A Framework for Quality of Context Management

Zied Abid, Sophie Chabridon, and Denis Conan

Institut TELECOM, CNRS UMR Samovar
9 rue Charles Fourier, 91011 Évry, France
Firstname.Lastname@institut-telecom.fr

Abstract. Context-aware computing has to deal with a huge amount of context data. Taking into account the quality of these data becomes a corner stone of an efficient context management solution. Information on the quality of context helps taking appropriate decisions and allows to identify uncertain context information saving processing time for deriving a pertinent description of the observed phenomenon.
This paper presents a work in progress for integrating Quality of Context in COSMOS (COntext entitieS coMpositiOn and Sharing) [4,13], a component-based framework for managing context data in ubiquitous environments, and illustrates it throughout the example of the composition of context information to implement a *network connectivity vs energy* adaptation situation.

Key words: context-aware computing, quality of context, component-based middleware.

1 Introduction

With the proliferation of wireless connectable devices, the environment can be enriched with sensors acquiring a huge amount of context data that is to be analysed by computing systems. We consider context as being "any information that can be used to characterize the situation of entities (*i.e.* whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves"[7]. Context-aware computing allows to detect specific conditions requiring some adaptation actions. This calls for context integration and context abstraction methods. Context integration concerns the extraction of the most accurate context from a number of noisy and conflicting contexts. Context abstraction, or context reasoning, allows to derive a higher-level application-relevant context from a number of lower-level context data [15]. Taking into account the quality of context data becomes a corner stone of an efficient context management solution and has given rise to a large number of research works over the past decade. The importance of the quality of context as such for context-aware computing has first been raised by [3]. This concept has then been refined with a notion of worth: quality of context is any inherent information that describes context information and can be used to determine the worth of the information for a

specific application [10]. Information on the quality of context helps taking appropriate decisions and allows to identify uncertain context information saving processing time for deriving a pertinent description of the observed phenomenon.

This paper proposes to integrate Quality of Context in COSMOS[1] (*COntext entitieS coMpositiOn and Sharing*), which is our framework for managing context data in ubiquitous environments [4,13]. The COSMOS framework relies on the FRACTAL[2] component-based middleware [2]. FRACTAL presents some original features among which two are important to us: recursivity and component sharing. Recursivity (which has given its name to FRACTAL) allows components to be nested within composite components. With sharing, a given component instance can be included (or shared) by more than one component, saving memory and other system resources. COSMOS then provides the concepts of context node and context management policies translated into FRACTAL software components.

COSMOS reorganises the classical functionalities of a context manager to systematically introduce a 3-steps cycle of data collection, data interpretation, and situation identification. Although situation identification actions should not be too frequent, processing context information is an activity that must be conducted more often, while data gathering is a third activity that must be continuous. Thus, we have three different activities with different frequencies. We decouple as much as possible these activities in order to obtain a non-blocking and usable framework.

QoC is supported in COSMOS through the notion of QoC operator that can integrate various kinds of QoC parameter operators, dedicated to a particular QoC parameter *e.g.* Up-to-Dateness. This approach is generic and our framework can easily be extended by adding new operators. Moreover, we propose several modes to transmit QoC. The QoC data can be communicated either as metadata in a context report or separately. This contributes to provide a flexible framework that can be adapted to the requirements of various applications.

This paper is organised as follows. Section 2 presents the design of our COSMOS framework. Section 3 details the way we propose to integrate Quality of Context in COSMOS. Section 4 presents the case study of a *network connectivity vs energy* adaptation situation. Section 5 positions our work with respect to other research dealing with QoC in context management. Finally, Section 6 concludes this paper and identifies some perspectives.

2 Presentation of COSMOS

This section presents the foundations of our work by summarising the principles of the COSMOS framework. We present the basic building units for composing of a context policy, that is context nodes.

