: Section 2 discusses the conflicts that a context management system can face at different layers of its conceptual architecture. Section 3 presents the policies that we have defined to resolve the conflicts discussed in Section 2. The implementation detail of our Quality-Aware Context Management Framework is discussed in Section 4. We have presented the experiments and evaluation of our system in Section 5. Section 6 gives an overview of related work and compares them with our approach. Finally, we conclude the paper and discuss our future work in Section 7.

2 Conflicting Situations in Context Management Systems

In this section we discuss the conflicts that can take place at different layers of context management system as presented in [1] and how these conflicts can affect the performance of context-aware applications. Table 1 provides the summery of these conflicts.

2.1 Context Acquisition

In pervasive environments the volume of data generated by sensors makes analysis of context impossible for a human [6]. Sensor data may also differ with each other considering the frequency of updating context, the capability of a sensor to collect the context of an entity, the accuracy of a method that is used by sensors, representation format, and the price of context information [5, 6]. For example, location information of a mobile user can be gathered using GPS and GSM methods. Problems also arise due to the mobility of sensors along with entities in pervasive environments. We cannot permanently rank a sensor to collect the context of a particular entity. So there is a need of a strategy that can dynamically decide which sensor is more reliable to collect the context of a certain entity at some specific time. QoC parameters that have been dynamically evaluated from the information about the source of context can be used to resolve the conflicts in such situation.

2.2 Processing

In processing layer, high level context is extracted from low level sensor data. Sensor data cannot be presented directly to applications. It needs to be filtered, fused, correlated, and translated to extract the higher level context data and detect the emergent events [22]. Some works, such as presented in [21], have made the supposition that sensors along with producing data can also estimate about its reliability and self-confidence. These metrics are used to do the reasoning to extract high level context information. In [8], advertised probability of correctness of context sources is used to do the reasoning to extract single piece of context information by combining the information from different providers. QoC parameters that provide information about up-to-datedness, trustworthiness, significance and completeness can replace those metrics and make the reasoning on data more meaningful and realistic to resolve conflicts.

2.3 Context Distribution

The high mobility of sensors, unreliable wireless connections, and the nature of tasks in pervasive environments result in the acquisition of a lot of redundant and conflicting context. This redundant and conflicting context not only results in the wastage of scare resources but also can lead to undesired behavior of context-aware applications. Simple conflict resolving policies, such as drop first, drop all, can result in deleting some valuable information. In critical situation, such as a context-aware ubiquitous home for patients [12] and telehealth applications [11], loss of information can result in severe situations for the people using it. Decision can better be made to discard or keep a context object on the basis of policies defined using these QoC parameters.

QoC Parameters	*Equations for the evaluation of QoC Parameters for context object O*
Up-to-datedness	$\begin{cases} 1 - \frac{Age(\mathcal{O})}{Lifetime(\mathcal{O})} & : \quad if \;\; Age(\mathcal{O}) < Lifetime(\mathcal{O}) \\ 0 & : \quad otherwise \end{cases}$
Trustworthiness	$\begin{cases} (1 - \frac{d(\mathcal{S},\mathcal{E})}{d_{max}}) * \delta & : \quad if \;\; d(\mathcal{S},\mathcal{E}) < d_{max} \\ undefined & : \quad otherwise \end{cases}$
Completeness	$\begin{cases} \frac{\sum_{j=0}^{m} w_j(\mathcal{O})}{\sum_{i=0}^{n} w_i(\mathcal{O})} & : \quad if \; m \; and \; n \; are \; finite \\ 0 & : \quad otherwise \end{cases}$
Significance	$\frac{CV(\mathcal{O})}{CV_{max}(\mathcal{O})}$

Table 2. Some of the QoC parameters defined and evaluated in [14]

2.4 Application

Context-aware applications use context information to adapt their behavior to user needs and changes in the environment. If conflicts are not resolved in context information at the earlier stages, the applications that take actions on the basis of that context information get in conflict while making decisions. Context-aware applications can also get in conflicts due to different priorities set by users. Different strategies are used to resolve their behavior as presented in [17]. Information about the up-to-datedness, trustworthiness, completeness, and significance of context information make it easy to resolve conflicts and make decisions on the basis of that context information.

