Towards a Context-Aware Service Design and Development Paradigm

Hatim Hafiddi, Hicham Baidouri, Mahmoud Nassar, Bouchra El Asri and Abdelaziz Kriouile

SI2M Laboratory, ENSIAS
Rabat, Morocco

{Hatim.HAFIDDI, Hicham.Baidouri}@gmail.com, {nassar, elasri, kriouile}@ensias.ma

Abstract
Today’s evolution of Ubiquitous Computing and Service Oriented Computing is leading to the development of Context-Aware Services. However, the ambiguity of the context concept and the multiplicity of services’ execution contexts make CASs hard to build and show why there is a lack of generic approaches for designing such services. In this paper, we first present our context metamodel based on a practical specification of the context. Then, based on this metamodel, we propose a Context-Aware Service specification and metamodel and show how aspect paradigm can be used to fulfill service adaptation to his execution contexts.

Key words: Context-Aware Service Oriented Architecture, Model Driven Engineering, Context-Aware Service, Context View, Adaptation Strategy.

1. INTRODUCTION

The federation of the joint evolutions, in the fields of telecommunication (fast networking protocols), of mobile infrastructures (new generations of mobile terminals) and of software engineering in term of architectures (emergence of new architectures like Service Oriented Architectures) and in term of development paradigms (from the functional to the service while passing by the object and component paradigms), promoted the birth of a new generation of information systems known as Context-Aware systems whose architecture is articulated on a new development paradigm called Context-Aware Service (see Fig. 1). A Context-Aware Service (CAS) provides users with a customized and personalized behaviour depending on their contexts. For example, a Restaurants Searching service gives users suggestions depending on their locations, preferences and even the used device capabilities. Generally, this kind of information is called context.

The ambiguity of the context concept and the multiplicity of services’ execution contexts make CASs hard to build and show why there is a lack of universally accepted basic design and development principles that can lead to a generic approach towards efficient CASs development. The traditional approaches for CAS implementation produce services which are able to function only in preset situations and whose business logic is tightly coupled with both of context management and adaptation logics. Thus, the result of such approaches is complex services, which generally depends on the technical platforms and the application domains, whose rate of evolution and re-use is reduced.

Nowadays, designing systems based on CAS enables them to sense and react to changes observed in their environment. This capability is particularly critical in ubiquitous environments, where context is the central element of mobile systems [19]. Though we base our remarks in this article on a specific application domain (i.e. E-tourism), we follow a Model Driven Engineering (MDE) approach for context and CAS specification and development (Platform & Domain Independent Specification and Development: PDISD). Model-Driven Engineering (MDE) is a model centric approach for software development in which models are used to drive the development of all software artifacts. It provides great benefits in terms of cost reduction and quality improvement. Our approach consists on providing context and CAS metamodels which will serve for constructing context and CAS models, then, context and CAS implementations can be generated automatically by models transformation. Thereby, in addition of profits in terms of re-use, evolution, integration and maintenance, our approach for context and CAS specification and development will be easily transposed to various domains and targets various technical platforms.

The reminder of this paper is organized as follows. We present in next section a scenario that concerns an E-tourism...
system, which will be used in subsequent sections as an illustrating example. In Sect. 3, we present and describe our context specification and metamodel. Sect. 4 presents our CAS specification and metamodel. We introduce in Sect. 5 how the aspect paradigm can be applied to CAS design. Sect. 6 briefly compares related research. Finally, we conclude the paper in Sect. 7 with plans for future work.

2. E-TOURISM MOTIVATING SCENARIO: RESTAURANTS SEARCHING SERVICE

Let’s imagine that a Swedish tourist wants to taste the local gastronomy of a Moroccan city which he’s visiting, so he connects himself via his mobile terminal (e.g., PDA, iPhone gold Blackberry, etc.) to a traditional E-tourism system in order to obtain a list of suitable restaurants. He subscribes to the service, launches his request and obtains one of the two following answers:

- Service failure (i.e. the system blocks and the application closes) because of its inadequacy for a mobile use (i.e. memory overloads considering the great number of returned records);
- In the contrary case (i.e. limited number of returned records), the service returns an inadequate answer for tourist expectations (i.e. inadequate display and inappropriate restaurants because the system doesn’t take into account parameters like tourist’s terminal type, his position, his language, his preferences, etc.).

