Science of Computer Programming 78 (2013) 1912–1929

Contents lists available at SciVerse ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

Building ubiquitous QoC-aware applications through model-driven software engineering

Sophie Chabridon *, Denis Conan, Zied Abid, Chantal Taconet

Institut TELECOM, TELECOM SudParis, CNRS UMR Samovar 9 rue Charles Fourier, 91011 Évry cedex, France

ARTICLE INFO

Article history:
Received 23 January 2012
Received in revised form 30 April 2012
Accepted 24 July 2012
Available online 16 August 2012

Keywords:
Model-driven software engineering
Context
Quality of context
Domain specific language
Ubiquitous computing
Pervasive computing

ABSTRACT

As every-day mobile devices can easily be equipped with multiple sensing capabilities, ubiquitous applications are expected to exploit the richness of the context information that can be collected by these devices in order to provide the service that is the most appropriate to the situation of the user. However, the design and implementation of such context-aware ubiquitous appplications remain challenging as there exist very few models and tools to guide application designers and developers in mastering the complexity of context information. This becomes even more crucial as context is by nature imperfect. One way to address this issue is to associate to context information meta-data representing its quality. We propose a generic and extensible design process for context-aware applications taking into account the quality of context (QoC). We demonstrate its use on a prototype application for sending flash sale offers to mobile users. We present extensive performance results in terms of memory and processing time of both elementary context management operations and the whole context policy implementing the Flash sale application. The cost of adding QoC management is also measured and appears to be limited to a few milliseconds. We show that a context policy with 120 QoC-aware nodes can be processed in less than 100 ms on a mobile phone. Moreover, a policy of almost 3000 nodes can be instantiated before exhausting the resources of the phone. This enables very rich application scenarios enhancing the user experience and will favor the development of new ubiquitous applications.

1. Introduction

In recent years, we have been witnessing a very fast evolution of mobile computing and ubiquitous services. Universal access to information is now an implicit requirement of distributed applications running on mobile devices as users expect data to be brought to them anywhere at anytime. As every-day mobile devices can easily be equipped with multiple sensing capabilities, ubiquitous applications are expected to exploit the richness of the context information that can be collected by these devices and provide the users with the proper service without explicit interaction. However, the design and implementation of such context-aware ubiquitous applications remain challenging as there exist very few models and tools to guide application designers and developers in mastering the complexity of context information.

Context information was identified several years ago as a corner stone of mobile, ubiquitous or pervasive applications [11,14]. Context managers have been proposed to infer high-level context data from low-level raw data extracted from several distributed sources such as operating systems, user profiles, knowledge bases and environment sensors [2,11,15,32]. However, context information is by nature imperfect [4,18] but only a few context managers [3,27,29,34,42] explicitly pay

* Corresponding author.
E-mail addresses: Sophie.Chabridon@telecom-sudparis.eu (S. Chabridon), Denis.Conan@telecom-sudparis.eu (D. Conan), Zied.Abid@telecom-sudparis.eu (Z. Abid), Chantal.Taconet@telecom-sudparis.eu (C. Taconet).

0167-6423/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2012.07.019

attention to the quality of context information (QoC). QoC was first identified by [6] as a fundamental concept for the development of context-aware services that are correctly adapted to the situation of the user. Ref. [6] defines QoC as "any information describing the quality of information that is used as context", representing it using a set of parameters such as accuracy, probability of correctness, trustworthiness, resolution or freshness, and considering that it is intrinsic to the information as opposed to the computing process (e.g., quality of service) or to the hardware equipment (e.g., quality of device). The notion of worth has then been added to introduce the point of view of the targeted applications [25]. Regarding the uncertainty of context information, [18] has identified four types of imperfection of context information. A context attribute may be *unknown* when there is no information about it, leading to incomplete context information. It may also be *ambiguous* as there is a risk of having contradictory information from different context sources. An attribute is *imprecise* when the reported information is correct but not provided with a sufficient degree or precision. As context data are by nature dynamic and very heterogeneous, they also tend to be *erroneous* and not exactly reflect the real state of the modeled entity. Therefore, taking into account the knowledge of the quality of context information through meta-data helps in mastering the uncertainty of this context information and appears to be essential to reach an effective and efficient context management and furthermore to provide context-awareness [47].

Regarding the context management service and the design of context-aware applications, our approach follows an imperative process rather than an ontological reasoning. We are indeed particularly concerned with performance, as one of our objectives is to handle context management on mobile devices. In [5], we have proposed an hybrid architecture for context management integrating both a process-oriented context manager (PCM) and an ontology-based context manager (OCM). We have shown the complementarity of these two approaches with the recommendation to prefer a PCM on a mobile device while benefiting from an OCM on a remote server. This enables one to reach good performance, to deal with the resource constraints of current mobile devices and to avoid sharing sensitive personal data while allowing reasoning on high-level knowledge described using ontologies when required. In this article, we further develop the design of the PCM part of the context management service and the design of the context-awareness of ubiquitous applications on mobile devices by proposing a process with its associated tool chain for handling QoC. This process extends the approach introduced in [7] and is built on our previous work [10,44]. The tool chain includes both a generic and extensible design process for context-aware applications taking into account QoC and a generic context manager which handles QoC. In the design process, we consider context management, context-awareness and business applications as separate concerns. We follow a model driven approach allowing us to generate the appropriate context management artefacts which include the computation of the associated QoC. For these generated artefacts, we have performed extensive performance measures on mobile phones both in terms of memory and processing time and determined the overhead associated to QoC management. In addition, we illustrate QoC modeling and management through an implemented prototype of a Flash sale application for mobile users. Using this demonstrator, we show how the addition of a context-awareness aspect in the application design process leverages the overall quality of mobile and ubiquitous applications.

The organization of the article is the following. We present in Section 2 the motivations of our work through a location-aware flash sale scenario. Then, in Section 3, we draw a global picture of the proposed design process which includes QoC requirement elicitation and we relate it to the different frameworks that we provide. In Section 4, we explain in detail the role of QoC in this design process, and we present the resulting context-awareness model for the illustrative scenario. Next, in Section 5, we describe the implementation of our QoC-aware context manager. In Section 6, we present an evaluation of our proposition, especially in terms of performance. Firstly, we compare the performance results of our context management framework with and without QoC and secondly we present the performance results for the Flash sale application on a mobile phone. In Section 7, we discuss related work concerning both context-awareness design and QoC management before concluding the article in Section 8.

2. Motivations and objectives

In Section 2.1, we introduce the flash sale offer scenario, a motivating scenario which illustrates the necessity to include QoC in the design and management of the context-awareness of ubiquitous applications. From this scenario, we highlight in Section 2.2 the objectives which drive our proposition.

2.1. Flash sale motivating scenario

At 10.00AM, *Celina drives to the largest mall in the region for some shopping. She has her new mobile phone with GPS navigation*, 3G, *WiFi and Bluetooth communication. When she arrives on the outdoor parking lot of the mall, she receives a short message informing her of the availability of the new* Flash sale offer *service and inviting her to download this new application. As Celina is a frequent client of this mall, she has already registered with the mall office and she has indicated her shopping interests and favorite products. Right after downloading the application, she receives a notification indicating that she still has one hour to benefit from a flash sale running in her favorite sportswear shop. As the accuracy of Celina's location is not very good, the positions of Celina and the shop are shown on the map, but no indication of the path to follow is provided. The remaining time before the end of the flash is displayed on the phone screen. Later in the afternoon, Celina is inside the mall at the grocery store. She gets another alert for a flash sale offer proposed by a new shop that has just opened and that she does not know yet. The flash sale is running for only* 15 min. *The radio coverage being currently very good, the location of Celina inside the mall can be determined*

with a very high quality level. She then receives on her phone a detailed map of the mall focused on her position and indicating the precise path to the shop and the distance still to go. All along the way, she is informed of the remaining time until the end of the flash sale and of the distance to reach the shop; the map on her phone also gets refreshed automatically. Thanks to the guiding, she reaches the shop before the end of the flash sale.

Through this scenario, we show that it is essential to provide applications with information on the quality of the context they depend on. For example, the location of a mobile user may be provided by several sensors (e.g. GPS, GPRS, Wifi), each one with a different accuracy property. QoC meta-data are taken into account at different levels of the context-awareness process. First of all, the locations shown on the user map are the locations with the best QoC (selected among the available locations). Furthermore, different services (or different behaviors of the services) are delivered to the user and are chosen in relation to the QoC. A flash sale recommendation is only triggered if the system can deduce with a sufficient confidence that this event will interest the user—that is, it matches the product preferences of the user and the current location of the user allows them to reach the place where the flash sale is taking place before it ends. Moreover, the precise guiding through the mall is only offered if the location is known with a good QoC.

In the remainder of this article, we present a generic design process and a generic framework to handle the QoC. In the different sections of this article, we illustrate the approach through the prototype of the Flash sale application we have designed and implemented with our frameworks for this scenario.

2.2. Objectives

Ubiquitous applications do need to be aware of their environment. A new situation may become a signal to start a new service (e.g., show an itinerary on a map) or to make a recommendation (e.g., flash sale alert). In addition, since the observation of the environment is imperfect, application designers do need means to express the minimum QoC requirements for a context-aware action to be triggered.

Context managers, which can be defined as software entities computing high-level context data from various sources, should have the capability to evaluate the QoC associated to each piece of context data, whatever the level of abstraction of these data. The context manager should be able to evaluate the QoC level from various QoC parameters (e.g., accuracy and freshness). Furthermore, the list of QoC parameters should be extensible and the QoC computation should also be optional.

Lastly, ubiquitous applications are undoubtedly mass-market applications. Context-awareness frameworks should then be designed with both scaling and privacy preservation requirements in mind. We argue that having context managers running on user mobile devices is appropriate since it offers better response times than a central computation and avoids sharing sensitive personal data with a central context manager.

3. Integration of a QoC concern in the design process of context-aware applications

In Section 3.1, we briefly present the design process we follow to build context-aware applications. It is important to notice that our objective in this article is not to introduce the design process in all its details, but to highlight where QoC requirements are elicited and taken into account. Next, in Section 3.2, we present the frameworks supporting the design process.