3 QoC Based Conflict Resolving Policies

The main consideration of these policies is to resolve the conflicts in such a way that the decision should have been taken in the favor of context object that contains the context information of the highest quality. This quality of context information is characterized by QoC parameters. Table 2 shows the equations that have been used to evaluate these QoC parameters in range [0..1] as described in [14]. If the user of these policies want more than one context object that have quality higher than a specific value then he can specify the threshold value in range [0..1] and all the context objects that have quality higher than that threshold value are selected. We have also taken into account the user centered design of context-aware systems and tried that human users of the system should not be distracted during the execution of these policies. In this section we discuss the fundamental policies based on different QoC parameters.

3.1 Up-to-datedness Based Policy

Up-to-datedness indicates the degree of rationalism to use a context object at a specific instance of time. We have calculated up-to-datedness of a context object as the ratio between the age of that context object and the lifetime of the type of context information contained by that context object. This metric can be useful in resolving the conflict in the context object that changes its value very rapidly, e.g., location of a fast moving vehicle. In this case, it will be more suitable to use the context object with the highest value of up-to-datedness. Whereas, up-to-datedness will not have much role in the case of conflict in static information that have been profiled in the system, e.g., information about the structure of a smart home.

3.2 Trustworthiness Based Policy

Trustworthiness is the degree of the suitability of a sensor to collect the context of a specific type. We have calculated the trustworthiness of a context object on the concept of space resolution and accuracy of sensor to measure that type of information. This concept is particularly useful in resolving the conflict when we have more than one sensor collecting the context of same entity or event. For example, we have temperature sensors at different places in the living room of a smart home that is built to provide comfortable life to old people. The sensors that are installed near the electric radiator heater will be sending the higher value of the temperature of living room as compared to the sensors in the other places in the living room. To provide a comfortable temperature in the room we will be more relying on the readings of the sensors, that are closer to the sitting area than the sensors in the far off corners of the living room and sensor near the radiator.

3.3 Completeness Based Policy

The completeness of context information indicates that all the aspects of context information have been presented by a context object. We have evaluated completeness of a context object as the ratio of the sum of the weights of available attributes of context object to the sum of the weights of total number of attributes of context object. Completeness of a context object is particularly important to get the complete picture of the current situation of real world. According to this policy decision is made on the basis of that context object which has more complete information about current situation.

3.4 Significance Based Policy

Significance measures the worth or preciousness of context object. It is particularly important to mention this metric when there is a context object of high critical value. For example, if smoke sensors detect heavy smoke in bedroom, it will be the information of high significance. This metric can be used to generate events that need prompt actions from the applications. Applications can specify that the context objects with high value of significance should be reported on priority basis.

Apart from the above mentioned fundamental policies, policies can also be defined based on two or more QoC parameters depending on the requirements of a particular application. For example, a policy can also be defined by combining QoC parameters, such as up-to-datedness and trustworthiness. In such policy an average value of the mentioned QoC parameters is used to make decisions. For example, if a context aggregator uses a policy based on the combination of the up-to-datedness of a context object with the threshold value of 0.8, then all the context objects having the average of the value of up-to-datedness and the value of trustworthiness more than 0.8 will be selected. Users of conflict resolving policies set threshold value according to their requirements considering the perspective of the use of context information.

4 Implementation

Figure 1 shows the components of our context management system corresponding to the conceptual framework layers and data flow among those components. Components used to evaluate and annotate QoC parameters and conflict resolving policies can be used with any system component. *QoC Evaluator* receives context objects as XML elements and evaluates QoC parameters for those context objects. QoC parameters are normalized to have values in range [0..1] and *Context Annotator* annotates context objects with those QoC parameters as we had presented in [14]. Figure 2 shows a context object annotated with QoC parameters. QoC parameters along with QoC based conflict resolving policies are used to resolve conflicts. Guidelines to select a policy in different conflict resolving situations provided to the system as an input file. For example, it can be mentioned that the selection among different sources of a particular type of context information should have been done on the basis of the combination of trustworthiness and up-to-datedness based policies.

We developed our prototype as the part of the implementation of the EU project WORKPAD [20] and was based on COSINE [10]. Java2 ME (CDC 1.1 profile) used for its development to make it able to run on mobile devices. Our context information model, to manage the context information in disaster response, was designed as XML schema. Context information also stored as XML elements. MXQuery and KSOAP2, having a low memory foot print to be able to run on mobile devices, were used for processing XML data. In this prototype, we dealt with high level context information. The components dealing with low level context, such as context fusion and high level context extractor, were not implemented yet.

