A CONTEXT-AWARE SERVICE CENTRIC APPROACH FOR SERVICE ORIENTED ARCHITECTURES

Hatim Hafiddi, Mahmoud Nassar, Hicham Baidouri, Bouchra El Asri and Abdelaziz Kriouile
IMS Team, SIME Laboratory, ENSIAS, Rabat, Morocco

Keywords: Context, Ubiquitous computing, Context-Aware Service Oriented Architectures, Model Driven Engineering, Aspect paradigm.

Abstract: Evolution in the fields of telecommunication and software engineering has promoted the birth of a new generation of software architectures known as Context-Aware Service Oriented Architectures (*CASOA*) which are articulated on a new design and development paradigm called Context-Aware Service (CAS). However, the ambiguity of the context concept and the multiplicity of services execution contexts make CAS hard to build and show why a generic approach, in accordance with best practices of software engineering for designing such services, is necessary. This paper focuses on a CAS design approach for building *CASOA*. To deal with such architectures development, challenges such as context management and dynamic service adaptation have to be faced. We propose in this article a design process that exploits both of our context and CAS specifications and metamodels in order to fulfil the passage from a core service in Service Oriented Architecture (SOA) to a CAS in *CASOA*. This passage is satisfied across a mechanism that, inspired by the Aspect Paradigm concepts, considers the service adaptations as aspects.

1 INTRODUCTION

Evolution in the fields of telecommunication (e.g., fast networking protocols), of mobile infrastructures (e.g., new generation of mobile devices) and software engineering in terms of architectures (i.e., emergence of new architectures like Service Oriented Architectures) and in terms of development paradigms (i.e., from the functional to the service while passing by the object and component paradigms) has promoted the birth of a new generation of software architectures known as Context-Aware Service Oriented Architectures (*CASOA*) which are articulated on a new design and development paradigm called Context-Aware Service (CAS). A 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 context situations to be considered make CAS hard to build and highlight the need of universally accepted basic design principles that can lead to a generic approach for efficient CAS development as an underlying mechanism for building *CASOA*. The traditional approaches for CAS development 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 whose rate of evolution and reuse is much 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 (Sheng, Yu and Dustdar, 2009). Though we base our remarks in this article on a specific application domain (i.e. The E-tourism), we follow a Model Driven Engineering (MDE) approach for *CASOA* artefacts development independently of the technical platforms and the application domains (Platform & Domain Independent Development Approach: *PDIDA*). Model-Driven Engineering (MDE) is a model centric approach for software development in which models are used to drive the development of all software artefacts. It provides great benefits in terms of cost reduction and quality improvement. Our

Hafiddi H., Nassar M., Baidouri H., El Asri B. and Kriouile A..
A CONTEXT-AWARE SERVICE CENTRIC APPROACH FOR SERVICE ORIENTED ARCHITECTURES.
DOI: 10.5220/0003473501760183
In *Proceedings of the 13th International Conference on Enterprise Information Systems* (ICEIS-2011), pages 176-183
ISBN: 978-989-8425-55-3

approach consists on providing *CASOA* artefacts metamodels which will serve for constructing models, then implementations can be generated automatically by performing a series of model to model transformations. Thereby, in addition of profits in terms of reuse, evolution, integration and maintenance, our approach can be easily transposed to various domains and target various technical platforms.

In the rest of this paper, we first present 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 and focus, in the fourth section, on giving a CAS specification and metamodel. Sect. 5 introduces how aspect paradigm can be applied to fulfil service adaptation to its execution contexts while in Sect. 6 we present our *CASOA* design process. Sect.7 briefly compares related work. Finally, we conclude the paper in Sect. 8 with plans for future work.

2 e-TOURISM SCENARIO

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 device (e.g., PDA, iPhone, BlackBerry, etc.) to a traditional E-tourism system in order to obtain a list of suitable restaurants. He subscribes to the system, launches his request (i.e. concerning a restaurants searching service) 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. the 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 response for tourist's expectations (i.e. inadequate display and inappropriate restaurants because the system doesn't take into account parameters like tourist's device type, his localization, his language, his preferences, etc.).