3.1. Design process

The role of context management is to provide high-level context data to identify situations which may produce reactions in context-aware applications. It consists in context data acquisition, context data processing (fusion, aggregation, interpretation), and context data presentation to context-aware applications. Our design process decouples as much as possible context management design from business application design. For this purpose, we augment the classical business application domain with two dedicated domains: context management and context-awareness, leading to two new roles, namely context management designer and context-awareness designer. For instance, a context management designer defines how to compute the distance between two entities whatever these entities are. Then, the context-awareness designer of the Flash sale application expresses the application's needs to obtain the distance between customers and shops with an accuracy of five meters. In the rest of this section, we present the activities and work products of each engineering domain. Of course, even if the two roles and their activities are presented separately, the context management designer and the context-awareness designer work collaboratively to add the context-awareness to business applications. For instance, the context-awareness designer can request missing work products from the context management designer.

Fig. 1 displays the activity diagram representing the four main activities of the context management designer. The figure uses the OMG SPEM notation (Software Process Engineering MetaModel) [30]. It summarizes the roles, the activities and the work products of the context management domain. The meta-model of the context management framework, namely COSMOS [10] in our work, is an input to all these activities. As a consequence, the models and the implementation artefacts produced conform to the COSMOS meta-model and API. The role of the context management designer is to define the inference of high-level context information such as the distance between two locations from raw context data collected by what are named *context collectors* in our terminology. For this purpose, the designer defines what we call *context nodes* using a domain specific language, namely the COSMOS DSL, in which the inference of high-level context information is

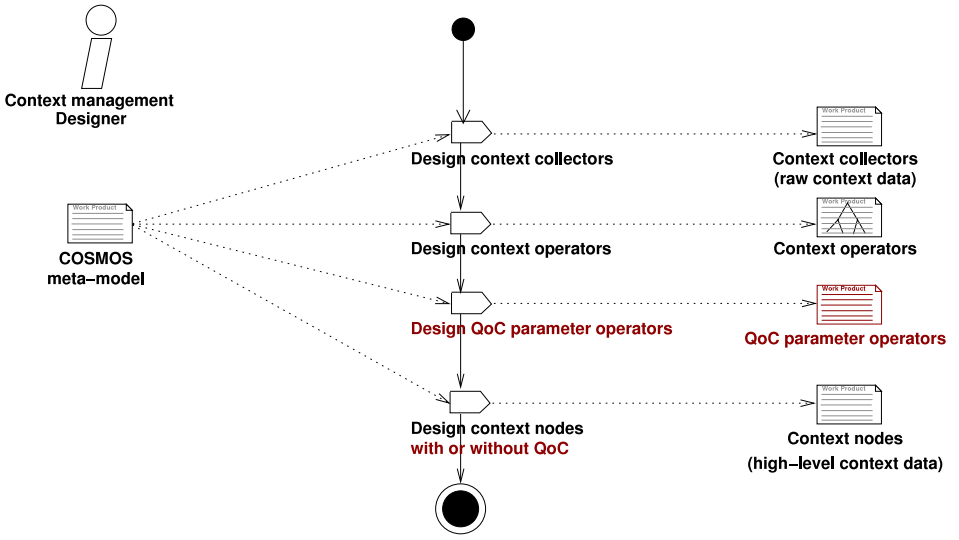

Fig. 1. Context management designer activity diagram.

an expression of low-level context data, the operators of this expression being called *context operators*. Therefore, the first activity consists in specifying the context collectors using the COSMOS DSL and implementing them into a few lines of Java code. For example, the designer may implement several context collectors to get raw location values from different sensors (e.g., GPS, GPRS, Wifi) and different devices (e.g., Android phones, iPhones, Windows Mobile phones). The second activity concerns the design of the context operators corresponding to the context processing. In this activity, the context management designer uses also the COSMOS DSL and Java. The novelty of our approach is that context operators are augmented with QoC computations, thus leading to what we call *QoC-aware context operators* and *QoC-aware context nodes*. In order to bring out the importance of qualifying context data, we introduce an activity dedicated to the design of the *QoC parameter operators* that process the QoC meta-data (e.g., the freshness of the context information, which is a function of its age since it was collected and its lifetime). These tasks constitute the third activity. The fourth and last activity consists in designing the entities that model the high-level context data by writing in the COSMOS DSL the expressions of the context processings. These expressions reuse the context collectors, the context operators, and the QoC parameter operators designed in the previous activities. This is where the COSMOS DSL is the most relevant since it enables reuse by composition and it relieves the context management designer from programming the expressions (cf. Section 5.1, and more particularly Fig. 8). For instance, a QoC-aware context operator is the composition of QoC parameter operators with a regular context operator (e.g., a global QoC level is computed by aggregating several QoC parameters like the freshness and the trustworthiness) and a QoC-aware context node is the composition of a QoC-aware operator with child context nodes (e.g., the outputs provided by several location collectors are compared and the one with the best quality is chosen). Beyong this set of context nodes, some of them are presented to the context-awareness part of the application. In our terminology, these context nodes are called *context policies* and constitute the context data presentation to context-aware applications. Finally, these context collectors, context operators, QoC parameter operators, QoC-aware context operators, context nodes, and QoC-aware context nodes are gathered into libraries and may be reused by other ubiquitous applications (e.g., location and distance nodes may be useful for many applications). However, each application may have specific QoC requirements defined by context-awareness designers. Some of these artefacts, more specifically the context nodes providing the highest-level context data for identifying situations, may also be dedicated to a given application (e.g., the flash sale node of our example application is indeed specific to the Flash sale application).

Fig. 2 displays the activity diagram representing the main activities of the context-awareness designer. The context-awareness designer defines the context-awareness of a context-aware application. The first activity shown in the diagram consists in the specification of the entities to be observed at runtime (called the *observable entities* or *entities* in our terminology), and the specification of which context data to collect or infer (called *observables* in our terminology) on these entities. For instance, in the case of the Flash sale application, customer and shops are examples of entities to observe, and location and preferred products are examples of observables. The CA3M context-awareness meta-model [44] and the business logic meta-model are inputs since the role of the designer in this activity is to tag with dedicated stereotypes the UML model elements representing the entities and the observables. Once the context-awareness designer has identified the entities and the observables in the first activity, the second activity consists in linking context management work products—that is, the context nodes or policies corresponding to the selected observables—to business application work products through the definition of specific contracts. For instance, a context-awareness contract specifies that the Flash sale service has to be triggered when an interesting flash sale is detected. When defining the context-awareness contracts, the context-awareness designer introduces the QoC requirements of the business application and configures the QoC-aware context

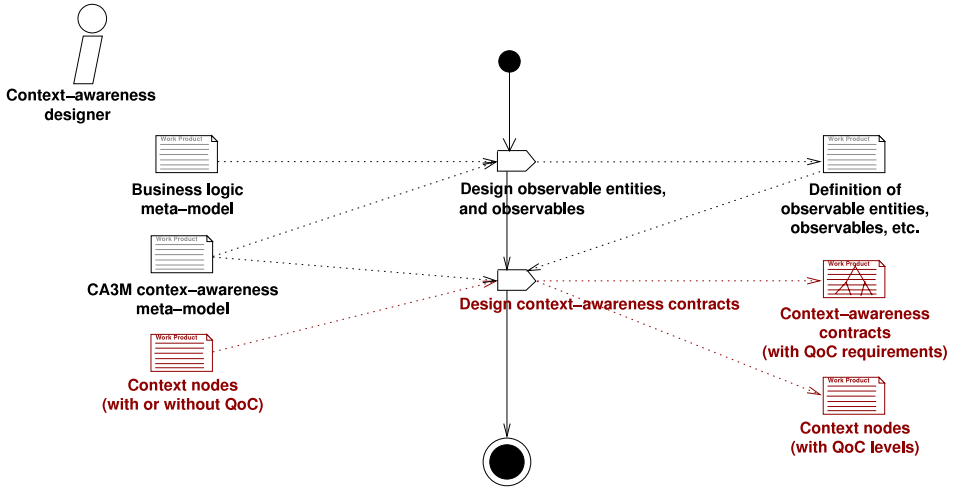

Fig. 2. Context-awareness designer activity diagram.

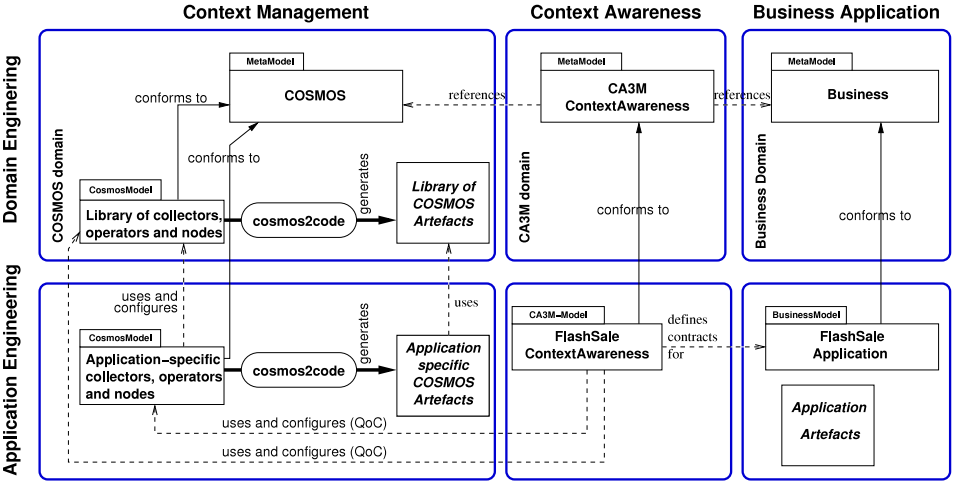

Fig. 3. Design process and associated frameworks.

nodes with the right QoC levels (cf. Section 4.1 for a detailed presentation of the QoC level artefact). Again, since the QoC requirements are different for each context-aware application, the configuration of the QoC levels is defined in the context-awareness domain.