5 Experiments and Evaluation

In our simulated environment, a team of five workers was performing rescue activities in response to a flood in a city. Those workers were randomly moving on the flood site and after every minute they were sending a context object to their team leader. This context object was of type infrastructure and contained information about the usability of a square in the city. Our context management system evaluated QoC parameters for that context object and annotated that context object with those QoC parameters as shown in Figure 2. We had applied QoC based conflict resolving policies on different layers

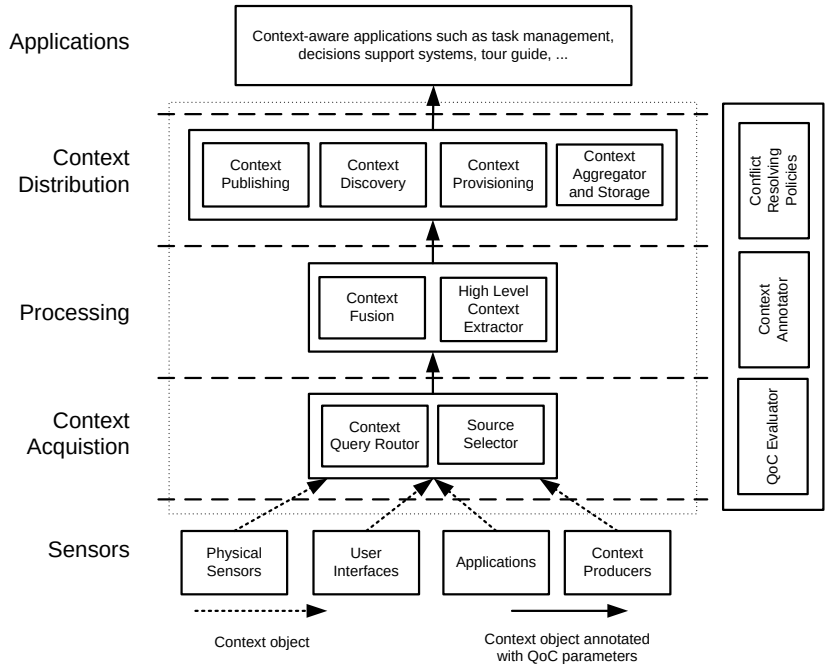

Fig. 1. Components of QCMF corresponding to different layers

to resolve conflicts that occurred while performing functions on different layers of our context management system. While applying those policies we have not considered the fact that those policies had already been applied to the underlying layers or not.

In the first case we applied conflict resolving policies at context acquisition layer to resolve conflict in making selection among different sources that were sending aforementioned context objects. We used the conflict resolving policies based on up-to-datedness, trustworthiness, and a combination of up-to-datedness and trustworthiness. The threshold value for QoC-parameters has been specified as 0.9. Thus, all the sources of context information that are producing the context objects that have the value of QoC parameters more than threshold had been selected. Figure 3 shows the number of context objects received in 60 minutes from the selected sources of context information with increase in number of workers. As in our simulated environment every source of context information is generating context objects after a fixed interval of one minute. The number of context objects having value of up-to-datedness more than specified value increases with increase in sources of context objects. As a result up-to-datedness based conflict resolving policy did not seem to be useful in making source selection in scenarios where every source of context information was generating context objects after a specific interval of time, e.g., sensor networks. Policies based on trustworthiness of the source of context object and combination of up-to-datedness and trustworthiness proved to be more useful for context data acquisition layer as mentioned in Table 3.

```
<Infrastructure sourceID = "UIAX00065"
 entityID = "Square000X38"
 timestamp = "12196686179 37"
 name = "MainSquare"
 location = "CityCenter"
 Usability = "70%">
 <QoCParameters>
  <uptodatedness >0.83</uptodatedness>
  <trustworthiness >0.63</trustworthiness>
  <completeness >0.94</completeness>
  <significance >1.0</significance>
 </QoCParameters>
</Infrastructure>
```