In purpose to use user's context and face its changes, this E-tourism system needs to be context-aware. 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 (device type is taken into account for display adaptation), close to his site (taken into consideration the localization), described in his language (the system will consider the user's language) and taking account his preferences (food preferences for instance). Also, let's note that such system will resort to a results pagination mechanism (considering the device capacities, the 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.

The development of this E-tourism system, in particular, and context-aware systems, in general, imply several challenges. First, Context definition (i.e. which context information are relevant for the adaptation of the system), structure (i.e. the properties and the connections between the information) and acquisition is not an easy process. Second, the adaptation process must be based on mechanisms in accordance with best practices (e.g., easy reuse and maintenance) of software engineering in order to produce well designed *CASOA*.

3 CONTEXT

Context is the information that characterizes the interactions between humans, applications, and the environment (Brezillon, 2003). Context information is dependent on system 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, (Chen and Kotz, 2000) and (Schmidt, Beigl and Gellersen, 1999) for example, 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" (Dey and Abowd, 1999, para. 2.2). As given in (Truong and Dustdar, 2009), 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 (Truong and Dustdar, 2010).

Rather than giving context formalization, case of

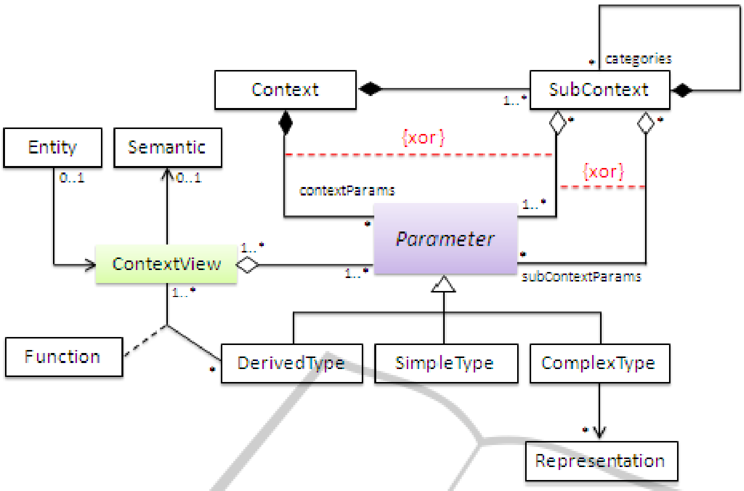

Figure 1: Core context metamodel.

figure for several researches on this topic, sometimes domain specific and sometimes generic but not very extensible, we choose to propose a metamodel which is, at the same time, generic and abstract (see Fig. 1). 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 many representations;
- A context view (i.e. a set of parameters) can have a semantic;
- An entity (e.g., service, user, device, etc.) is described by a set of parameters.

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 computing, 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.);
- EnvironmentSubContext: this sub context contains the environment parameters (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.).

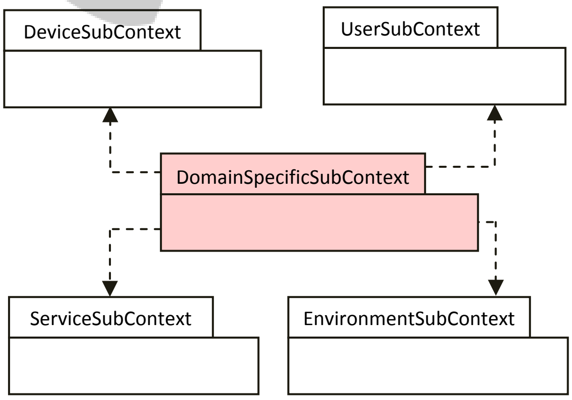

Figure 2: Ubiquitous context packages.

For organization and context management facility reasons, we structure our context model in packages as illustrated in Fig. 2. Some parameters (e.g., device type, 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 the application domain (E-tourism in our case), so they are placed in DomainSpecificSubContext.