3.2. Frameworks associated to the design process

Fig. 3 presents a global view of the frameworks associated with the model-driven design process. Regarding the context management, our proposition relies on the COSMOS framework [10]. This framework is based on the concepts of context collector, QoC-aware context operator, QoC-aware context node and context policy. High-level context data are computed into context nodes. Context nodes are written with the COSMOS Domain Specific Language (COSMOS DSL) derived from the COSMOS meta-model. From the context nodes, the cosmos2code tool generates most of the COSMOS nodes' artefacts: component-oriented architecture definitions and classes. Each context node contains one context operator which contains the instructions to compute both higher-level context data from lower-level ones and their associated QoC meta-data.

Concerning the context-awareness part of the business application, we make use of CA3M (Context-Aware Middleware based on a context-awareness Meta-Model) [44]. CA3M concepts allow context-awareness designers to link application entities with context data sources (the context policies) and to define specific contracts which link the application with the context management framework. In this article, we focus on showing how we complement those contracts with QoC requirements. The QoC requirements coming from the context-awareness contracts are taken as input to build configured context nodes and policies. These QoC requirements define both the QoC parameter operators which have to be included into the hierarchy of context nodes and the QoC levels to be applied to filter context data before triggering an action in the application. In the remainder of the paper, for the sake of simplicity, we do not make the distinction between the

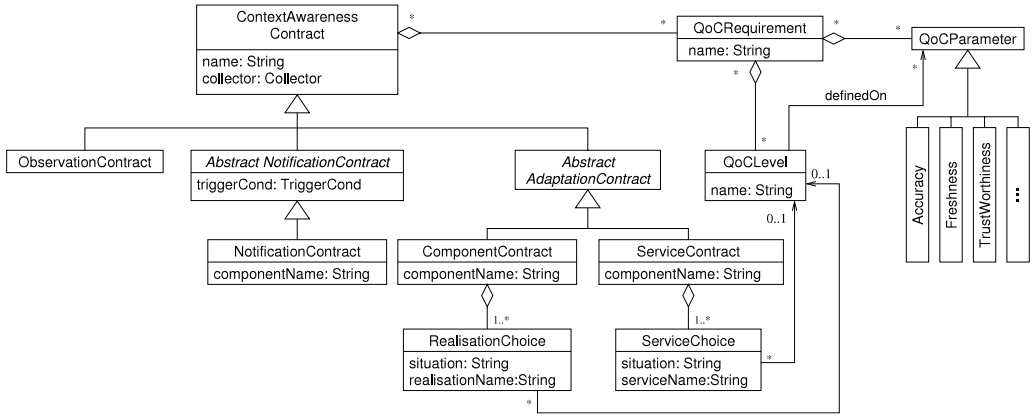

Fig. 4. The CA3M meta-model of the context-awareness contract augmented with QoC.

middleware service corresponding to the context-awareness management framework (CA3M) and the middleware service corresponding to the context management framework (COSMOS): both are collectively named the context management service.

4. Model-driven approach for context-awareness with QoC

We follow a model-driven approach to define the context-awareness of the application. As stipulated by the model-driven approach, designers write models conforming to meta-models. Since we target ubiquitous applications, besides classical application models, we need additional domain specific context-awareness models [43]. In this article, we focus on the treatment of the QoC in the context-awareness management process. So, in this section, we start by describing the concepts manipulated in the context-awareness meta-model with a specific focus on QoC. We then describe the Flash sale application which implements the scenario presented in Section 2. Finally, we show how we use the context-awareness meta-model to derive the context-awareness model of the Flash sale application.

4.1. Context-awareness meta-model with QoC

The context-awareness meta-model is central to the design process of ubiquitous applications. This meta-model is an extension of the one presented in [43]. Fig. 4 displays the subset of this meta-model which presents context-awareness contracts. The contribution of this article lies in the context-awareness contracts, and, more especially, in the QoC aspects of these contracts.

A context-aware system is composed of observable entities, observables, and context-awareness contracts defined for these observables. The *observable entities* (not displayed in the view presented in Fig. 4) are logical or physical elements to be observed. The context data types of the information observed on these entities are called *observables*. A *context-awareness contract* defines a contract for a given observable and a given application to be fulfilled by the context management service.

In the context-awareness contracts, the context-awareness designer expresses *QoCRequirement*s indicating what level of QoC (*QoCLevel*) associated to an observable value is necessary for an action to take place. A *QoCLevel* defines the expected value of some QoC parameters. The QoC parameters correspond to the meta-data we associate to context data. A first set of these parameters is directly collected from context sources, depending on the information available at the sources, and additional parameters can be computed at the acquisition step or even later during the inference process by the context management service [1]. As an example, Fig. 5 presents the three QoC levels named *high*, *medium* and *low* defined for the Flash sale application and concerning the user location observable. However, a *QoCLevel* is not restricted to being used with a specific observable. For a different observable, like a temperature, the same QoC levels can be used or new ones can be defined, depending on the application needs. We have chosen to identify QoC levels using a name expressing the global quality expected, but other identification means could be used.

We introduce three subclasses of context-awareness contracts that express three ways of modeling context-awareness requirements. Firstly, when an application designer wants to synchronously observe context data, it specifies an *ObservationContract*. A context management service, such as CA3M [44], uses these contracts to instantiate observation artefacts, such as COSMOS context nodes, able to provide the context information with the right level of QoC. Secondly, by using a *NotificationContract*, the application designer can specify subscriptions to events that occur for instance when a numerical observable value reaches a fixed threshold or when an enumerated observable value changes from one enumeration to another one with a given QoC. The condition is defined via the *triggerCondition* attribute. The designer is also asked to indicate the application entity that will receive the notifications. Note that observation and notification contracts managed by the context management service do not deal with the adaptation decisions which are the responsibility of the application. Then, the third kind of context-awareness contract, namely the *AdaptationContract*, puts in action the adaptation

```
quality high (freshness : Freshness,
              accuracy : Accuracy,
              trustworthiness : Trustworthiness) {
   freshness >= 0.8;
   accuracy < 10;
   trustworthiness >= 0,9;
}

quality medium (freshness : Freshness,
                accuracy : Accuracy,
                trustworthiness : Trustworthiness) {
   freshness >= 0.8;
   accuracy < 10;
   trustworthiness < 0,9 ;
}

quality low (freshness : Freshness,
             accuracy : Accuracy,
             trustworthiness : Trustworthiness) {
   freshness < 0.8;
   accuracy < 10;
   trustworthiness < 0,9;
}
```

Fig. 5. QoC levels defined for the Flash sale application.

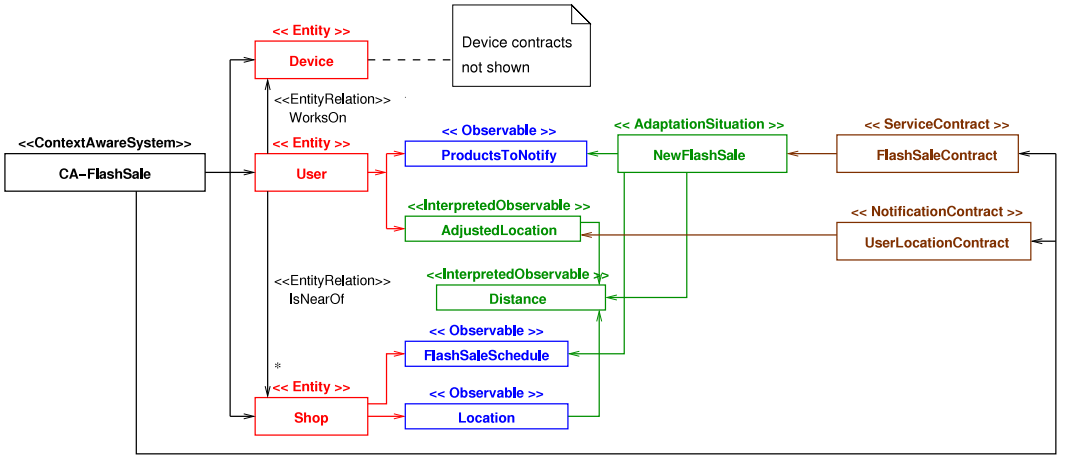

Fig. 6. Flash sale application context-awareness model excerpt.

decisions taken by the application following the detection of a specific adaptation situation. A *ServiceContract* is an example of an *AdaptationContract* which triggers the activation of a service. This kind of contract may require support from the middleware layer to find, instantiate and deploy the required service.

4.2. Modeling the context-awareness of the Flash sale application

We now discuss the design of the Flash sale application from Section 2.1. One characteristic of this application is to exploit the knowledge of the QoC. The application sends attractive messages to the mobile clients present at the mall to inform them about on-going special commercial offers that might interest them. This potential interest is evaluated by the system according to the context information available and its associated QoC. For this application, we have chosen three QoC parameters to characterize the quality of the location information as discussed in [8]: the accuracy (expressed in m), indicating how close the estimated value is to the real one, the freshness or up-to-dateness (between 0 and 1), representing how old the context information is relative to its lifetime, and the trustworthiness (between 0 and 1), which depends on how context data are collected or computed and on how much the context source can be trusted (see [1] for computation details).

We manipulate context data of different types. Some of them, such as the shop location, the user's profile, and the characteristics of their mobile device, can be considered as static since their values change only occasionally. In contrast, some context data are dynamic and change very often. In this article, the application example we consider involves two kinds of dynamic context data, the location and the movement speed of the user. However, the engineering approach we propose can also be applied to other types of context. We have considered in previous works weather and temperature information [43], waiting time in a queue, printer availability and classroom availability [5]. The context management process may be applied to any kind of dynamic context data.

Since the application is expected to be used outdoors (in a parking lot or on the street) or indoors (in a mall), we consider multiple methods of localization but with the constraint that they do not require the deployment of a specific infrastructure. As we want to favour low-cost solutions, the quality of the location information becomes a key issue. It is important to master the QoC of the location in order to choose the best available location information. Depending on the capabilities of the mobile devices, several localization means can possibly be available. In our demonstrator, we have considered three different technologies: GPS, 3G Telephone Cellular network (GSM, GPRS…) and WLAN radio communication.