[1] http://picoforge.int-evry.fr/projects/svn/cosmos/
[2] http://fractal.ow2.org

Concepts and properties of a context node The basic structuring concept of COSMOS is the *context node* [4] which is a context information modelled by a software component. Context nodes are organised into hierarchies to form context management policies. Context nodes possess some properties which define their behaviour with respect to the context management policy. A context node can be *passive* or *active*. An active node is equipped with an activity to execute a given task. Communication into the hierarchy of context nodes may be bottom-up (*notification*) or top-down (*observation*). Observation (or notification) reports are messages formed of sub-messages and typed chunks. For instance, the information on the WIFI bit rate is stored in a chunk of type WifiBitRateChunk. A context node which receives data transmitted by a notification or an observation may be *blocking* or *non-blocking*. Non-blocking nodes propagate observations and notifications. Blocking nodes stop the traversal: for observations, the most up-to-date context information is transmitted without polling child nodes, and for notifications, context data is used to update the state of the node but parent nodes are not notified. COSMOS allows all kinds of combinations in the properties of context nodes (active/passive, observation/notification, blocking/non-blocking). This makes it possible to tune very precisely the level of computing resources used and to balance it with the requirements of applications.

COSMOS provides the developer with pre-defined generic context operators. They are organised following a typology: *Elementary operators* for collecting raw data, *memory operators*, such as averagers requiring a history of values or translation operators, *data mergers*, *abstract or inference operators*, such as additioners or thresholds operators. The only programming is in the context operators. Following the component-based software development principles, with a sufficiently large library of context operators, there should be no programming at all, but only declarative composition of context nodes.

Architecture of a context node Each context node extends the abstract composite ContextNode depicted in Figure 1. The interfaces Pull and Push are the interfaces for the observation and the notification, respectively. The abstract composite ContextNode contains at least one operator (ContextOperator primitive component) as well as the message and activity managers. The Message Manager is in charge of handling the observation and notification reports which are sent and received by the component on the Pull and Push interfaces. The Activity Manager provides the support for dealing with active components. The Child (or children) is optional and can be a composite or primitive ContextOperator component. Child, Message Manager and Activity Manager components can be shared with other components, saving computing resources.

Pattern-oriented architecture of COSMOS For mapping context policies to context node hierarchies, COSMOS follows well-known design patterns [8], *Factory method*, *Composite*, *Flyweight*, and *Singleton*, enabling a scalable, extensible and efficient architecture [13].

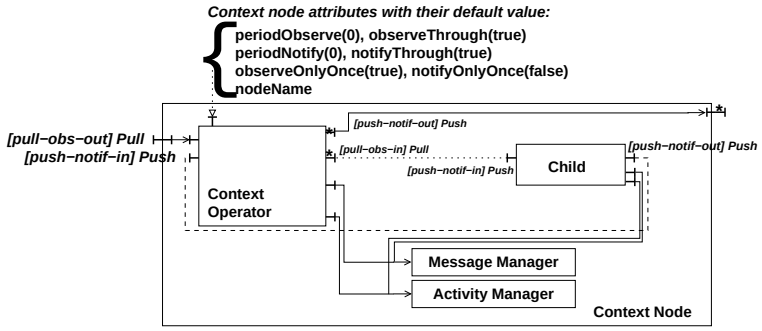

Fig. 1: Core architecture of a ContextNode component.

3 Quality of Context management

We present in this section the way we propose to manage quality of context within the COSMOS framework. We detail the component based architecture that allows us to compute QoC parameters and to integrate them into the messages transmitted from context sources to the application.

3.1 Integrating QoC in COSMOS

In order to have a flexible framework, we propose three modes to transmit context information. The first two modes deal with QoC information while the last mode ignores it and allows to transmit context information without QoC. As shown in the case 1 of Figure 2, the first mode consists in injecting QoC information as meta-data into the context information itself before sending it to upper layers. This mode is useful to filter context according to a particular policy: for instance, no context message is sent if the *completeness* parameter is less than 50 %. The second mode sends QoC information independently from any context information in a separate message (cf. Figure 2-2). This mode enables to supervise the QoC of the system, with a limited overhead as only QoC data is computed and extracted. The third mode allows to transmit context information with standard child and/or parent components that cannot deal with QoC (cf. Figure 2-3). This mode is proposed to remove the cost of managing QoC information when this additional information is not necessary. It also ensures ascending compatibility with applications developed with previous versions of COSMOS not supporting QoC.