This scenario shows that the E-tourism system needs to be context-aware in order to use user’s context and face its changes. Indeed, if such system was conceived to be Context-Aware, the tourist once connected to the system will receive automatically (time is taken into account: it’s midday for example) a list of restaurants well presented (terminal type is taken into account for display adaptation), close to his site (taken into consideration the localization), described in his language (system will consider the user’s language) and taking account of his preferences (food preferences for instance). Also, let’s note that such system will resort to a results pagination mechanism (considering as well the terminal capacities, RAM in this case) to avoid its blocking and if ever it detects any change in tourist’s context (e.g., weak battery or change of the connection type), it will automatically adapt his behaviour (e.g., passage to a reduced view) in purpose of optimization.

3. CONTEXT

Context is the information that characterizes the interactions between humans, applications, and the environment [21]. Context information is dependent on system’s domain, as a type of information might be considered as context information in one domain but not in another one. So, several context definitions were proposed in the literature [7, 27] serving various domains, however the context definition, given by Dey and Abowd, remains the most generic. Indeed, these authors have defined context as "any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves" [28]. As given in [8], we consider context parameters as any additional information that can be used to improve the behaviour of a service in a situation. Without such information, the service should be operable as normal but with context information, it is arguable that the service can operate better or more appropriately [12].

Rather than giving context formalization, case of figure for several researches on this topic, sometimes domain specific and sometimes general but not very extensible, we chose to propose a metamodel which is, at the same time, generic and abstract (see Fig. 2). This metamodel is based on the following specification:

- A context decomposes into sub contexts;
- A sub context can be, recursively, decomposed into categories for its structuring;
- A context, a sub context and a category are constituted of parameters;
- A parameter is simple, derived or complex;
- A derived parameter is obtained by derivation from a set of parameters;
- A complex parameter can have a representation as it can be, in its turn, recursively constituted of other complex parameters;
- A context view (i.e. a set of parameters) can have a semantic;
- An entity (e.g., service, user, terminal, etc.) is described by a set of parameters.

Fig. 2 Context metamodel

As we already mentioned, our context metamodel can be extended to include other concepts such as parameters’ quality which is a key factor, to take into account if that is necessary,
in the view of the fact that the parameters’ quality influences the service behaviour’s quality.

To illustrate our metamodel, let’s project it on the case of figure of the E-tourism system presented in the second section. The context for this system in particular and Context-Aware data processing in general is composed mainly of the following sub contexts (see Fig. 3):

- **DeviceSubContext**: it contains the parameters which describe the entity device. It breaks up into two categories which are the software category (e.g., operating system, navigator type, supported type of data, etc.) and the hardware one (e.g., processor type, screen size, battery level, memory size, etc.);
- **UserSubContext**: it’s a sub context which contains the parameters describing the entity user (e.g., preferences, localization, profile, etc.);
- **D-EnvironmentSubContext**: this sub context contains the parameters of environment (e.g., time, weather, etc.);
- **ServiceSubContext**: in its turn, this sub context contains the parameters which characterize a service (e.g., price, availability, response rate, response time, etc.);

For reasons of organization and facility of context management, we structure our context model in packages as illustrated in Fig. 4. Some parameters (e.g., device type, localization, service price, etc.) are common to any ubiquitous system. Thus, they are defined in non domain specific sub context (i.e., ServiceSubContext, EnvironmentSubContext, UserSubContext and DeviceSubContext), while others are specific to application domain (E-tourism in our case); so, they are placed in DomainSpecificSubContext. Let’s stress that the separation of sub context’s information into Static information (e.g., user profile and interests) and Dynamic ones (e.g., time, location and direction) is relevant (see Fig. 4):

Finally, to support the evolution and the re-use of the packages ServiceSubContext, DeviceSubContext, UserSubContext and D-EnvironmentSubContext, in other application domains, it’s necessary to reduce their tight coupling with the DomainSpecificSubContext package, as illustrated in the Fig. 4, by reducing navigation between classes of various packages and proceeding to a dependence inversion so that the DomainSpecificSubContext package depends on the other ones and not the reverse (see Fig. 5).

4. CONTEXT-AWARE SERVICE

One of the first uses of the term context-aware appeared in 1994 [18]. A service is Context-Aware if it provides customized and personalized behaviour to users depending on their contexts [28, 3 and 11]. In Service Oriented Computing (SOC), a service is defined as self-contained and platform-agnostic computational element that supports rapid, low-cost and easy composition of loosely coupled and distributed software applications [25]. To be Context-Aware, a service must be able to adapt dynamically its behaviour to its several
execution (i.e. use) contexts. In other words, the service (i.e. core service) must possess mechanisms in purpose to exploit only relevant information of the execution context and to adapt dynamically its behaviour. Henceforth, these appropriate context information relative to a specific execution situation forms what we call the ContextView of the service and the result of service adaptation to this ContextView forms the ContextViewService (see Fig. 6).