The application scenario results in a context-awareness model conforming to the context-awareness meta-model. We show in Fig. 6 an excerpt of this model focusing on the flash sale. The context manager takes advantage of the presence of

the QoC attached to context information in order to provide the best location *BestLocation*—that is, the location with the best level of QoC among the available ones at a given time. We define two contracts related to the flash sale. The first one is the *UserLocationContract* notification contract. Thanks to this contract, the application component responsible for displaying the map of the mall receives the user location updates as soon as the user moves significantly. The second contract is the *FlashSaleContract* service contract. It allows a Flash sale service to be triggered when there is a flash sale of interest in the user's vincinity. The Flash sale service is chosen according to the value of the QoC.

As presented in Section 4.1, Fig. 5 gives the definitions of three *QoCLevel*s. The QoC level named *high* corresponds to the case where the three QoC parameters that we consider in the scenario have sufficient values. The freshness is only 20% below the maximum, the accuracy indicates that the estimated position is less than 10 m away from the real position, and the trustworthiness is greater than 90%. With the QoC level named *medium*, only the trustworthiness is below what is expected in the previous case. Finally, the QoC level named *low* is even lower. More levels could be defined, but by experience these three levels are realistic and represent sufficiently discriminated cases. The Flash sale services associated to the different QoC levels differ in the added value they provides to the user. When the QoC level is *high*, the application has a sufficient confidence in the estimation of the location of the user to guide them precisely. It displays an alert message with the distance and the remaining time to reach the shop where the flash sale is taking place and also draws the route to follow on the map. When the QoC level is *medium*, the route is not displayed on the map. Finally, with a *low* QoC level, the distance to walk is not displayed for the user and only the remaining time is indicated.

5. Managing the context and its associated quality

We present in this section the architecture of the Flash sale application we have developed using the COSMOS framework [10]. In Section 5.1, we first start by describing the main concepts that guided the design of COSMOS. We then present how it takes QoC into account.

The component-based orientation of the approach demonstrates the interest of the model-driven engineering (MDE) approach: as the designers of the context management framework, we provide application designers with libraries of components for instance to process the context data and the QoC meta-data; the context framework is extensible so that other designers can develop and provide off-the-shelf components that get integrated into the context framework for instance to take into account new QoC parameters; application designers do not program context processing of the raw context data and QoC meta-data they collect but rather compose the architecture of the context manager that fits their needs. Section 5.2 presents such a design: the application designer models the inference treatments both of the context data and of the QoC meta-data; these treatments are organised into a graph; the context framework being a process-oriented component-based context framework, the nodes of the graph correspond to components and the graph corresponds to an architecture. Finally, Section 5.3 provides some details on the realization of the illustrative scenario on mobile phones.

5.1. Context and QoC management with COSMOS

COSMOS is a process-oriented context manager that collects raw context data from the different context sources and transforms them to higher-level context data. COSMOS can both be responsible for inferring high-level context data and situations, and supply other context managers with low-level context data, for instance to ontological inference engines such as MUSE [5] or agent-based frameworks such as SALSA [37]. COSMOS is implemented in Java as an open source framework.[1] Its usefulness has already been demonstrated in different projects, for instance for processing simulation data in a distributed simulation environment [35] or for aggregating location data provided by various sources [8].

The processing is organized into a graph representing a context policy which is a hierarchy of context nodes. These nodes are implemented as software components and can be shared across several context policies. They perform basic context-related operations (e.g., gathering data from a system or network probe, computing threshold or average values) and are assembled with a set of well-identified architectural design patterns [38]. A library of context operators allows designers to define new COSMOS nodes by composition.

A context node has properties which define its behavior with respect to the context management policy. It can be *passive* (by default) or *active*. An active node is equipped with an activity to execute a given task. Control of the communication into the hierarchy of context nodes may be bottom–up (*notification*) or top–down (*observation*). 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 provided without polling child nodes; for notifications, the received data are 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 allows the level of computing resources used to be finely tuned.

As shown in [1], COSMOS manages QoC thanks to a QoC-aware context node composed of a `QoCAwareOperator` and `QoCParameterOperator` components. The QoC is determined using QoC parameters taking as input QoC data coming from child components. We manipulate QoC separately from context data since we consider QoC as an additional concern of

[1] http://picoforge.int-evry.fr/projects/svn/cosmos/.

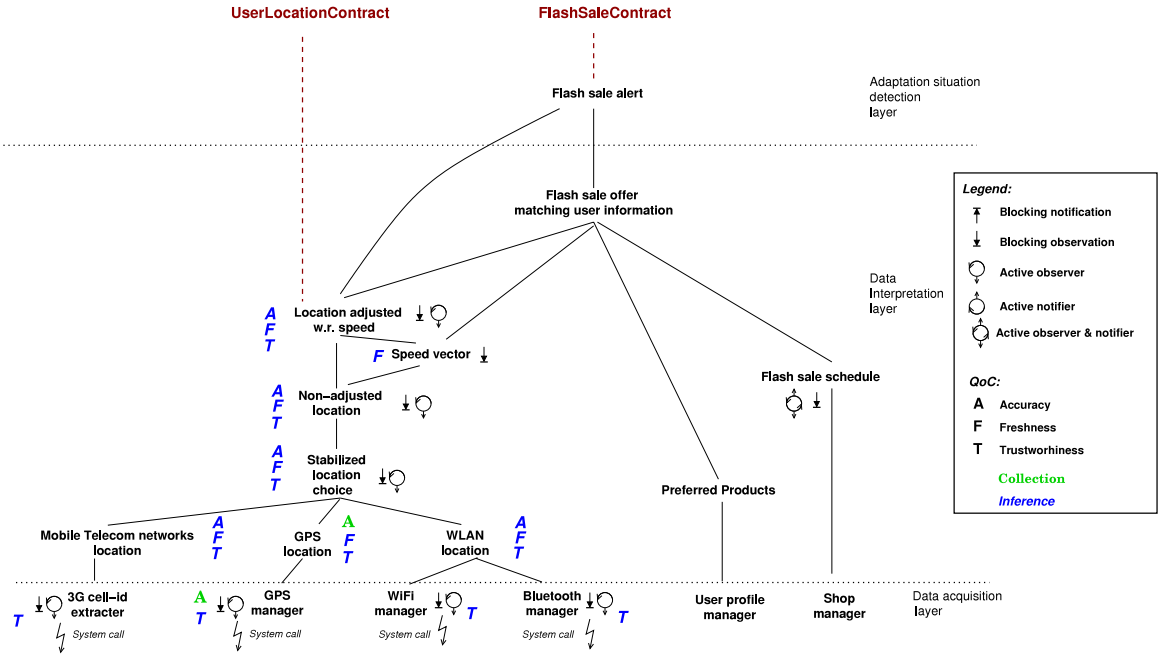

Fig. 7. Flash sale COSMOS context policy with QoC.

```
package evry2mallqoc;
chunk position = {
    classname : evry2mallqoc.PositionChunk,
    typeparam : evry2mallqoc.PositionInfo
}
message location = {
    position, accuracy, freshness, trustworthiness
}
operator computeGprsLocation = {
    output : location,
    classname : evry2mallqoc.ComputeGprsPosition,
    input : gprs
}
node gprsLocation = {
    operator : computeGprsLocation,
    children : gprsExtracter
}
node stabilizedLocation = {
    operator : computeStabilization,
    children : gprsLocation gpsLocation wlanLocation
}
operator computeNonAdjustedLocation = {
    output : location,
    classname : evry2mallqoc.ComputeNonAdjustedLocation,
    input : location
}
node nonAdjustedLocation = {
    operator : computeNonAdjustedLocation,
    children : stabilizedLocation
}
operator computeAdjustedLocation = {
    output : location,
    classname : evry2mallqoc.ComputeAdjustedLocation,
    input : location speed
}
node adjustedLocation = {
    operator : computeAdjustedLocation,
    children : nonAdjustedLocation speedVector
}
configuration : gprsExtracter
    [observethrough=false] [periodobserve=10000]
configuration : stabilizedLocation
    [observethrough=false] [periodobserve=2000]
configuration : nonAdjustedLocation
    [observethrough=false] [periodobserve=3000]
configuration : adjustedLocation
    [observethrough=false] [periodobserve=5000]
```

Fig. 8. Adjusted location policy in COSMOS DSL.

context management. This allows for a flexible QoC management, which is activated only when necessary. This further opens the way for performance optimization such as computing the QoC on a set of context data and not simply for each sensed piece of data. Moreover, each QoCParameterOperator component computes a specific QoC parameter such as accuracy, freshness, etc. As a consequence, for a given application, application designers select the relevant QoC parameters in the library of COSMOS QoCParameterOperator components and compose their QoC context nodes.

5.2. COSMOS context policy of the Flash sale application

We show in Fig. 7 the graph of the *New Flash Sale* COSMOS context policy defined for the Flash sale application. Fig. 8 presents the description written in COSMOS DSL[2] of the `Location adjusted` subtree. The subtree is not completely

[2] http://picoforge.int-evry.fr/-websvn/filedetails.php?repname=cosmos&path=/trunk/cosmos/dsl/dsl/cosmos.dsl/documentation/grammar/grammar.pdf.

described, only the application specific parts are shown. Also, nodes coming from the COSMOS generic library (referenced in italic) are described elsewhere. This specification in COSMOS DSL is taken as input by the cosmos2Code tool to generate the programming code of the context nodes of the application. The component architecture is completely generated and skeletons of primitive components are also generated so that developers only have to write a few lines of Java code to complement them.

The policy takes advantage of the available positioning technologies, such as cellular networks, wireless radio and GPS, and determines a stabilized location choice. This choice is guided by the QoC of the location information. As introduced in Section 4.2, we consider three QoC parameters for the location information, which are the accuracy (denoted A in the graph), the freshness (denoted F) and the trustworthiness (denoted T). We distinguish the QoC parameters that are collected from context collectors of the data acquisition layer from the QoC parameters that are computed by inference following the paths of the context policy. With satellite positioning technologies, an accuracy measure may be provided with the position. This is the case for Assisted-GPS [39]. Therefore, we indicate in Fig. 7 that the accuracy parameter is collected by the GPS manager. For the other positioning technologies, the accuracy is measured via statistics derived from experimental observations. The trustworthiness is computed as depending upon the location source. For instance, with Wi-Fi communication, we derive a trustworthiness measure from the strength of the received radio signals [8]. The freshness parameter is also computed by the COSMOS framework at the time the context data is exploited, that is in the data interpretation layer. During the inference process, the Stabilized location choice context node determines the best location according to the values of the trustworthiness, the freshness and the accuracy parameters, in this order.

