Journal of Computer and Communications, 2017, 5, 44-59
http://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Intermediate Common Model—The Solution to Separate Concerns and Responsiveness in Dynamic Context-Aware System

Gaëtan Rey, The Can Do, Jean-Yves Tigli, Stéphane Lavirotte, Nhan Le Thanh

Université Côte d'Azur, Nice, France
Email: Gaetan.REY@unice.fr, the-can.do@univ-cotedazur.fr, Jean-Yves.TIGLI@unice.fr, Stephane.LAVIROTTE@unice.fr, Nhan.LE-THANH@unice.fr

How to cite this paper: Rey, G., Do, T.C., Tigli, T.-Y., Lavirotte, S. and Thanh, N.L. (2017) Intermediate Common Model—The Solution to Separate Concerns and Responsiveness in Dynamic Context-Aware System. *Journal of Computer and Communications*, 5, 44-59.
https://doi.org/10.4236/jcc.2017.54004

Received: January 26, 2017
Accepted: March 20, 2017
Published: March 23, 2017

Copyright © 2017 by authors and Scientific Research Publishing Inc.
This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).
http://creativecommons.org/licenses/by/4.0/
Open Access

Abstract

Nowadays, many works are interested in adapting to the context without taking into account neither the responsiveness to adapt their solution, nor the ability of designers to model all the relevant concerns. Our paper provides a new architecture for context management that tries to solve both problems. This approach is also based on the analysis and synthesis of context-aware frameworks proposed in literature. Our solution is focus on a separation of contextual concerns at the design phase and preserves it as much as possible at runtime. For this, we introduce the notion of independent views that allow designers to focus on their domain of expertise. At runtime, the architecture is splitted in 2 independent levels of adaptation. The highest is in charge of current context identification and manages each view independently. The lowest handles the adaptation of the application according to the rules granted by the previous level.

Keywords

Middleware, Ubiquitous Computing, Context-Aware System, Software Composition, Software Architecture

1. Introduction

Today, it is essential that software applications are able to adapt to their environment. The vision of Weiser [1] on Ubiquitous computing and works of Dey *et al.* [2] [3] about context opened the way for a multitude of solutions to this problem. The majority of these solutions (named Auto adaptive system in Figure 1) operate according to the cycle "observation, decision, action" as shown in Figure 1, and take their decision based on the set of rules (or equivalent) which

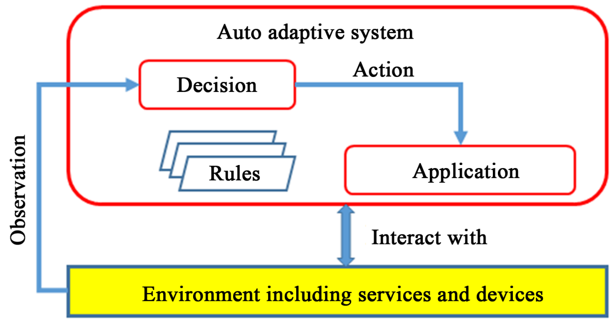

Figure 1. Classical model of Auto adaptive system.

define the adaptation capacities (also call adaptation range) of the application (or provide by the Auto adaptive system). But problems remain with those solutions and in particular if we want to improve the adaptation capacities to take into account new situations or contexts. Indeed, the interest of having an automatic adaptation of an application to the context is to allow the continuity in the services of the user, and that, whatever the context. If we add this to the increasing mobility of users, it is necessary to adapt the applications to all situations of life, whether professional, personal or others.

With this system, if we want to increase the adaptation capacities which are necessary in order to take account of the continuity of user services, we must use more Adaptation Rules to drive the decision process. But this results in:

- System takes more time to adapt properly.
- System consumption increases (memory, power …).
- Risk of conflict between Adaptation Rules is also increasing.

Many works have been dedicated to advance the adaptation domain [4]; frameworks [2] [5] [6], methods and techniques [3] [7] [8] [9], principles [10] [11] have already been proposed. However, most of these information focus on solving problems of system at runtime and they do not provide any solution to separate concerns and responsiveness in dynamic context-aware system. Some others suppose to use meta-model [7] [11], in design time that consistently defines related concepts, properties and relationships, establishing a common ground for implementing adaptation. This approach provides a good solution to manage context elements but it makes each expert’s task become complicated.