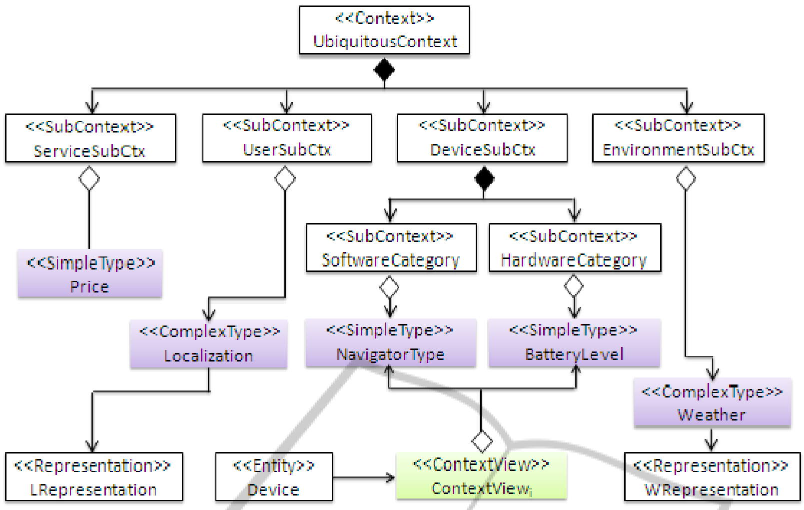

Figure 3: Succinct context model for the E-tourism scenario.

4 CONTEXT-AWARE SERVICE

One of the first uses of the term context-aware appeared in 1994 (Schilit and Theimer, 1994). A service is context-aware if it provides customized and personalized behaviour to users depending on their contexts (Dey and Abowd, 1999). In Service Oriented Computing (SOC), a service is defined as self-describing and platform-agnostic computational element that supports rapid, low-cost and easy composition of loosely coupled and distributed software applications (Papazoglou, 2003).

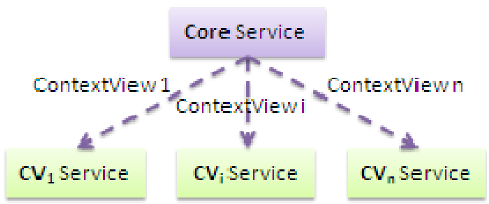

Figure 4: Core service adaptation to its various ContextViews.

To be context-aware, a service must be able to adapt dynamically its behaviour to its several execution context and adapt dynamically its behaviour. Henceforth, these appropriate context information relative to a specific execution situation form what we call the *ContextView* of the service and the result of service adaptation to this *ContextView* forms the *ContextViewService* (see Fig .4).

(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

Fig. 5 illustrates our CAS metamodel. This metamodel is based on the following specification:

- Both context-aware service and context-view service are specific services;
- A context-aware service (respectively context-view service) possesses a CAS adaptation strategy (respectively CVS adaptation strategy) which concerns a set of context views (respectively a given context view);
- A CAS adaptation strategy aggregates a set of CVS adaptation strategies;
- For a given CVS adaptation strategy and context view, a set of adaptation conditions is deduced;
- An adaptation condition can involve the execution of an ordered set of adaptations;
- For a given CVS adaptation strategy and adaptation, an adaptation rule is associated;
- A CVS adaptation strategy aggregates a set of adaptation conditions, ordered adaptations and adaptation rules.

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 conditions expressed on *ContextView* parameters) and how (i.e. *AdaptationRule*: defines the places in the service where the dynamic ordered 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*. So, for a given service, the set of its