As users are mobile, we take into account in the context policy the speed of the movement of the user. The speed vector is deduced from the position history of the user as stored on the phone. This allows the location of the user at a given time to be adjusted and the amount of time they might need to reach a given flash sale location to be determined. A freshness QoC parameter is computed and associated to the speed vector. In the case where the mobile device is equipped with an accelerometer, we can benefit from this additional capability to estimate the movement of the user as done in [45]. The remaining part of the context policy directly derives from the context-awareness model of the Flash sale application (cf. Fig. 6). When a flash sale is scheduled in the short term (information coming from the Flash sale schedule node), the Flash sale offer context node makes use of the location of the user, the location of the shop where the flash sale is taking place and the user's movement speed to determine whether the flash sale offer matches the user's situation. If this is the case, the Flash sale alert adaptation situation is detected and the Flash sale service gets activated. Then, depending on the QoC of the user location, the appropriate service is proposed to the user according to the flash sale contract.

The context nodes of the context policy shown in Fig. 7 are to be deployed entirely on the mobile phone of the user. We have designed the Flash sale application as an autonomous application that can entirely run on the mobile phone. Therefore, the deployment of the components managing the global map of the mall center and the list of the flash sales that are scheduled within the next hours occurs when the user arrives near the mall. The map of the mall center is displayed on the user phone screen with a focus on the current location of the user. There is also a zoom feature allowing the user to change the scale of the map. This design removes any concern the user might have with regard to the preservation of their privacy. The mall center information system does not get access to the location information of the different clients. The users control the knowledge of their own location which is stored only on their mobile phone.

5.3. Implementing the Flash sale application on a mobile phone

We present in this section the prototype we implemented to validate the Flash sale application. We use the Siafu open source context simulator[3] to generate context information. By using a context simulator, we can better experiment and measure the adequacy of our prototype with more complex scenarios. We use it also for integration tests before the validation tests with end-users. We have prepared a map of the Évry 2 mall center and defined 17 product types for which flash sales can be proposed. Network overlays can also be defined and several access points can be positioned in the mall. Moreover, the context simulator allows the behavior of agents to be simulated very precisely. Some anonymous agents have random behaviors and other named agents, such as Celina, have well-defined behaviors. A full map of the mall center is deployed on the mobile phone. This map is then displayed on the screen with a sufficient zoom level to be readable, centered on the user's location.

We have developed the mobile application in Java and tested it on Android phones. Fig. 9 shows a screen copy of the wireless toolkit phone emulator. As this is a prototype built for demonstration purposes, we display the different locations available on the phone which correspond actually to internal information. A real application would only show the chosen estimated location to the user. During the simulation, an alert message gets displayed on the phone screen when a flash sale notification is received. Depending on the value of the QoC, additional information is given in order to guide the user towards the flash sale location (see Section 4.2).

We show in Fig. 10 a screen copy of the case where a flash sale offer occurs and the location information is of a *high* quality. This is the optimal case where full information on the flash sale is given to the user with the distance to the shop, the remaining time to get there and also the route to follow displayed with dotted lines.

[3] http://siafusimulator.sourceforge.net/.

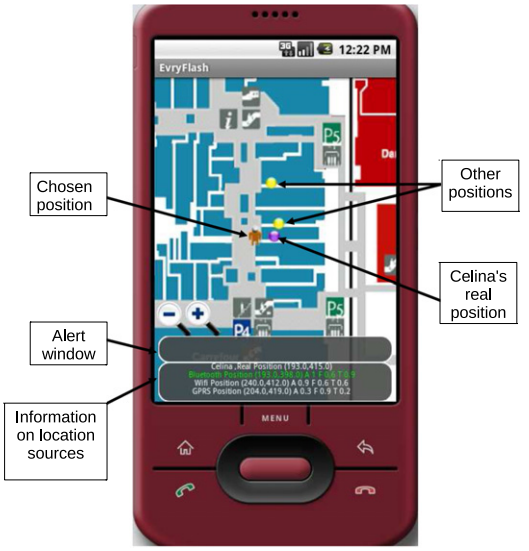

Fig. 9. The multiple positions of Celina.

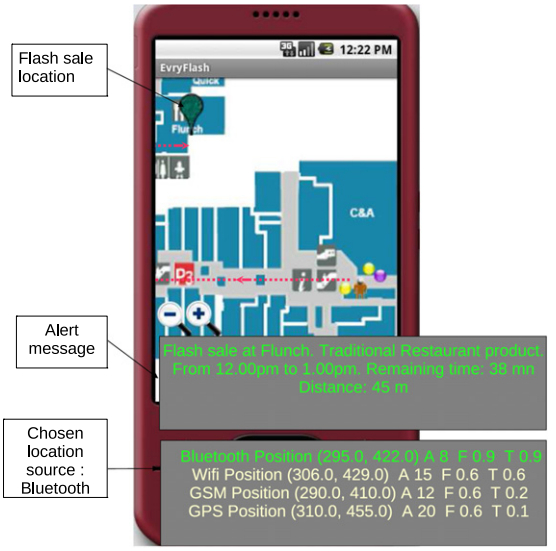

Fig. 10. Flash sale offer notification.

6. Evaluation

We present in this section the experiments we have conducted to evaluate this proposition. We give in Section 6.1 the first experience feedback coming from our partners. Next, we discuss in Section 6.2 the obtained performance evaluation results.

6.1. Qualitative evaluation

We start by discussing the quality of experience of end users as evaluated by our industrial partners. We then report how the proposed design process together with a complete tool chain has been welcomed by ubiquitous application designers and developers. We finally broaden the applicability of our proposal by considering two other applications concerned with other types of context data and not only with location information.

6.1.1. CAPPUCINO demonstrator

The Flash sale application has been experimented on as one of the demonstrator applications of the CAPPUCINO project[4] (funded by the French Unique Inter-ministerial Fund) in which two large French chain stores were involved. Demonstrations

4 http://www.cappucino.fr.

were made to our industrial partners who welcomed the new shopping experience we bring to the users. As malls are becoming very large, or as outside shopping centers can be extended over a rather wide area, any information on shop and client location is expected to be relevant and correct. Knowing the quality of the location information brings an added-value to the Flash sale application in terms of the quality of the user experience.

In this project, the industrial partners are very concerned with privacy issues. They appreciate that the COSMOS manager can execute entirely on mobile phones. Thanks to this characteristic, there is no need to transfer any personal data outside of the mobile device.

By adding quality meta-data to context information, we improve the quality of the experience received by end users. For instance, in the case of a flash sale offer of limited duration, it is important to know how fresh the location information concerning the users is in order to provide them at the right time with offers that are relevant to them. Moreover, it would be counterproductive to send messages concerning special offers to clients after they have already left the mall. This feature is made possible through QoC management. However, as the project did not include any tests in a real shopping mall, further evaluation in real life conditions is still required.

6.1.2. Benefits of a model-driven design process and its associated tool chain

From the point of view of application designers, designing the context-awareness concern of a ubiquitous application is a complex task. For this purpose, a model-driven design process breaking down the different steps to be followed is very helpful. Domain specific models simplify the task of providing designers with specialized editors which guide them during the whole design process. This task will become even easier as new COSMOS context nodes, operators, QoC-aware operators, QoC requirements and QoC levels are made available through libraries. With these libraries of artefacts and thanks to the cosmos2code tool that helps to generate a large part of the context-management code, applications can become context-aware with a minimum of code writing.

Concerning the QoC meta-data, they are involved both when collecting raw context data from sensors and also when inferring higher-level context data. The QoC meta-data and the associated treatments can be added at the design time in the application model. They can also be added at runtime, as the meta-models and the models are made available at runtime thanks to the tool chain we provide. Indeed, the list of QoC treatments and parameters can dynamically vary since the component-oriented architecture of the context manager reifies context management concepts as components. For instance, adding a QoC treatment to a node is performed by encapsulating a regular context operator with a QoCAwareOperator, and adding a QoC parameter is performed by adding a QoCParameterOperator component.

At the early stages of this research project, designers from other research teams and from the industry were concerned with the complexity of an MDE approach that favors the use of meta-modeling and component-based software engineering for designing ubiquitous applications targeting small devices such as smartphones. A key issue of our work is therefore to relate the MDE approach to software engineering concepts, disciplines and technologies in usage in the communities of programmers of mobile phones and small devices. In this article, we claim that, thanks to an MDE approach, designers can assert a given level of quality of their ubiquitous applications developed, deployed and used. A better separation of concerns is achieved through the distinction of three distinct roles (business logic designer, context management designer, and context-awareness designer) with specific tools for each of them. The context-awareness variation points are explicitly identified in business logic models, and the context-awareness models conform to the CA3M meta-model. The context management artefacts conform to the COSMOS meta-model; context processing is expressed as expressions using COSMOS DSL; and the cosmos2code tool generates all the technical parts so that only the functional parts of context management must be written in a few lines of code.

An important point for the adoption of our solution is the massive use of up-to-date tools from the open source community. The ecosystem of our solution benefits from the support of the open source community. Apache Maven[5] is used for software project management of all the activities of the design process. The CA3M and COSMOS meta-models are built using the Eclipse Modeling Framework (EMF).[6] A dedicated editor plug-in is then created for modeling with CA3M. The COSMOS meta-model is an input to the COSMOS DSL. The DSL editor is a dedicated Eclipse plug-in specified in Eclipse Xtext[7] as an ANT-LR grammar annotated with COSMOS meta-model elements. We have written several Maven plug-ins to generate COSMOS artefacts (skeletons of primitive components and definition of the software architecture of composite components), and to transform Java 1.5 code into code compatible with the J2ME CLDC profile. We also use many existing plug-ins for instance to package applications using our middleware for MID-Let phones and Android phones. In addition, in order to exercise our context management policies by simulation before deploying them in physical environments, we use the Siafu context simulator[8] for writing scenarios.

6.1.3. Dealing with various kinds of context information

We discuss in this section two application scenarios dealing with context information different from the location context considered by the Flash sale application and we present how the solution we propose in this article can be applied.

5 http://maven.apache.org.