Fig. 2. XML representation of context object of type Infrastructure

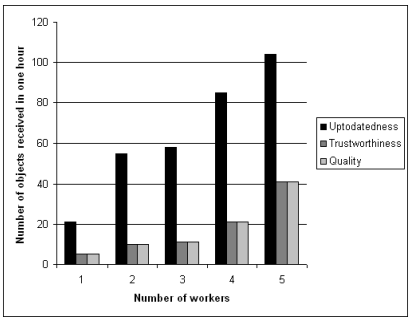

Fig. 3. Source selection using different QoC-based conflict resolving policies

In the second case we applied the conflict resolving policies at context distribution layer to make efficient use of context store by context aggregator. Context aggregator was receiving context objects from five workers that were sending the data after random intervals of less than one minute. Context aggregator was initiating a cleaning service after every minute and conflicting context objects that did not meet the specified policy criteria were deleted. Conflict resolving policies based on up-to-datedness, trustworthiness, and quality, i.e., the combination of up-to-datedness and trustworthiness, has been used and the threshold value of 0.85 has been set for those policies. Figure 4 shows the number of context objects that were currently stored in context store using aforementioned policies and threshold value. As it is apparent from Figure 4, using the policy based on trustworthiness did not prove to be very useful as some context objects that have been captured long time ago still have higher value of trustworthiness and are uselessly kept in the context store. Using up-to-datedness based policy proved to be same as keeping latest context objects and it has deleted the old context objects that can result in loss of some important context information. Finally, we have used a quality policy based on the combination of both up-to-datedness and trustworthiness to detect useless context objects. With this policy we had not only been able to detect more number of useless context objects but also kept the context objects of high trustworthiness that are highly valued in making any decision on the basis of context information. From this observation we had concluded that the conflict resolving policy based on the combination of more than one QoC parameters is more effective in resolving conflicts particularly in context aggregation and in general in performing the functions on context distribution layer, as we have mentioned in Table 3.

In the final case conflict resolving policies applied at the context distribution layer to generate the events of interest for various subscribed applications, as it is nearly impossible for a human to analyze those context objects. Firstly, we only used the significance of context information to generate the events of interest as shown in Figure 5. We observed that only considering the significance of context objects is not sufficient to generate the events of interest. Therefore, we combined the trustworthiness of the source of context information with significance and found it quite useful to assist decision making in performing functions at context distribution layer as shown in Table 3.

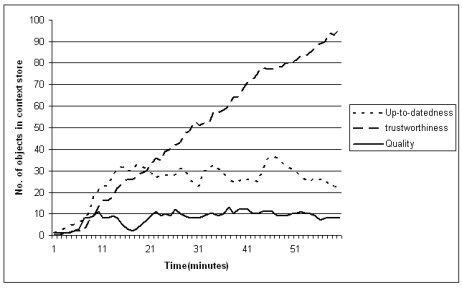

Fig. 4. Context aggregation using different QoC-based conflict resolving policies

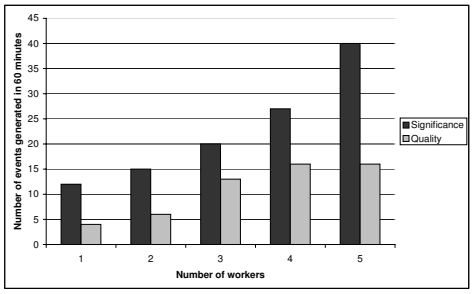

Fig. 5. Events generated with help of QoC-based policies

Layers / Policy	*Context Acquisition*	*Processing*	*Context Distribution*	*Application*
up-to-datedness based policy	+ +	+ +	+ +	+ + +
trustworthiness based policy	+ + +	+ +	+ +	+ + +
completeness based policy	+ +	+ +	+ +	+ + +
significance based policy	+	+ +	+ +	+ + +
combinations based policy	+ + +	+ + +	+ + +	+ + +

Table 3. Table showing importance of different policies at different layers

The overall behavior of our simulated experiments show that any QoC parameter alone is not sufficient to perform the task of decision making on any layer of context-aware systems. QoC parameters used in combination of two or more parameters are more effective to perform this functionality at different layers of context-aware systems. We have also observed that the value of QoC parameters for different applications merely depends on the need of that specific application as shown in Table 3. In our experiments we gave preference to context object with greater value of QoC parameters over context objects with lower value of QoC parameters. Their can also be the situations where more sophisticated reasoning is required to make decisions on the basis of these parameters. These reasoning can be made on the basis of probability theories such as Basian theory, Dempster-Shafer theory or on the basis of neural networks.

6 Related Work

Different policies have been defined in literature to resolve the conflicts in context-aware systems. Mostly these policies are based on involving the user in mediation process [7], resolving the conflicts by using some predefined static policies based on user

preferences [17], discarding all the conflicting context, discarding the last received, or discarding the first received context objects [24]. Some works have also used QoC parameters to perform different tasks in middleware solutions to manage context information. In the remaining section we will discuss about the works that have suggested different conflict resolving policies and have used QoC parameters to perform different tasks in context-aware systems.