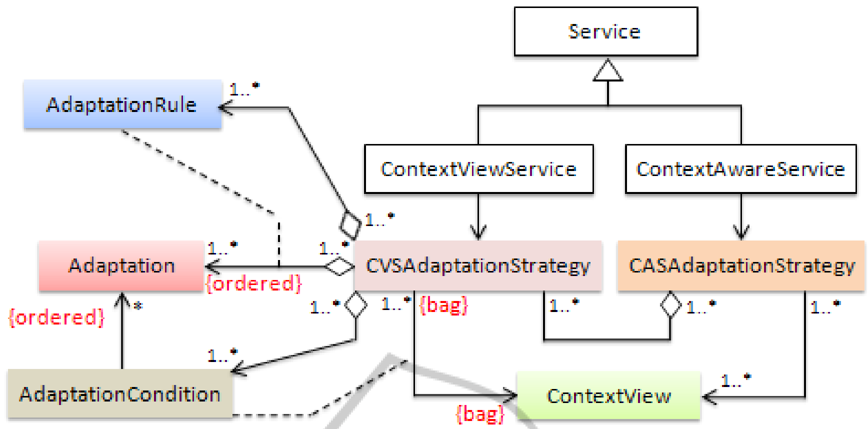

Figure 5: Core CAS metamodel.

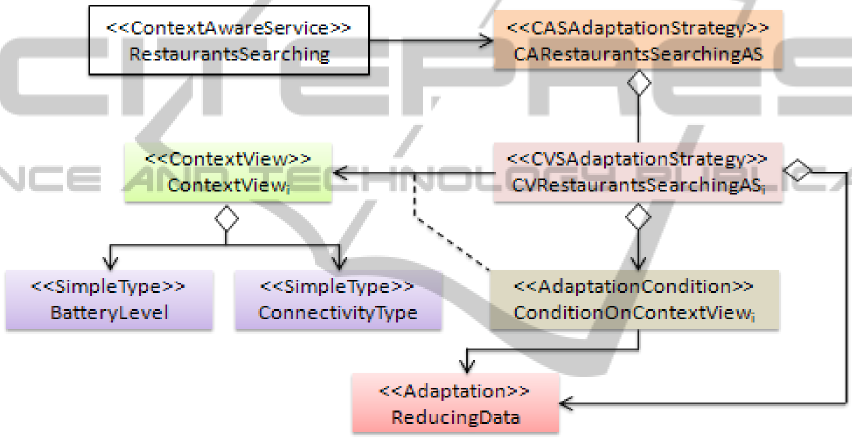

Figure 6: Succinct CAS model for the e-tourism scenario.

ContextViewServices (respectively *CVSAdaptationStrategies*) forms the CAS (respectively *CASAdaptationStrategy*).

For instance, in the E-tourism motivating scenario (c.f. Sect. 2), battery level and connectivity type represent one of the Restaurants Searching service *ContextViews* which can provoke service adaptation by reducing the amount of data returned (i.e. *Adaptation*) whenever this level is lower than 20% or the connectivity is changed from a high connectivity to a low one (i.e. *AdaptationCondition*). Fig. 6 presents a succinct CAS model in the case of Restaurants Searching service.

5 CONTEXT AWARE SERVICE ADAPTATION MECHANISM

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 and maintained. So, in order to rationalize the 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 favourite loosely coupling between the core service and its adaptations seen as crosscutting concerns.

Inspired by Separation of Concerns (Hürsch and Lopes, 1995) and Aspect Paradigm concepts (Kiczales, Lamping, Mendhekar, Maeda, Lopes, Loingtier and Irwin, 1997), our CAS design and development approach consists of 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