6 http://www.eclipse.org/modeling/emf.

7 http://www.eclipse.org/Xtext.

8 http://siafusimulator.sourceforge.net.

Test case	Average (ms)	95% confidence interval
Leaf node	1.76	[1.704..1.821]
Non-leaf node	3.62	[3.594..3.635]
Non-leaf node with QoC	10.34	[9.980..10.699]

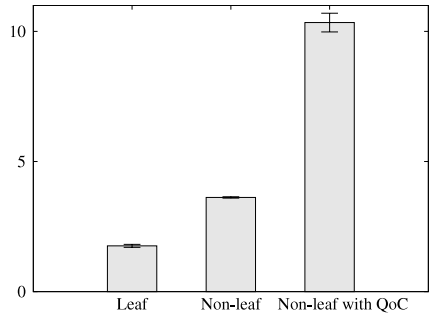

Fig. 11. Time for instantiating a context node.

We first describe an application example from the Ambient Assited Living domain inspired by [23]. A smart home is equipped with a video camera to determine the gait of a person. The person can be sitting, standing, walking, or lying. In the case of the elderly, this would help to determine an emergency situation where the person needs assistance after a bad fall or because they cannot move. The video camera is impacted by the amount of ambient light in the room. This requires measurement of the light level using a light meter. In this example, we have two *observables*: the video image and the light level. Following the design process we propose, the context-awareness designer will thus specify the QoC requirements and the QoC levels necessary to define the context-awareness contracts. Meta-data providing knowledge on the quality of the light level are required. Different QoC parameters such as the freshness or the spatial resolution of the light can be used. The QoC of the light level information will then be taken as input to determine the probability of correctness of the video image. This probability is low when the contrast between the background color and the color of the person's clothes is not sufficient. This happens when the light level is also low. The context-awareness designer will then define the COSMOS *context policy* involving the different context nodes required to collect the light level and then to compute the probability of correctness of the video image. If a change is detected in a video image, the probability of correctness of this video image is computed and compared to some threshold. If it is above the threshold, a notification is delivered to the smart home application.

As a second scenario, adapted from the PROSENSE European FP7 project,[9] we consider an Interactive Street Sensing (ISS) application for detecting hazardous pollution conditions in the street. Nodes with temperature, humidity, light, and pollutant gas sensors are deployed in the street and are equipped with short and long distance wireless communication capacities. Citizens willing to check the pollution level in the street, before their daily jogging for instance, can download the ISS mobile application on their mobile device in order to receive information from the sensors in their vicinity. A monitoring facility is also used by the city authorities to collect information in real-time and to save historic data of the pollution. Either for instantaneous data or for historic data analysis, QoC meta-data will help to determine consolidated information. Let us consider as an example the ozone level which is modeled as an observable in the ISS context-awareness model. As this pollutant gas can have a strong impact on people's health, its level must be measured carefully and therefore QoC meta-data will be associated to each measure. QoC levels will be defined using two QoC parameters: accuracy, as it can be provided directly by the gas meter physical sensor, and freshness, as it is important to know whether the measure is current or not. For instance, a QoC level named *relevant* is defined with: accuracy >0.8 and freshness >0.9.

6.2. Quantitative evaluation

This section reports and analyzes the evaluation results we have obtained with the COSMOS framework on a mobile phone. Firstly, we compare the performance results of elementary operations with and without QoC and secondly we present the performance results for the Flash sale application.

Evaluation of COSMOS elementary operations. We have first measured the response time of COSMOS elementary operations on context nodes. We have then evaluated the cost of adding QoC management to context nodes. We have also determined the maximum size of the context policy that can be deployed on a mobile phone. We herewith demonstrate that process-oriented context management can be performed entirely on mobile devices for on-the-move users without requiring interactions with a remote computer for context management purposes.

All the performance measures were conducted on an off-the-shelf mobile phone with the COSMOS framework deployed on it. The mobile phone was a Samsung Galaxy S2 phone, with the Android 2.3.5 operating system, an ARM Cortex-A9 dual-core processor at 1.2 GHz, 16 GB of ROM and 1 GB of RAM (660 MB initially available), running a native Java Virtual Machine 1.5.

We show in Fig. 11 the mean processing time required to create a context node, measured on 100 runs (the right hand side of the figure presents the results in a table, and we show the same results on the left hand side in the form of a histogram). A context node is a composite node. Fig. 12 shows the different composite nodes that we have instantiated

9 http://www.prosense-project.eu/.

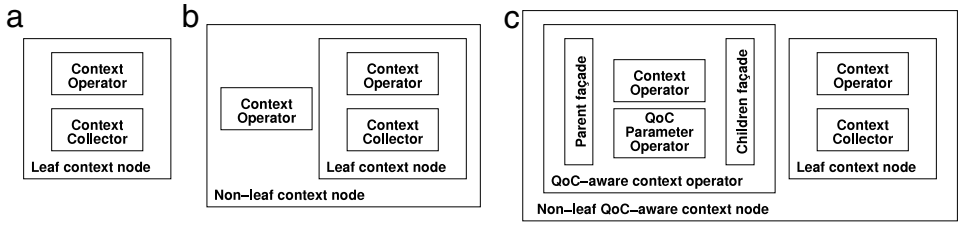

Fig. 12. Architecture of (a) a leaf context node, (b) a non-leaf context node, and (c) a non-leaf context node with QoC.

Test case	Average (ms)	95% confidence interval
Blocking	0.13	[0.121..0.139]
Non-blocking	0.4	[0.386..0.406]
Non-blocking — with QoC	0.82	[0.800..0.841]

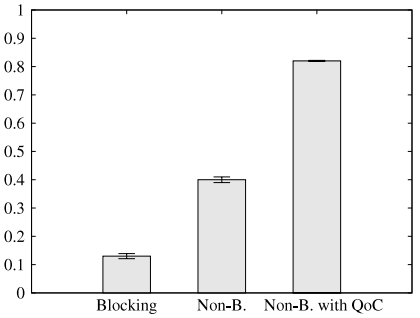

Fig. 13. Response time for the observation of a non-leaf node.

for this experimentation. As shown in the architecture depicted in Fig. 12(a), in the case of a leaf node, it requires the instantiation of 2 primitive components corresponding to a context collector to get context data from context sources and a context operator. As a leaf node corresponds to the collection of raw data, including QoC data, it does not perform any processing—that is there is no associated QoC computation at this level.

As an example of the instantiation of a non-leaf node, we created a composite node including a full leaf node, corresponding to a child node, and also a context operator. Fig. 12(b) depicts the corresponding architecture. A non-leaf node can also be configured to manage the QoC of the manipulated context information requiring additional sub-components, leading to a more complicated architecture as mentioned in Section 5.1 and as detailed in Fig. 12(c). First, recall that the manipulation of such an architecture is not described in many complicated component ADL files, but performed using COSMOS DSL. Then, observe that a QoC-aware context operator is the composition of a regular context operator with reusable QoC parameter operator components. In the experiment, there is one QoC parameter operator, and thus one QoCParameterOperator component. Therefore, for these measures, the leaf node, the non-leaf node and the non-leaf node with QoC contain respectively 2, 3 and 6 primitive components.

The time needed to create a leaf node is less than 2 ms, and for a non-leaf node it is less than 4 ms. With the additional components required to manage QoC, we reach an average of about 10.5 ms which is less than 3 times that for a standard non-leaf node. The difference is due to the additional creation of the component membranes for the composite components. This allows the creation of almost 100 QoC-aware context nodes in only 1 s.

Concerning the processing time, we have first measured the observation time for a leaf node. It is very low (below 0.3 ms) and corresponds to the execution of a collect action. For non-leaf nodes, we present in Fig. 13 the observation time in the blocking mode (the context data are consumed without a new computation) and the non-blocking mode (a new computation is made through the whole graph of components). In the blocking mode, QoC management does not have any impact as the most recent value is provided by the node and no processing is taking place (cf. Section 5.1). In the non-blocking mode, some overhead appears (0.4 ms) but remains very low. The overhead is due to more components leading to more computation and to more component membranes to go through.

From these measures, we can estimate the average processing time of a given context policy. This corresponds to the sum of the processing times of all the paths that need to be crossed from the root context node of the context policy to the "next" child nodes that block observations. For instance, this indicates that in less than 100 ms we can process 1 path of 121 nodes or about 20 paths of 5 nodes.

Maximum size of a context policy. We now investigate what the limits of the COSMOS framework are in terms of the size of the context policy that can be instantiated before exhausting the processing and memory resources of the mobile phone. We have first measured what the maximum number of non-leaf context nodes that can be instantiated for a context policy is. This has been tested for nodes without QoC and also for nodes with QoC management. The results are indicated in Table 1. We instantiated a context policy without QoC management containing a maximum of 5500 nodes which is far more than the context policy defined for the Flash sale application. When nodes are extended to support QoC management, we could create more than half of the nodes of the previous case, with almost 3000 nodes.

Table 1
Maximum number of nodes instantiated.

Test case	Max. number of nodes
Without QoC	5500
With QoC	2999

Table 2
Maximum height between two non-blocking nodes.

Test case	Max. height
Without QoC	45
With QoC	19

Test case	Average (ms)	95% confidence interval
Without QoC	69.53	[67.77..71.31]
With QoC	99.28	[97.04..101.53]

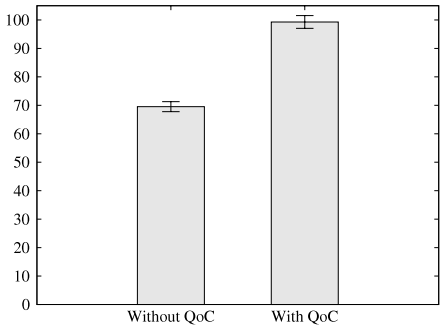

Fig. 14. Time for instantiating the Flash sale alert context policy.