6.1 Conflict Resolving Policies

Xu et al. in [24] presented an impact-oriented automatic resolution of pervasive context inconsistency. Trying all possible resolution policies to find one that brings the least impact on the context-awareness of applications is referred as impact-oriented resolution. Conflict resolution scheme such as drop-latest, drop-all and drop-earliest are used in this work. But it had not presented the procedure for the situation evaluation of a context-aware system to calculate the impact of a resolution strategy.

A model for managing context information and resolving the inconsistencies is presented by Bu et al. in [2]. This work has used the policy of discarding the context with low value of relative frequency in conflicting cases. Accepting new data and discarding the existing conflicting context object or rejecting the new data and keeping the old conflicting context object are used as accept and reject conflict resolving policies in [23]. But there have not been any assurance whether the new or old data is more reliable or not. Both these policies can result in discarding the important context information. Dey et al. [7] involved the user in mediation process to resolve ambiguity in context information.

In [17], Park et al. suggested to resolve conflicting situation between the applications by using a static policy, based on user preferences, that describes how conflicting applications need to adapt in case of conflicts between them. A configuration file gives a set of rules that specifies the behavior of applications in different contexts. These static policies may not comply to user needs in more dynamic and unknown environments, as Capra et al. argued in [4] context conflicts cannot be resolved at the design time and need to be resolved at the execution time.

6.2 Using QoC Parameters

Mihaila et al. [16] have identified four quality of data parameters: completeness, recency, frequency of updates, and granularity and have used these parameters to make source selection and ranking on WWW. Sources publish these quality of data parameters using WS-XML and they propose a query language that exploits those quality of data parameters to make source selection and ranking on WWW. Chantzara et al. [5] have presented an approach that used quality of information for evaluating and selecting the information to be used as context information. They calculate a utility function based on QoC attributes.

Huebscher et al. [9] have also used QoC parameters in their adaptive middleware for context-aware applications in smart homes. They have used QoC parameters to perform different tasks in their middleware, such as context provider selection. In [15], we have used QoC parameters to detect and remove duplicate and conflicting information

and perform context aggregation. Sheikh et. al. [19] have also used QoC parameters to enforce privacy of a user. But all these works [5, 9, 16, 19] are based on the assumption that sources of information also provide information about the quality parameters. In this case sources of information can affect the decisions based on those quality parameters. In contrast to this approach, we evaluate QoC parameters in our quality-aware context information management framework that works as a middleware solution to provide context information to context-aware applications and users.

7 Conclusion and Future Work

In this paper we have discussed conflicting situations that can occur at different layers of a context management system. We have also presented the conflict resolving policies based on QoC parameters that can be used to resolve the conflicts in such situations. We have performed the experiments to evaluate the performance of different policies and observed that conflict resolving policies that are defined upon the combination of different QoC parameters considering the context of the use of context information by a specific application showed better performance. For our next steps, we plan to use these policies to do more sophisticated reasoning in the fusion of low level context and extraction of high level context information. We also plan to enhance the quality of context information by combining the context information and QoC parameters from more than one context objects.

References

1. Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *Int. J. Ad Hoc Ubiquitous Comput.*, 2(4):263–277, 2007.
2. Yingyi Bu, Tao Gu, Xianping Tao, Jun Li, Shaxun Chen, and Jian Lu. Managing quality of context in pervasive computing. In *QSIC '06: Proceedings of the Sixth International Conference on Quality Software*, pages 193–200. IEEE Computer Society, 2006.
3. Thomas Buchholz, Axel Küpper, and Michael 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)*. Hewlet-Packard OpenView University Association, 2003.
4. Licia Capra, Wolfgang Emmerich, and Cecilia Mascolo. Carisma: Context-aware reflective middleware system for mobile applications. *IEEE Transactions on Software Engineering*, 29:929–945, 2003.
5. Maria Chantzara, Miltiades Anagnostou, and Efstathios Sykas. Designing a quality-aware discovery mechanism for acquiring context information. In *AINA '06: Proceedings of the 20th International Conference on Advanced Information Networking and Applications - Volume 1 (AINA'06)*, pages 211–216. IEEE Computer Society, 2006.
6. Diane J. Cook. Making sense of sensor data. *IEEE Pervasive Computing*, 6(2):105–108, 2007.
7. Anind K. Dey and Jennifer Mankoff. Designing mediation for context-aware applications. *ACM Trans. Comput.-Hum. Interact.*, 12(1):53–80, 2005.
8. Marcus C. Huebscher, Julie A. McCann, and Naranker Dulay. Fusing multiple sources of context data of the same context type. In *Proceedings of the 2006 International Conference on Hybrid Information Technology*, pages 406–415. IEEE Computer Society, 2006.