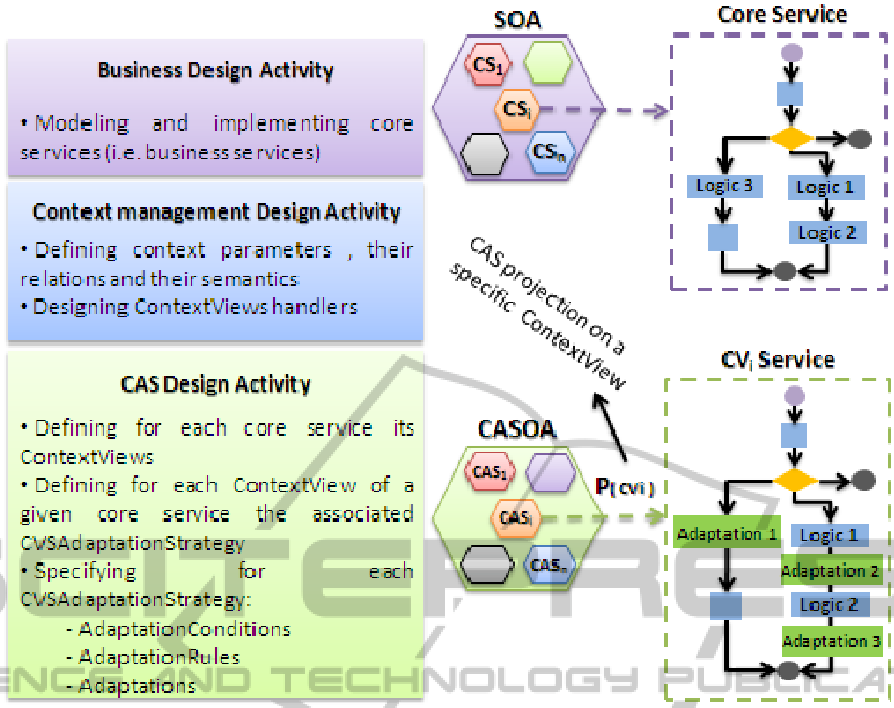

Figure 8: CAS design process.

weaved at runtime into core service by our tool named *Adaptation Aspects Weaver* (*A^2W*), in order to produce the expected *ContextViewService*.

The Fig. 7 illustrates the mechanism behind our *A^2W* tool. The *Request Notifier* notifies the *Decision Maker* with the executed service id and the execution context in order to recuperate the *CASAdaptationStrategy*. Then, the *Decision Maker* inspects it in order to retrieve and interpret only the *CVSAdaptationStrategy* corresponding to the pertinent current *ContextView*.

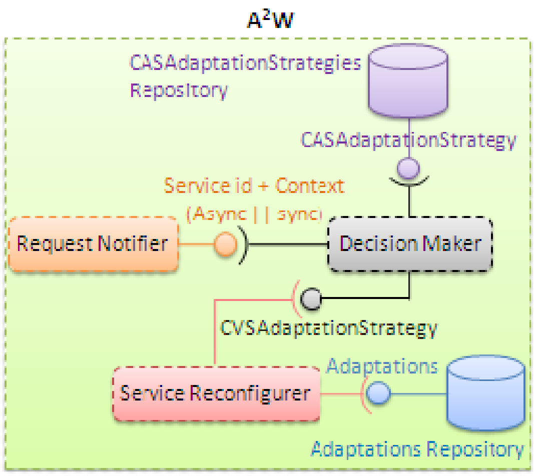

Figure 7: Adaptation Aspects Weaver architecture.

The interpretation mechanism, operated by the *Service Reconfigurator*,
consists in checking the *AdaptationConditions* in order to weave only the required *Adaptation Aspects*, following a set of *AdaptationRules*, into core service to produce the corresponding *ContextViewService*.~

Let's mention that our CAS development approach combined to the *A^2W* tool provide, in addition to dynamic service adaptation to the context, the ability to evolve service behaviour during the CAS life cycle.

6 CONTEXT AWARE SERVICE DESIGN PROCESS

The Fig. 8 illustrates our CAS design process to build *CASOA*. The whole process contains three main activities: the business design, the context management design and the CAS design.
The business design activity consists of specifying and implementing all core services that fulfil the system business requirements, resulting in an artifact of the design process: the system model. The two other activities deal with the context-awareness of the core services obtained in business design activity. Thereby, the context management design

activity consists on modelling context information that has an impact on the system and specifies the collection process (i.e. parameters handlers) while the CAS design activity aims at specifying the services variability according to its *ContextViews*.