Test case	Average (ms)	95% confidence interval
Without QoC	15.2	[14.87..15.52]
With QoC	16.27	[15.54..17.00]

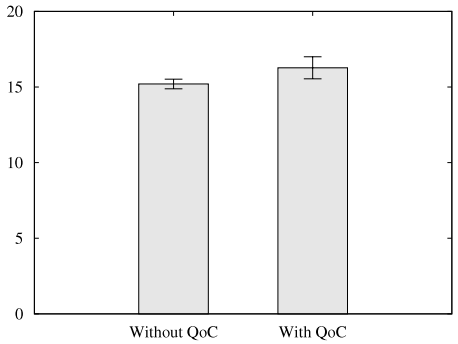

Fig. 15. Time for traversing the Flash sale alert context policy (worst case, that is all the context nodes are non-blocking).

We have also determined the maximum length of the path between two non-blocking nodes and give the results in Table 2. This corresponds to the maximum number of nodes to cross before the processing flow is stopped by a blocking node. Without QoC, there is a maximum of 45 nodes that can be crossed, while with QoC, we could cross 19 nodes in a row. This allows us to develop much more complex context policies than the Flash sale alert policy which requires to cross only 3 nodes between 2 blocking nodes.

Performance of the Flash sale application. The following measurements give the performance results obtained for the Flash sale application. Fig. 14 presents the instantiation time for the whole *Flash sale alert* context policy. With or without QoC management, the whole context policy is instantiated in less than 100 ms. This appears to be very reasonable and will not impact very much the deployment time on the mobile phone.

Fig. 15 indicates the observation time measured for the *Flash sale alert* context policy with and without QoC management in the worst case, that is when all the context nodes are non-blocking nodes. The remarkable point is that the overhead implied by QoC management on observation time is only of a few ms.

Discussion. With the performance evaluations presented in this section, we show that context management with the COSMOS framework can take place entirely on current mobile phone devices. The computation of QoC meta-data implies some additional cost, explained by the components added to the architecture, but this cost remains low. The figures presented in this section demonstrate the scalability of our approach when a high number of context nodes is necessary.

For instance, the total processing time to observe new context information and analyze it remains lower than 20 ms for the *Flash sale alert* policy.

7. Related work

Production of context-aware software is a very complex task. MDE for context-awareness and context-management is essential to ease the production of context-aware applications. We present below related works for context-awareness modeling and context management, with a particular focus on QoC.

Due to the variety of context data to be collected and analyzed, context management needs the support of abstract context modeling. The main families of context modeling are profiling (*e.g.*, CC/PP [24]), data-bases (*e.g.*, CML [19]), ontologies (*e.g.*, CONON [46]) and MDE. Our work aims at using MDE for defining links between context modeling to express complex context situations, and context-awareness modeling to associate context situations with application entities.

ContextUML [41] is one of the first domain specific models for context-awareness. It defines a meta-model for modeling context-awareness of Web services. Consequently, Web service elements such as *Service*, *Operation* and *Message* are represented in the model as well as related adaptation mechanisms of type *Binding* or *Triggering*. FamiWare [16], a family of middleware for ambient intelligence, relies on ContextUML for modeling context-awareness: it provides an automatic mapping from ContextUML to FamiWare and allows for self-adaptation of the middleware to context changes. At design time, feature models allow middleware configurations to be derived. These feature models are then used at runtime to drive the middleware reconfiguration. Ref. [17] extends this work to customize a middleware platform for a particular device so that it fulfills specific application requirements. The authors follow a Software Product Line (SPL) engineering approach. CAPucine [31] describes an MDE approach for dynamically producing product lines according to context information. Kulkarni et al. [26] present a generative programming framework for context-aware cooperative applications. ContextUML, FamiWare, CAPucine and the Kulkarni proposition put the stress on adaptation mechanisms rather than on modeling context with its associated meta-data like QoC. Our work enables application designers to express interpreted context such as situations computed from distributed context observations and include the analysis of the QoC for driving the context-awareness of the application.

MLContext [21] is a DSL for modeling context. It is extended in [22] and integrates QoC in the expression of situations. A situation is then detected if the context data it depends on fulfill the required QoC. The model presented in our article goes one step further as it also takes into consideration the context-awareness aspect of the business application and its specific QoC requirements. Therefore, the middleware which uses the model is also able to connect detected situations to appropriate business services according to the QoC levels.

Manzoor et al. [29] propose an objective view of QoC, independent of any application requirement, and a subjective view of QoC, considering its worth for a specific context requirement. Our context-awareness meta-model is also generic and independent of the applications, while the context-awareness model takes into account the requirements of a given application. However, the differentiating aspect of our work consists in a model-driven approach to guide application developers all along the software lifecycle from the design stage to the execution stage as the context-awareness meta-model and model can be accessed at runtime [44].

Regarding context management, two main approaches exist: process-oriented context management (PCM) and ontology-based context management (OCM). Concerning the PCM approach, many frameworks have been proposed and have become references in the domain of ubiquitous computing like the Context Toolkit [15], the Contextor [12], Draco [33], MoCA [13] or MoCoA [40]. However, context management frameworks integrating and manipulating QoC are only beginning to appear. Concerning the ontology approach, Ref. [4] presents a survey on context modeling and reasoning. With the semantic Web trend, several frameworks have been proposed, among them SOUPA [9], SOCAM [46] and COSAR [36]. Handling QoC with ontology reasoning is highlighted by [4]. For this purpose, the meta-context proposal [20] is promising. It complements context domain ontologies with time, owner and reliability domains.

We have compared the PCM and OCM approaches in [5] and we have shown that they are complementary. On the one hand, an ontology-based approach puts the stress on reasoning from knowledge bases and includes learning mechanisms. On the other hand, a process-oriented approach is better suited for handling dynamic context with a low latency for identifying situation changes and for performing well under the resource constraints of mobile devices. As one of the objectives of the presented work is the processing of context on mobile devices, we have chosen the PCM approach.

After having presented related works concerning model-driven engineering for context-awareness and context management in general, we now consider works focusing on context management taking into account the quality of the context information. One can notice that these approaches are only targeted at the manipulation of location context. Even though our prototype application takes location information as an example, the approach we present in this article is not restricted to this kind of context as enabled by the general context-awareness meta-model we propose.

Middlewhere [34] relies on three metrics for determining the quality of location information: resolution, confidence and freshness. It proposes an uncertainty model based on a predicate representation of contexts allowing the use of mechanisms such as probabilistic logic, fuzzy logic and Bayesian networks to combine multiple sensory data. However, the resulting quality of location information is not exposed to the applications and the models cannot easily be extended by application developers. On the contrary, we consider that applications must be informed of the quality of the context information manipulated by the context manager and that the context models must easily be extendible.

Nexus [27] is an open platform to ease the development of location-aware applications. It considers three quality aspects which are degradation, consistency and trust. As in our approach, Nexus considers uncertainty as a key factor of location information and proposes a generic mathematical uncertainty model for position information [28]. This model is very powerful but requires application designers to specify probabilities in order to perform position queries. We propose a more user-friendly solution where the framework informs the user of the obtained context quality rather than requiring the user to restrict the search domain.

The LOC8 framework [42] is a recent effort to provide application developers with easy access to location information. The focus is on a clear separation of the three concerns of mapping space, working with positioning systems and querying data. LOC8 emphasizes quality meta-data. It defines a quality matrix consisting of granularity, frequency, coverage and a list of accuracy and precision pairs. LOC8 also relies on a sensor fusion method, with a default implementation based on fuzzy logic integrating the confidence on location data. Our work has similar motivations: manipulating different sensor data and exposing the knowledge of its quality. However, we promote a context processing that considers a larger set of quality criteria, and not only confidence.

8. Conclusion

In this article, we promote a model-driven approach for designing QoC-aware ubiquitous applications. We propose a complete and fully integrated tool chain encompassing a context-awareness meta-model (CA3M) and a domain specific language for context management (COSMOS DSL). The `cosmos2code` tool automatically generates the code of the artefacts deployed at runtime. We demonstrate our contributions through a Flash sale application. As for numerous mobile distributed applications, location-awareness is essential for the Flash sale application. We consider that location information requires specific care to deal with its inherent uncertainty and that applications need to have the knowledge of this uncertainty level. We identify accuracy, freshness and trustworthiness as being the quality criteria that are particularly relevant for location information, but other QoC parameters are provided by our context management service and this list can be extended at will. With this demonstrator, we show that QoC may be used at different levels: at the context management level, for instance to choose the best location among several ones, and at the application level, for instance to trigger the appropriate service according to the current context situation and its QoC level. This justifies the model-driven approach that we have followed from the specification of context-awareness contracts to the design of the architecture of the context manager that runs on the mobile phone of the end-user.

We present extensive performance results in terms of memory and processing time. The cost of the QoC management is also measured and appears to be limited to a few milliseconds. We show that a context policy with 120 QoC-aware nodes can be processed in less than 100 ms on a mobile phone. Moreover, we could create almost 3000 nodes before exhausting the resources of the phone. This enables very rich application scenarios enhancing the user experience and will favor the development of new ubiquitous applications.

As future work, we will apply the model-driven approach to other concerns of context-awareness management, for instance to enhance the privacy of personal context data during the whole context management process. In addition, since it is a process-oriented component-based context manager, we view COSMOS as a natural basis for distributing both processing flows of context data and their QoC meta-data. This clearly opens new research directions for QoC management.