3.2 Architecture

We define a QoC ContextNode as a COSMOS composite ContextNode (cf. Figure 1) responsible for computing QoC parameter values. It is composed of Context Collectors which are themselves ContextNodes and a QoC Operator. A

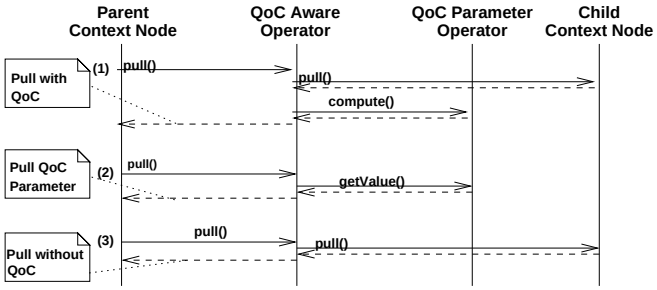

Fig. 2: Sequence diagram

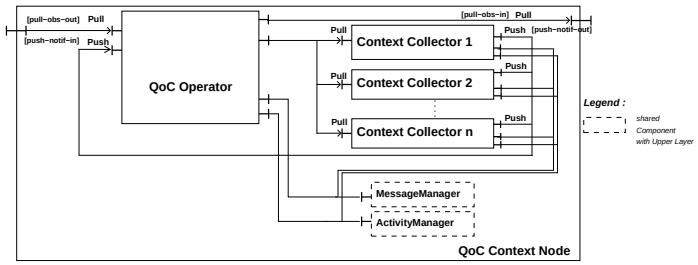

Fig. 3: QoC Context Node Architecture

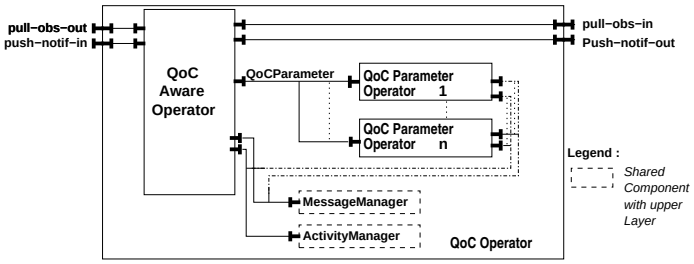

Fig. 4: QoC Operator Architecture

Context Collector collects raw meta-data coming from sensors or another part of the distributed system such as *Measurement Time*, *Source Location*, *Data accuracy* [12]. These raw meta-data are then transformed by the QoC Operator to deliver QoC parameters as shown below.

QoC Operator The QoC Operator is responsible for extracting required data, computing QoC and supplying it to upper layers via the Message Manager (cf. Figure 3). As shown by the inner architecture of a QoC Operator (cf. Fi-gure 4), raw meta-data coming from different Context Collectors get analysed by a QoC Aware Operator component which extracts relevant data and distributes them to

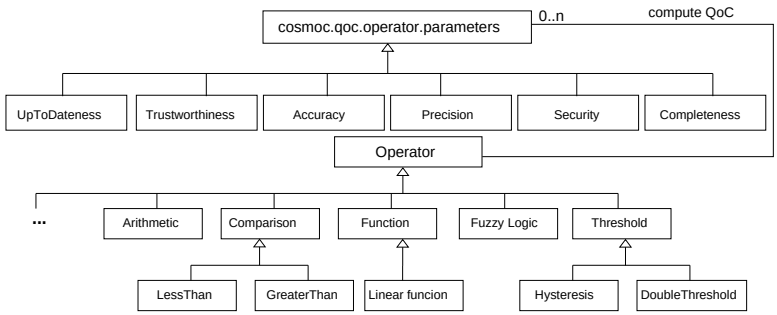

Fig. 5: Relation between QoC Parameters and COSMOS Operators

QoC Parameter Operator components. Each QoC Parameter Operator computes a specific QoC parameter such as accuracy, precision, up-to-dateness, etc.

QoC Parameter Operator The choice of the nature of the QoC Parameter Operator component depends on what type of QoC the application needs and what computing methods are available. We propose in Figure 5 a first list of operators used in most of context-aware applications but other operators can easily be added. We give as an example a way to compute the Up-to-Dateness QoC parameter with function $\mathcal{U}(\mathcal{O})$ as presented in [12]:

$$\mathcal{U}(\mathcal{O}) = \begin{cases} 1 - \dfrac{Age(\mathcal{O})}{Lifetime(\mathcal{O})} & \text{if } Age(\mathcal{O}) < Lifetime(\mathcal{O}) \\ 0 & \text{Otherwise} \end{cases}$$

where $Age(\mathcal{O}) = tcurr - tmeasure(\mathcal{O})$, with $tcurr$ representing the current time and $tmeasure(\mathcal{O})$ the measurement time of object O.