7 RELATED WORK

Several context models have been defined (e.g., Key-value pairs (Schilit, Theimer and Welch, 1993), databases (e.g., CML (Henricksen and Indulska, 2006)), ontologies (e.g., CMF (Korpipää and Mäntyjärvi, 2003)), profiling (e.g., CC/PP (Klyne, Reynolds, Woodrow, Ohto, Hjelm, Butler and Tran, 2007), etc.) and various context-aware middleware and frameworks have been developed (e.g., context Toolkit (Salber, Dey and Abowd, 1999), CoBrA (Chen, 2004), K-Components (Dowling and Cahill, 2001), CORTEX (Sorensen, Wu, Sivaharan, Blair, Okanda, Friday and Duran-Limon, 2004), etc.) to deal with context- aware systems development. The main objective of context modelling researches is to provide an abstraction of context information to permit easy context management and they do not deal, in general, with application variability and adaptation to the context, whereas researches that focus on frameworks and middleware development try to simplify context-aware systems development by decoupling context management from adaptation logic but they suffer from a lack of well designed approach and introduce several technical details reducing systems portability.

Some other projects focus on context-awareness metamodeling. An important effort is the work conducted by the Taconet and Kazi-Aoul team in (Taconet and Kazi-Aoul, 2010). Authors define metamodels, following a MDE approach, for modelling context-aware applications by planning several model views that model system context sensitivity but they do not deal with adaptability. In our approach the system variability and adaptability to the context is realized through the notion of *CASAdaptationStrategy* and the A^2W tool. Ayed, Delanote and Berbers (2007) specify a MDD (Model Driven Development) approach and an UML profile to design context-aware applications independently of the platform. They propose a design process that models the contexts that impact an application and its variability but does not specify the mechanism to fulfil application adaptation to the context. In ContextUML project, Sheng and Benatallah (2005) define an approach for modelling context-aware Web Services. The approach is platform dependent and the context is specialized into AtomicContext and CompositeContext, so the semantic expressed in this metamodel is limited. Also, authors don't specify the mechanism used to fulfil CAS adaptation. Keidl and Kemper (2004) 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 (PLE) which has a great potential in modelling service variability. An important work is the one conducted in CAPPUCINE project (Parra, Blanc and Duchien, 2009). Authors focus on context-aware adaptation in Dynamic Service-Oriented Product Line (DSOPL) rather than context modelling and propose two different processes for the initial and iterative phases of product derivation. The main challenge to be faced in this work is to reduce non-deterministic behaviours when non-deterministic context-aware assets are introduced. In our work, this challenge is faced by the execution of an ordered set of adaptations.

8 CONCLUSIONS

In this article, we followed a MDE approach to realize *CASOA* artefacts independently of the technical platforms and the application domains (*PDIDA*). Thus, we presented, firstly, our context specification as a base for the context metamodel. 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 the A^2W tool at runtime. Finally, we proposed a CAS design process that allows designers to model the context that impacts the system and its variability to its execution contexts independently of system model.

We focused in this article on context and CAS specifications and metamodels and proposed an adaptation approach those lead to a *CASOA* design process. 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 pertinent *ContextView*s to A^2W. In the long term, our objective is to propose a framework allowing the CAS development. We target mainly the Web Services as a technical platform for implementing *CASOA*.

REFERENCES

Ayed, D., Delanote, D. and Berbers, Y. (2007). *MDD Approach for the Development of Context-Aware Applications*. In CONTEXT'07, the 6th International and Interdisciplinary Conference on Modeling and Using Context, Roskilde University, Denmark.

Brezillon, P. (2003). *Focusing on context in human-centered computing*. IEEE Intelligent Systems, 18(3), 62-66.

Chen, G. and Kotz, D. (2000). *A Survey of Context-Aware Mobile Computing Research*. Technical Report, Issue: TR2000-381, Dartmouth College.

Chen, H. (2004). *An Intelligent Broker Architecture for Pervasive Context-Aware Systems*. PhD thesis, University of Maryland, Baltimore County.

Dey, A., K. and Abowd, G., D. (1999). *Towards a Better Understanding of Context and Context-Awareness*. In Technical Report GIT-GVU-99-22, GVU Center, Georgia Institute of Technology.

Dowling, J. and Cahill, V. (2001). *The K-Component Architecture Meta-model for Self-Adaptive Software*. In REFLECTION'01, the 3rd International Conference on Metalevel Architectures and Separation of Crosscutting Concerns. Kyoto, Japan.