References

[1] Z. Abid, S. Chabridon, D. Conan, A framework for quality of context management, in: Proc. 1st Int. Workshop on Quality of Context, Stuttgart, Germany, June, in: LNCS, vol. 5786, Springer-Verlag, 2009.
[2] M. Baldauf, S. Dustdar, F. Rosenberg, A survey on context aware systems, International Journal of Ad Hoc and Ubiquitous Computing 2 (4) (2007) 263–277.
[3] B. Beamon, M. Kumar, Adaptive context reasoning in pervasive systems, in: Proc. 9th Workshop on Adaptive and Reflective Middleware, ARM 2010, held at the ACM/IFIP/USENIX Int. Middleware Conference, Bangalore, India, November 2010.
[4] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, D. Riboni, A survey of context modelling and reasoning techniques, Pervasive and Mobile Computing 6 (2) (2010) 161–180.
[5] A. Bouzeghoub, C. Taconet, A. Jarraya, N.K. Do, D. Conan, Complementarity of process-oriented and ontology-based context managers to identify situations, in: Proc. 5th International Conference on Digital Information Management, Thunder Bay, Canada, June 2010.
[6] T. Buchholz, A. Kupper, 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, Geneva, Switzerland, July 2003.
[7] S. Chabridon, Z. Abid, C. Taconet, D. Conan, A model-driven approach for the QoC-awareness of ubiquitous applications, in: UCAmI '11 : 5th International Symposium on Ubiquitous Computing and Ambient Intelligence, Riviera Maya, Cancun, Mexico, December 2011.
[8] S. Chabridon, C.-C. Ngo, Z. Abid, D. Conan, C. Taconet, A. Ozanne, Towards QoC-aware location-based services, in: P. Felber, R. Rouvoy (Eds.), Proc. 11th IFIP International Conference on Distributed Applications and Interoperable Systems, Reykjavik, Iceland, June, in: LNCS, vol. 6723, Springer-Verlag, 2011.
[9] H. Chen, T. Finin, A. Joshi, Semantic web in the context broker architecture, in: Proc. 2nd IEEE International Conference on Pervasive Computing and Communications, PERCOM '04, Washington, DC, USA, IEEE Computer Society, 2004, pp. 277–286.
[10] D. Conan, R. Rouvoy, L. Seinturier, Scalable processing of context information with COSMOS, in: Proc. 6th IFIP International Conference on Distributed Applications and Interoperable Systems, Cyprus, June, in: LNCS, vol. 4531, Springer-Verlag, 2007, pp. 210–224.
[11] J. Coutaz, J.L. Crowley, S. Dobson, D. Garlan, Context is key, Communications of the ACM 48 (3) (2005) 53.
[12] J. Coutaz, G. Rey, Foundations for a theory of contextors, in: C. Kolski, J. Vanderdonckt (Eds.), Proc. 4th International Conference on Computer-Aided Design of User Interfaces, Valenciennes, France, May, Kluwer, 2002, pp. 13–34.
[13] R.C.A. da Rocha, M. Endler, Evolutionary and efficient context management in heterogeneous environments, in: Proc. 3rd International Workshop on Middleware for Pervasive and Ad-hoc Computing, Grenoble, France, November 2005.

[14] A.K. Dey, G.D. Abowd, Towards a better understanding of context and context-awareness, in: Proc. CHI Workshop on the what, who, where, when, and how of context-awareness, pp. 304–307, 2000.
[15] A.K. Dey, G.D. Abowd, D. Salber, A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications, in: Context-aware Computing, Human-Computer Interaction Journal 16 (2) (2001) 97–166 (special issue).
[16] N. Gámez, J. Cubo, L. Fuentes, E. Pimentel, Modeling context-awareness in FamiWare, in: Proc. 5th International Symposium on Ubiquitous Computing and Ambient Intelligence, Riviera Maya, Cancun, Mexico, December 2011.
[17] N. Gámez, L. Fuentes, FamiWare: a family of event-based middleware for ambient intelligence, Personal and Ubiquitous Computing 15 (2011) 329–339.
[18] K. Henricksen, J. Indulska, Modelling and using Imperfect Context Information, in: Proc. 1st PerCom Workshop CoMoRea, pp. 33–37, March 2004.
[19] K. Henricksen, J. Indulska, Developing context-aware pervasive computing applications: models and approach, Pervasive and Mobile Computing 2 (1) (2006) 37–64.
[20] R. Hervás, J. Fontecha, V. Villarreal, J. Bravo, Meta-context: putting context-awareness into context, in: Andreas König, Andreas Dengel, Knut Hinkelmann, Koichi Kise, Robert J. Howlett, Lakhmi C. Jain (Eds.), Proc. 15th International Conference, Knowlege-Based and Intelligent Information and Engineering Systems, KES 2011, in: Lecture Notes in Computer Science, vol. 6882, Springer-Verlag, 2011, pp. 296–305.
[21] J.R. Hoyos, J. Garcìa-Molina, J.A. Bota, MLContext: a context-modeling language for context-aware systems, in: 3rd DisCoTec Workshop on Context-aware Adaptation Mechanisms for Pervasive and Ubiquitous Services, Amsterdam, Netherlands, June 2010.
[22] J.R. Hoyos, D. Preuveneers, J.J. García-Molina, Y. Berbers, A DSL for context quality modeling in context-aware applications, in: Ambient Intelligence - Software and Applications: 2nd International Symposium on Ambient Intelligence, volume 92, pages 41–49. Springer-Verlag, April 2011.
[23] M. Huebscher, J. McCann, An adaptive middleware framework for context-aware applications, Personal and Ubiquitous Computing 10 (1) (2006) 12–20.
[24] G. Klyne, et al., Composite Capability/Preference Profile (CC/PP): Structure and vocabularies 2.0. W3C recommendation, April 2007.
[25] M. Krause, I. Hochstatter, Challenges in modelling and using quality of context (QoC), in: T. Magedanz, et al. (Eds.), Mobility Aware Technologies and Applications, in: LNCS, vol. 3744, Springer-Verlag, 2005, pp. 324–333.
[26] D. Kulkarni, T. Ahmed, A. Tripathi, A generative programming framework for context-aware CSCW applications, ACM Trans. on Software Engineering Methodology 21 (2) (2012).
[27] R. Lange, et al., Making the world wide space happen: new challenges for the nexus context platform, in: Proc. 7th IEEE International Conference on Pervasive Computing and Communications, Galveston, TX, USA, March, IEEE, 2009, pp. 300–303.
[28] R. Lange, et al., On a generic uncertainty model for position information, in: Proc. of 1st Int. Workshop on Quality of Context, Stuttgart, Germany, June, Springer-Verlag, 2009, pp. 76–87.
[29] A. Manzoor, H.-L. Truong, S. Dustdar, Quality of context: models and applications for context-aware systems in pervasive environments, in: Web and Mobile Information Services, The Knowledge Engineering Review (2011) (special issue).
[30] Object Management Group. Software & Systems Process Engineering Metamodel (SPEM) v2.0. Formal/2008-04-01, April 2008.
[31] C. Parra, X. Blanc, A. Cleve, L. Duchien, Unifying design and runtime software adaptation using aspect models, Science of Computer Programming 76 (12) (2011) 1247–1260.
[32] N. Paspallis, R. Rouvoy, P. Barone, G.A. Papadopoulos, F. Eliassen, A. Mamelli, A pluggable and reconfigurable architecture for a context-aware enabling middleware system, in: Proc. 10th International Symposium on Distributed Objects and Applications, Monterrey, Mexico, November, in: LNCS, vol. 5331, Springer-Verlag, 2008, pp. 553–570.
[33] D. Preuveneers, Y. Berbers, Adaptive context management using a component-based approach, in: L. Kutvonen, N. Alonistioti (Eds.), Proc. 5th IFIP International Conference on Distributed Applications and Interoperable Systems, Athens (Greece), June, in: LNCS, vol. 3543, Springer-Verlag, 2005, pp. 14–26.
[34] A. Ranganathan, J. Al-Muhtadi, S. Chetan, R. Campbell, M.D. Mickunas, MiddleWhere: a middleware for location awareness in ubiquitous computing applications, in: H.-A. Jacobsen (Ed.), Proc. IFIP/ACM/USENIX Middleware, Toronto, Canada, October, in: LNCS, vol. 3231, Springer-Verlag, 2004, pp. 397–416.
[35] J. Ribault, O. Dalle, D. Conan, S. Leriche, OSIF: a framework to instrument, validate, and analyze simulations, in: Proc. 3rd International ICST Conference on Simulation Tools and Techniques, Malaga, Spain, March 2010.
[36] Daniele Riboni, Claudio Bettini, COSAR: hybrid reasoning for context-aware activity recognition, Personal Ubiquitous Comput. 15 (3) (2011) 271–289.
[37] M.D. Rodríguez, J. Favela, Assessing the SALSA architecture for developing agent-based ambient computing applications, Science of Computer Programming 77 (1) (2012) 46–65.
[38] R. Rouvoy, D. Conan, L. Seinturier, Software architecture patterns for a context processing middleware framework, IEEE Distributed Systems Online 9 (6) (2008).
[39] N. Samama, Global Positioning: Technologies and Performance, Wiley-Interscience, 2008.
[40] A. Senart, R. Cunningham, M. Bouroche, N. O'Connor, V. Reynolds, V. Cahill, MoCoA: customisable middleware for context-aware mobile applications, in: Proc. 8th International Symposium on Distributed Objects and Applications, Montpellier, France, November, in: LNCS, vol. 4275, Springer-Verlag, 2006, pp. 1722–1738.
[41] Q.Z. Sheng, B. Benatallah, ContextUML: a UML-based modeling language for model-driven development of context-aware web services, in: Proc. 4th IEEE ICMB, pp. 206–212, Sydney, Australia, July 11–13 2005.
[42] G. Stevenson, J. Ye, S. Dobson, P. Nixon, LOC8: a location model and extensible framework for programming with location, IEEE Pervasive Computing 9 (2010) 28–37.
[43] C. Taconet, Z. Kazi-Aoul, Building context-awareness models for mobile applications, Journal of Digital Information Management 8 (2) (2010) 78–87.
[44] C. Taconet, Z. Kazi-Aoul, M. Zaier, D. Conan, CA3M: a runtime model and a middleware for dynamic context management, in: Robert Meersman, Tharam Dillon, Pilar Herrero (Eds.), Proc. 11th International Symposium on Distributed Objects and Applications, Vilamoura, Algarve, Portugal, November, in: LNCS, vol. 5870, Springer-Verlag, 2009, pp. 513–530.
[45] R. Vera, S. Ochoa, R. Aldunate, EDIPS: an easy to deploy indoor positioning system to support loosely coupled mobile work, Personal and Ubiquitous Computing 15 (2011) 365–376. http://dx.doi.org/10.1007/s00779-010-0357-x.
[46] X. H. Wang, D. Q. Zhang, T. Gu, H. K. Pung, Ontology based context modeling and reasoning using OWL, in: Proc. 2nd IEEE International Conference on Pervasive Computing and Communications, pp. 18–22, Orlando, FL, USA, March 2004.
[47] T. Zimmer, QoC: quality of context improving the performance of context-aware applications, in: Advances in Pervasive Computing 2006, Adjunct Proceedings of Pervasive 2006, Dublin, 7–10 May, 2006.