After having been computed, QoC parameters are forwarded to a QoC Aware Operator which is responsible for sending QoC information to upper layers.

QoC Aware Operator As introduced in Section 3.1, there are two modes to transmit QoC information that we now present in further details.

Adding QoC to context information Raw QoC information is transmitted to QoC Parameter Operators to be processed (cf. Figure 2-1). Afterwards, once computed, QoC values can be either added in an existing message chunk or, taking advantage of the flexibility of COSMOS, they can be placed into a new message chunk dedicated to common QoC information like timestamp. In this mode, all context information messages are enriched with QoC meta-data. This mode is useful for applications interested in the QoC at the same time as the context information itself, that is when QoC information is systematically required. As a consequence, QoC parameters are strongly related to this context. This results

in a more reliable and accurate analysis. However, this method requires time and resources to create a new message chunk or to update an existing one for each context information, which also increases the cost of the transmission of this information.

Sending QoC separately Periodically, or on request, only the QoC is transmitted to upper layers (cf. Figure 2-2). This mode is well suited for applications that do not require QoC information with a strong timing constraint and that can wait for the next periodic information. It also suits applications requiring that context information is passed as soon as possible without waiting for the next notification. This enables to supervise the QoC of the system, with a limited overhead as only QoC data is computed and extracted. In this mode, the QoC is sent at the request of the application (Pull mode) or as a periodic report (Push mode), so sending the QoC does not add any significant cost. One limitation is that QoC information is not strongly related to a specific context information instance. The QoC information sent may correspond to the last calculated QoC or to an average of the previous untransmitted values.

Regarding the ease of use of our framework, describing such an architecture with FractalADL [11] would be cumbersome and error-prone for users. So we have defined a first version of a domain specific language (DSL) for describing the composition of context nodes and context processors [13] that we will extend with QoC declarations.

4 Application scenario

In this section, we add QoC meta-data to an application scenario originally proposed for mobile commerce [13]. We consider a family shopping at a mall, each member of the family having a mobile device. This application allows them to share information, to consult product information, to download discount tickets, to be notified of advertisements, or to find the location of a product or a shop in the mall. The parents want their children to remain in the mall, with their devices connected as far as possible, so that everybody knows the location of the other family members. Nevertheless, a family member can disconnect for some periods of time in order to save their battery. The COSMOS context policy for this *network connectivity vs energy* scenario is shown in Figure 6. It involves different network technologies, such as Bluetooth or WIFI, and requires the application to adapt itself depending on network connectivity and context information availability. Each adaptation situation (in the upper part of Figure 6) is isolated in a context tree with the possibility of sharing sub-trees between policies.

We consider three QoC parameters in this scenario, *Trustworthiness*, *Up-to-Dateness* and *Precision* and detail below the way they are dealt with in our framework.