![Fig.6 Core service adaptation to its various ContextViews](image)

Fig. 7 illustrates our CAS metamodel. This metamodel is based on the following specification:

- Both ContextAwareService and ContextViewService are specific services;
- A ContextAwareService possesses a CASAdaptationStrategy which concerns a set of ContextViews;
- A ContextViewService possesses a CVSAdaptationStrategy which concerns one ContextView;
- A CASAdaptationStrategy aggregates a set of CVSAdaptationStrategies;
- For a given CVSAdaptationStrategy and ContextView, a set of AdaptationConditions is deducted;
- An AdaptationCondition can involves the execution of an ordered set of Adaptations;
- For a given CVSAdaptationStrategy and Adaptation, an AdaptationRule is associated;
- A CVSAdaptationStrategy aggregates a set of AdaptationConditions, Adaptations and AdaptationRules.

![Fig. 7 CAS metamodel](image)

Thus, CAS is seen as a specific service with a number of ContextViews. For each one, we associate an adaptation strategy (i.e. CVSAdaptationStrategy) which indicates when (i.e. AdaptationCondition: classical condition expressed on ContextView’s parameters) and how (i.e. AdaptationRule: defines the places in the service where the dynamic adaptations will be realized) a set of ordered adaptations (i.e. Adaptation) must be applied on the core service in order to provide the expected behaviour regarding the current execution context. The adaptation result forms the ContextViewService (see Fig. 9). So, for a given service, the set of its ContextViewServices (respectively CVSAdaptationStrategys) forms the CAS (respectively CASAdaptationStrategy).

For instance, in the e-tourism motivating scenario (see Sect. 2), battery level represents one of the Restaurants Searching service’s ContextViews which can provoke service adaption by reducing the amount of data transferred (i.e. Adaptation) whatever this level is lower than 20% (i.e. AdaptationCondition). The figure below presents a succinct CAS model in the case of Restaurants Searching service.

![Fig. 8 Succinct CAS model for the Restaurants Searching service](image)

5. CAS DEVELOPMENT ARTIFACT

Traditional approaches used for CAS design and development present several problems. In fact, simple core service duplication for each ContextView is a software engineering anti-pattern (e.g., high-cost of maintenance) as far as integrating adaptations logic into core service makes it complex and decreases his ability to be reused. So, in order to rationalize development and maintenance of CAS, we have to resort to new mechanisms and strategies that allow core service extension without any duplication or regression risks. These mechanisms will favor loosely coupling between the core service and its adaptations seen as transversal preoccupations.

Aspect Paradigm [6] allows the modification of applications with so-called aspects. Aspects are modular units of functionality which are used across the application’s code. They are woven into an application’s code at so-called pointcuts, thereby allowing to transparently extend, e.g., objects with new functionalities.

Inspired by Separation of Concerns [24] and Aspect Paradigm concepts, our CAS design and development approach consists on considering the Adaptation as an aspect.
So, the core service focuses only on the business logic and all of its Adaptations relatives to its ContextViews will be defined separately as aspects called Adaptation Aspects. These Adaptation Aspects will be dynamically weaved at runtime into core service by our tool named Adaptation Aspects Weaver (A2W) (see Fig. 10), in order to produce the expected ContextViewService (see Fig. 9).

![Fig. 9 CV service as Core Service adaptation to a specific ContextView](image)

Fig. 9 CV service as Core Service adaptation to a specific ContextView

The figure below illustrates the mechanism behind our A2W tool. A2W operates by recuperating the CASAdaptationStrategy of the executed service then inspects it in order to retrieve and interpret only the CVSAdaptationStrategy corresponding to the current ContextView. The interpretation mechanism consists in checking the AdaptationConditions in order to weave only the suited Adaptation Aspects, following a set of AdaptationRules, into core service to produce the corresponding ContextViewService (see Fig. 9).

![Fig. 10 Adaptations Aspects Weaver](image)

Fig. 10 Adaptations Aspects Weaver

Let’s mention that we don’t deal in this paper with the architecture of our A2W tool but it’s very interesting to stress that our CAS development approach combined to the A2W tool provide, in addition to dynamic service adaptation to the context, the ability to evolve service behaviour during the CAS life cycle.