9. Markus C. Huebscher and Julie A. McCann. Adaptive middleware for context-aware applications in smart-homes. In *MPAC '04: Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing*, pages 111–116, New York, NY, USA, 2004. ACM.
10. Lukasz Juszczyk, Harald Psaier, Atif Manzoor, and Schahram Dustdar. Adaptive query routing on distributed context - the cosine framework. In *International Workshop on the Role of Services, Ontologies, and Context in Mobile Environments (ROSOC-M), 10th International Conference on Mobile Data Management (MDM'09), May 18-20, Taipeh, Taiwan*. IEEE Computer Society, 2009.
11. Nadjia Kara and O. Andrei Dragoi. Reasoning with contextual data in telehealth applications. In *Proceedings of the Third IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*. IEEE Computer Society, 2007.
12. Younghee Kim and Keumsuk Lee. A quality measurement method of context information in ubiquitous environments. In *ICHIT '06: Proceedings of the 2006 International Conference on Hybrid Information Technology*, pages 576–581. IEEE Computer Society, 2006.
13. Michael Krause and Iris Hochstatter. Challenges in modeling and using quality of context (qoc). In *Proceedings of Mobility Aware Technologies and Applications*, volume 3744/2005, pages 324–333. Springer Berlin / Heidelberg, 2005.
14. Atif Manzoor, Hong Linh Truong, and Schahram Dustdar. On the evaluation of quality of context. In *Proceedings of Third European Conference on Smart Sensing and Context, EuroSSC*, volume 5279, pages 140–153. Springer, 2008.
15. Atif Manzoor, Hong Linh Truong, and Schahram Dustdar. Quality aware context information aggregation system for pervasive environments. In *The 5th International Symposium on Web and Mobile Information Services, The IEEE 23rd International Conference on Advanced Information Networking and Applications (AINA-09), Bradford, UK, May 26-29*. IEEE Computer Society, 2009.
16. George A. Mihaila, Louiqa Raschid, and Mara esther Vidal. Using quality of data metadata for source selection and ranking. In *Vossen (Eds.), Proceedings of the Third International Workshop on the Web and Databases, WebDB 2000, Adams Mark Hotel*, pages 93–98, 2000.
17. Insuk Park, Dongman Lee, and Soon J. Hyun. A dynamic context-conflict management scheme for group-aware ubiquitous computing environments. In *COMPSAC '05: Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05) Volume 1*, pages 359–364. IEEE Computer Society, 2005.
18. Filip Perich, Anupam Joshi, Timothy W. Finin, and Yelena Yesha. On data management in pervasive computing environments. *IEEE Trans. Knowl. Data Eng.*, 16(5):621–634, 2004.
19. Kamran Sheikh, Maarten Wegdam, and Marten van Suinderen. Quality-of-context and its use for protecting privacy in context aware systems. *Journal of Software*, Volume 3:83–93, 2008.
20. The EU WORKPAD Project. http://www.workpad-project.eu.
21. Huadong Wu, Mel Siegel, and Sevim Ablay. Sensor fusion using dempster-shafer theory ii: static weighting and kalman filter-like dynamic weighting. In *Proceedings of the 20th IEEE Instrumentation and Measurement Technology Conference, 2003*, pages 907–912, 2003.
22. Alex Wun, Milenko Petrovi, and Hans-Arno Jacobsen. A system for semantic data fusion in sensor networks. In *DEBS '07: Proceedings of the 2007 inaugural international conference on Distributed event-based systems*, pages 75–79, New York, NY, USA, 2007. ACM.
23. Chang Xu and S. C. Cheung. Inconsistency detection and resolution for context-aware middleware support. In *Proceedings of the 10th European software engineering conference*, pages 336–345. ACM, 2005.
24. Chang Xu, S. C. Cheung, W. K. Chan, and Chunyang Ye. On impact-oriented automatic resolution of pervasive context inconsistency. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 569–572. ACM, 2007.