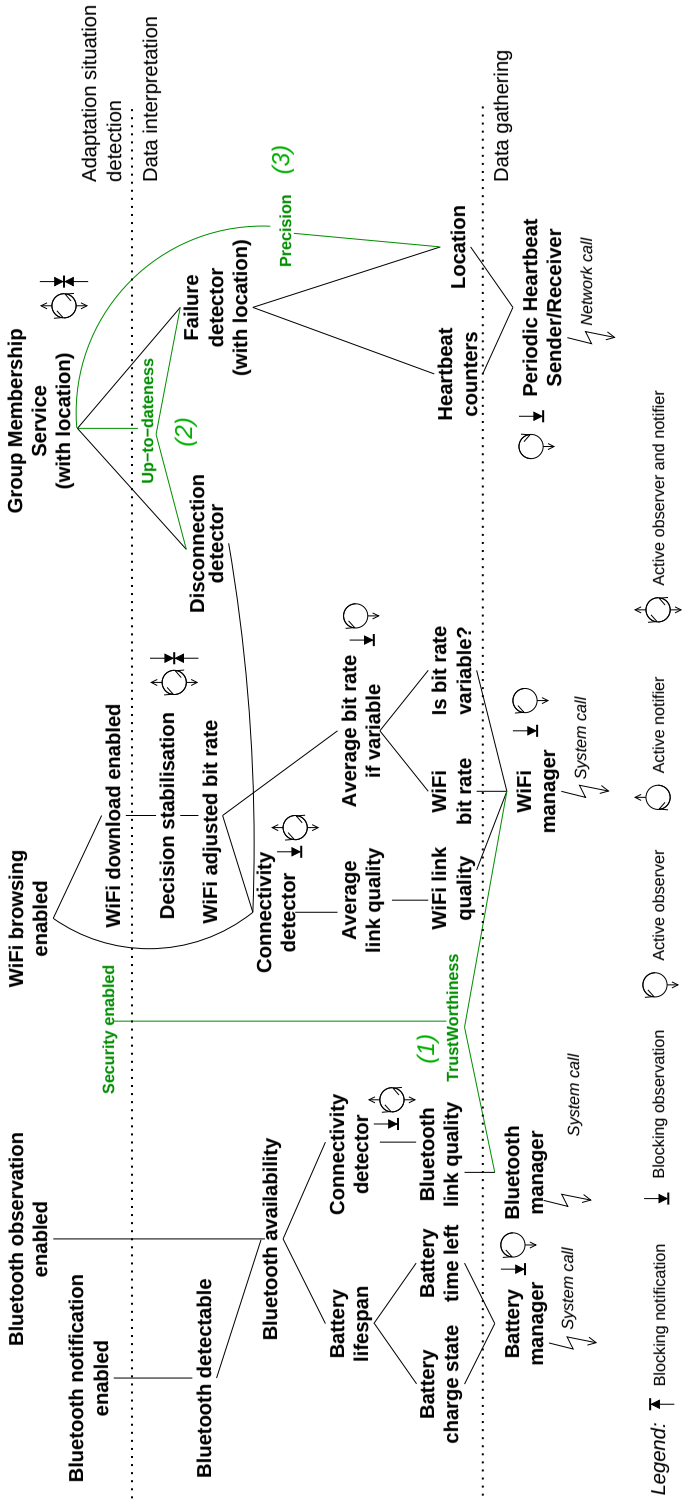

Fig. 6: Application scenario - Monitoring Network Connectivity and Battery Level

Trustworthiness Adding trustworthiness to this system is essential to measure how much a device can trust data coming from a connection even before testing the link quality of the connection or starting to download data. We consider that trustworthiness depends on the identity of the data sender which may correspond to a family device, a mall information source, a shop in the mall, or be unknown. A trustworthiness manager observes the WIFI and Bluetooth managers that are directly connected to the system. The sender id can be extracted as a MAC[3], IP or DNS address (for WIFI sources), device name or BD_ADDR[4] (for Bluetooth sources).

Here are some examples for trustworthiness values :

- 1 (100 %) *known device*: Value given to family devices. MAC or BD_ADDR address is unique and can be identified, therefore each family device can be registered into the application's configuration file.
- 0.5 (50 %) *verifiable device id*: Value given to mall or shops devices. These devices are trusted enough when their name or IP address can be verified (with the access provider of the mall for instance).
- 0 (0 %) *unknown device*: for all other devices without a specific information or not known at all.

In the data interpretation layer (cf. Figure 6-1), the trustworthiness is derived from the information coming from the *Bluetooth manager* and the *WIFI manager*. It then helps to decide whether to give direct access or not to incoming data to upper layers. Thus, context information has not to get up along the whole context node hierarchy.

Up-to-Dateness We propose to measure the freshness of the information coming from the *Disconnection detector* as well as from the *Failure detector* ((cf. Figure 6-2). As these data are critical, up-to-dateness may be used to optimize connection management for better result and also to save energy.

- If failure or disconnection's up-to-dateness value is high (the event just happened), the group membership service can try to reconnect to the concerned sender/receiver in order to continue the current action (send/receive).
- If failure or disconnection's up-to-dateness value is medium, new connections can be postponed for a laps of time and another communication mechanism is to be used.
- If failure or disconnection's up-to-dateness value is low, future connections to the concerned sender/receiver can be postponed or even canceled.