Henricksen, H. and Indulska, J. (2006). *Developing context-aware pervasive computing applications: Models and approach*. Pervasive and Mobile Computing, 2(1), 37–64.

Hürsch, W. and Lopes, C., V. (1995). *Separation of concerns*. In Technical Report NUCCS-95-03, Northeastern University. Boston, Massachusetts.

Keidl, M. and Kemper, A. (2004). *Towards Context-Aware Adaptable Web Services*. In WWW'04, the 13th International World Wide Web Conference. New York, USA.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., V., Loingtier, J., M. and Irwin, J. (1997). *Aspect-Oriented Programming*. In ECOOP'97, the 11th European Conference on Object-Oriented Programming, vol. 1241 of LNCS, Springer-Verlag.

Klyne, G., Reynolds, F., Woodrow, C., Ohto, H., Hjelm, J., Butler, M., H. and Tran, L. (2007). *Composite Capability/Preference Profile (CC/PP): Structure and vocabularies 2.0*. Technical report, W3C recommendation.

Korpipää, P. and Mäntyjärvi, J. (2003). *An ontology for Mobile Device Sensor-Based Context Awareness*. In CONTEXT'03, the 4th International and Interdisciplinary Conference on Modeling and Using Context. Stanford, USA.

Papazoglou, M., P. (2003). *Service Oriented Computing: Concepts, Characteristics and Directions*. In WISE'03, the 4th International Conference on Web Information Systems Engineering. IEEE Computer Society, 3-12.

Parra, C., Blanc, X. and Duchien, L. (2009). *Context Awareness for Dynamic Service-Oriented Product Lines*. In SPLC'09, the 13th International Software Product Line Conference. San Francisco, USA.

Salber, D., Dey, A., K. and Abowd, G., D. (1999). *The Context Toolkit: Aiding the Development of Context-Enabled Applications*. In CHI'99 Conference on Human Factors in Computing Systems. Pittsburgh, Pennsylvania, USA.

Schilit, B., N., Theimer, M., M. and Welch, B., B. (1993). *Customizing mobile applications*. In Proc. USENIX Symposium on Mobile and Location-Independent Computing. Cambridge, Massachusetts, USA.

Schilit, B. and Theimer, M. (1994). *Disseminating Active Map Information to Mobile Hosts*. IEEE Network, 8(5), 22–32.

Schmidt, A., Beigl, M. and Gellersen, H., W. (1999). *There is more to context than location*. Computers and Graphics Journal, 23(6), 893–902.

Sheng, Q., Z. and Benatallah, B. (2005). *ContextUML: A UML-based modelling language for model-driven development of context-aware web services*. In ICMB'05, the 4th International Conference on Mobile Business. Sydney, Australia.

Sheng, Q., Z., Yu, J. and Dustdar, S. (2009). *Enabling Context-Aware Web Services: Methods, Architectures and Technologies (*Ed. 2009*)*. London: Chapman and Hall/CRC.

Sorensen, C., F., Wu, M., Sivaharan, T., Blair, G., S., Okanda, P., Friday, A. and Duran-Limon, H. (2004). *Context-aware Middleware for Applications in Mobile Ad Hoc Environments*. In the 2nd workshop on Middleware for pervasive and ad-hoc computing. Toronto, Canada.

Taconet, C. and Kazi-Aoul, Z. (2010). *Building context-awareness models for mobile applications*. Journal of Digital Information Management, 8(2), 78-87.

Truong, H., L. and Dustdar, S. (2009). *A survey on context-aware web service systems*. International Journal of Web Information Systems, 5(1), 5–31.

Truong, H., L. and Dustdar, S. (2010). *Context Coupling Techniques for Context-aware Web Service Systems: An Overview*. In Sheng, Q., Z., Yu, J. and Dustdar, S. (Eds.), *Enabling Context-Aware Web Services: Methods, Architectures and Technologies (*pp. 337-364*)*. London: Chapman and Hall/CRC.