6. RELATED WORK

Based on Dey’s definition of context [28], several research efforts in the domain of context modeling have been addressed. The main families of context modeling are Key-value pairs [16], databases (e.g., CML [17]), ontologies (e.g., Omnipresent [2], CMF [5]) and profiling (e.g., CC/PP [10]). The main objective of these researches is to provide a high level abstraction of context information to evolve easy context management and they don’t deal, in general, with adaptation (e.g., CAS development). For instance, in the omnipresent project [2], the context is modeled using OWL and is grouped in four categories which are: user profile, emotion and physiologic state, location and task. Other researches have tried to simplify Context-Aware systems development in an artisanal manner (i.e., without giving for such systems a modeling and developing process) by providing frameworks and middleware such as: context Toolkit [1], CoBrA [4], K-Components [23] and CORTEX [13]. Even if these frameworks and middleware simplify context-aware systems development by decoupling context management from adaptation logic, they suffer from a lack of well designed approach and introduce several technical details reducing such systems portability.

In this section, we focus on some projects which specialize on context and CAS meta-modeling. An important effort is the work conducted by the Sheng and Benatallah team in ContextUML project [15] which defines a metamodel for modeling Context-Aware Web Services. Though the ContextUML approach seems to be the same of our approach, however they differ in several points. First, our approach is platform independent and can be specialized to Web Services platform or other ones. Secondly, unlike our meta-models that give the service layer more abstraction of the context one, the proposed metamodel in ContextUML makes service and context tightly coupled. Also, Sheng and Benatallah don’t specify the mechanism used to fulfill CAS adaptation which is done in our work by the A2W tool. Keidl and Kemper [29] propose a context framework for the development and deployment of context-aware adaptable Web Services. In the framework, context is limited to the information of service requesters and the approach is platform dependent. Another important domain concerns product-line engineering which has a great potential in modeling service variability. Core assets are the reusable units in PLE where they embed common and variable features in a product family. System variations are defined in [9] using Feature-Oriented Domain Analysis (FODA) which proposes features as the unit to represent system characteristics [20]. The features are represented with three kinds of relationships; mandatory, alternative, and optional. Context in Scatter research is limited to the Device entity and the computed product line is undeterminist. Another important work is the one conducted in [14]. Authors deal with context-aware adaptation in DSOPL by proposing two different processes for the initial and iterative phases of product derivation. For the initial phase, authors define a process that integrates multi-view assets and propose domains separation. Concerning the iterative phase,
authors propose a context-aware asset that defines the information of an adaptation in response to context change and use the COSMOS [26] and FraSCAti [22] tools for realization technique for context-aware variability. The main challenge to be faced is this work is to reduce non-deterministic behaviours when non-deterministic context-aware assets are introduced.

7. CONCLUSION AND PERSPECTIVES

In this article, we followed a MDE approach to realize both CAS and context metamodels independently of the platforms and the application domains (PDISD). Our approach provides a high level of abstraction and has several advantages. Despite of the benefits of productivity and quality improvement of context and CAS development, context and CAS models can be applied for different platforms and technologies by defining models transformation rules. Thus, we presented, firstly, our context specification as a base for the context metamodel which is generic and open to allow its extension to various domains depending on needs. Our context metamodel is mainly composed of several kind of parameters well structured in sub contexts and other artifacts. Secondly, we proposed a CAS specification and metamodel and an approach that, based on the separation of concerns (e.g., Aspect Paradigm), considers the adaptations to a current execution context as Adaptation Aspects dynamically woven by our A²W tool at runtime. Our CAS metamodel highlights the notions of adaptation strategy, adaptation, adaptation condition and adaptation rule.

We focused in this article on the context and CAS specifications and metamodels and proposed an adaptation approach. In our future work, we project to provide, in the short term, an applicative layer of Context handling which will allow the collection and the transmission of ContextViews to A²W.

In the long term, our objective is to propose a framework (Context-Awareness module + A²W) allowing the CAS development. We target mainly the Web services as a technical platform for implementing Context-Aware Service oriented Architectures (CASOA). As we have already mentioned, we will base essentially our CASOA development process on models transformation by defining a set of transformation rules, using for example ATL, in order to automate the code generation.

8. ACKNOWLEDGEMENTS

We’d like to thank the anonymous reviewers for their comments and feedback.

REFERENCES