Precision In this example, the precision of the location information is used to adapt this information with appropriate display (cf. Figure 6-3). An application can have different map categories and precision degrees, and let the display manager choose the best map to promote location information.

[3] Media Access Control address
[4] Bluetooth Device Address

5 Related work

The work presented in this paper proposes a component-based middleware approach for context management taking into account Quality of Context as meta-data. In the design of our solution, we have taken a particular care of the performance issue in terms of the cost of observations and notifications. We favor a flexible architecture and try to save system resources in order to be able to reach a good level of scalability without degrading performance when the number of observed context sources and context processors becomes very large. In this section, we compare our work with other middleware frameworks for context management.

The Context Toolkit is one of the first middleware framework for context management. It is based on event programming and widget concepts introduced by GUI (Graphical User Interfaces) [7]. In the same framework, all the following functionalities are grouped: The interpreter for composing and abstracting context information, the aggregator for the mediation with the application, the service for controlling application actions performed on the context, and the discoverer that acts as a registry. Following the same philosophy, interpretation and aggregation functionalities have to be programmed in monolithic blocks: One interpreter and one aggregator per application, independently of the number of widgets and the level of abstraction requested by the application. This implies a lack of flexibility, that can impact performance and scalability. Moreover, the management of system resources consumed by context management treatments and, in particular, activities management, is not addressed. Concerning the quality of context, the authors of the Context Toolkit have considered means to deal with the uncertainty of context data through its level of accuracy. Three complementary approaches were proposed: passing ambiguity on to applications; attempting to disambiguate context automatically; and attempting to disambiguate context manually. Only the latter manual approach has been further investigated.

MoCoA provides an environment for building context-aware applications for ad hoc networks based on sentient objects [14]. The low-level inference treatments are organised as data merging pipes. MoCoA only allows notifications, contrary to COSMOS that adds observations. The pipes are logically enclosed in sentients objects, including the control of system resources' consumption. But, contrary to COSMOS, MoCoA neither details nor provides any means to externally specify these controls. Pipe treatments are complemented with inference ones with facts and rules. Sensor fusion is then used to manage the uncertainty of data captured from the real-world and to derive higher-level context information. Fusion can perform a sum, an average function or might rely on a Bayesian network. However, the quality of context data is not considered as a first-class concept and remains restricted to a single certainty value.

Contextors [5] are software entities similar to data components, and their meta-data (describing the data quality) as well as their controllers (modifying the configuration) are available for both inputs and outputs. A Contextor is a Java class that is associated to an XML descriptor. Thus, the software framework

builds, in an ad hoc manner, a container around the Contextor component. This ad hoc component model is implicit and not configurable (*e.g.* for managing system resources). For each Contextor using at least an activity, the local resource consumption can not be controlled. Furthermore, the sharing of context nodes supported by COSMOS is not addressed by Contextors. In addition, Contextors exchange control information in order to ask to stop or force the data notification for example. However, given that there is no explicit component model, it is impossible to introduce new configurations, such as some new attributes or control modes. In COSMOS, the structure and the life-cycle of components is finely managed by the FRACTAL controllers. One important aspect of the Contextor is the notion of data quality. Unlike to our solution, this quality meta-data can only be sent with the context data itself. We made the choice to provide different modes of transmission of QoC information for more flexibility and to allow to better tune the performance of the framework.

The list of quality parameters we consider is comparable to what is proposed in [12]. However, a specificity of our implementation framework is that it benefits from a component-based middleware following specific design patterns allowing to control very precisely resource consumption and performance.

6 Conclusion and future work

This paper presents a work in its early stage on Quality of Context management. We pay particular attention to performance and scalability issues by tailoring QoC management respectively to applications requirements and performance expectations. We are currently building a library of operators allowing to use combinations of rule-based and probabilistic solutions when appropriate in order to deal with uncertainty during the context abstraction process.

Regarding the ease of use of our framework, we have defined a first version of a domain specific language (DSL) for describing the composition of context nodes and context processors [13]. As future work, we intend to extend this DSL with QoC declarations.

Another research direction concerns the design of look-up mechanisms to find a child in the context hierarchy with a specific QoC level. Indeed, the component model we use is loosely typed, as mainly push and pull interfaces are defined. Therefore, our framework could benefit from a type system like Dream Types [1], allowing an operator to ask for a child node with a specific QoC type.

According to [6], specific probabilistic schemes are to be used at different abstraction levels. Depending on the amount of available knowledge, we envisage to experiment a method like FSI (Fuzzy Situation Inference) [9].

References

1. P. Bidinger, M. Leclercq, V. Quéma, A. Schmitt, and J.-B. Stefani. Dream Types: A Domain Specific Type System for Component-Based Message-Oriented Middleware. In *4th ESEC/FSE Workshop on Specification and Verification of Component-Based Systems*, Lisbon (Portugal), Sept. 2005.

2. É. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani. The Fractal Component Model and Its Support in Java. *Software—Practice and Experience, special issue on Experiences with Auto-adaptive and Reconfigurable Systems*, 36(11):1257–1284, Sept. 2006.
3. T. Buchholz, A. Kupper, and M. Schiffers. Quality of context information: What it is and why we need it. In *10th Int. Workshop of the HP OpenView University Association (HPOVUA)*, ACM, Geneva, Switzerland, 2003.
4. D. Conan, R. Rouvoy, and L. Seinturier. Scalable Processing of Context Information with COSMOS. In *Proc. 6th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems*, volume 4531 of *LNCS*, pages 210–224, Cyprus, June 2007. Springer-Verlag.
5. J. Coutaz and G. Rey. Foundations for a Theory of Contextors. In *4th International Conference on Computer-Aided Design of User Interfaces*, pages 13–34, Valenciennes (France), May 2002. Kluwer.
6. W. Dargie. The Role of Probabilistic Schemes in Multisensor Context-Awareness. In *5th IEEE Int. Conf. on Pervasive Computing and Communications. PerCom'07*. IEEE Computer Society, Mar. 2007.
7. A. Dey, D. Salber, and G. Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Special issue on context-aware computing in the Human-Computer Interaction Journal*, 16(2–4):97–166, 2001.
8. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
9. P. D. Haghighi, S. Krishnaswamy, A. Zaslavsky, and M. M. Gaber. Reasoning about context in uncertain pervasive computing environments. In *3rd IEEE European Conf. on Smart Sensing and Context (EuroSSC)*, volume 5279 of *Lecture Notes in Computer Science (LNCS)*, pages 112–125. Springer-Verlag, Oct. 2008.
10. M. Krause and I. Hochstatter. Challenges in Modelling and Using Quality of Context (QoC). In T. M. et al., editor, *Mobility Aware Technologies and Applications (MATA)*, volume 3744 of *LNCS*, pages 324–333. Springer-Verlag, 2005.
11. M. Leclercq, A. E. Ozcan, V. Quema, and J.-B. Stefani. Supporting heterogeneous architecture descriptions in an extensible toolset. In *ICSE '07: Proceedings of the 29th international conference on Software Engineering*, pages 209–219, Washington, DC, USA, 2007. IEEE Computer Society.
12. A. Manzoor, H. Truong, and S. Dustdar. On the Evaluation of Quality of Context. In *IEEE EuroSCC, 3d European Conference on Smart Sensing and Context*, volume LNCS 5279, Zürich, Switzerland, Oct. 2008. Springer.
13. R. Rouvoy, D. Conan, and L. Seinturier. Software Architecture Patterns for a Context Processing Middleware Framework. *IEEE Distributed Systems Online*, 9(6), June 2008.
14. A. Senart, R. Cunningham, M. Bouroche, N. O'Connor, V. Reynolds, and V. Cahill. MoCoA: Customisable Middleware for Context-Aware Mobile Applications. In *Proc. 8th International Symposium on Distributed Objects and Applications*, volume 4275 of *LNCS*, pages 1722–1738, Montpellier (France), Nov. 2006. Springer-Verlag.
15. J. Ye, S. McKeever, L. Coyle, S. Neely, and S. Dobson. Resolving uncertainty in context integration and abstraction. In *ICPS08: 5th Int. Conf. on Pervasive Services*, pages 131–140, New York, NY, USA, 2008. ACM.