In this paper, we propose an approach called Intermediate Common Model with main purpose that is improving the adaptation capacities of system. Intermediate Common Model uses independent views that will increase the reuse of views between different applications. In addition, we propose to use context-aware management to manage views; it is the solution to separate concerns and responsiveness in dynamic context-aware system.

This paper is organized as follows: Section 2 presents and discuss related work, motivates this research; Section 3 details hypotheses and architecture of system; Section 4 describes Intermediate Common Model; Section 5 applies the results and implementation; Section 6 presents limitation of approach and Section 7 concludes this work and future work.

2. Related Works and Motivation

We can find many work has been done in the area of context-aware application in the past few year. In this section we present the results or proposals of works that focus on building context-aware application based on models to improve the adaptation capacities of system.

The work has been done by Dey *et al.* [2] which develop a context toolkit based on composed of sensor to collect information from context. This architecture supports a framework and reusability of components, evolution of applications and the acquisition and use of complex context. The context toolkit provides methods to access to such information, transform the context information into high-level formats that are easier to handle, allow the separation of acquisition process and context representation of the adaptation. However, these systems do not provide solution to use specific views and context management, so it is become complicated to use in large system with the wide range of context.

Henricksen [10] focuses on modeling at the conceptual level using Context Modeling Language, a novel modelling approach which offers a graphical notation and an accompanying modelling methodology for describing context information. A modeling process involving the following steps is assumed: construction of a conceptual model of context requirements using Context Modeling Language; mapping of the conceptual model to the relational model; and generation of a context management infrastructure based on the relational model. This approach use domain independent support on developing context-aware system but it is slightly hindered by the absence of a context modeling editor.

Person is AD [5], is a context-aware framework which builds upon the homogenous modeling of all the entities relevant to supporting ubiquitous applications. In this approach, the model is organized as a tree of model context which contain the components of the model. This framework makes it possible to quickly create new context-aware applications. The Person is AD architecture is based upon a small set of operations for applications to interact with the models: access, ask and tell. The service discovery facilitates distributing the models across various machines across a network. The application writer can simply use these, in conjunction with the active models, to build applications. This approach offers reusing active model in different applications but it does not consider information about the context-aware system behavior. Moreover, they do not provide an solution to replace or change model when the application change during adaptation process.

Costa [12] proposes an integrated solution for development of context-aware system. In this approach, context-aware application behaviors can be described as logic rules, which are called Event-Condition-Action rules that are consistent with the Event-Control-Action pattern. The context processor component gathers context information from the user's environment performs context reasoning and generates context and situation events. The controller Component observes events from context processors, monitors conditions rules, and triggers actions on action performer when the condition is satisfied. This approach provides a solution that facilitates the dynamic configuration and execution of particular application behaviors which based on a rule engine that gathers context and situation values from context processing components. However, this approach was not support solution to separate the management context and adaptation process and no concrete implementation was suggested.

Achilleas, Kun and Nektarios [6] propose a model-driven approach that provides a higher level of automation in software generation. The approach is strictly based on the Model Driven Architecture paradigm and provides the capability to semi-automatically generate service creation environments for different application domains. Hence, the approach and the generic framework are utilized to define and generate the Context Modeling Framework in the form of an Eclipse plug in. The plug in is then integrated into the generic framework, comprising a new software capability. Consequently, merely the modelling, validation and implementation tasks must be carried out for the creation of pervasive services. In addition the capability to generate diverse implementations and deploy pervasive services on different devices is provided. This simplifies the process and enables the rapid creation of pervasive services at the static compile time. Although this approach supports the ATL editor to transfer context to context, they do not provide solution to use independent models which can increase reusability of views.

AOCI [11] proposes programming support for context-aware adaptations that is built upon a semantic model. Their solution is integrated within our Aspect-Oriented Component Infrastructure (AOCI) framework that so far was limited to handle basic annotations using a semantic layer to make AOCI enhanced applications adaptable. They extend this basic support by explicit modelling of the context as well as application-specific domains inside this layer using ontologies of different granularity of abstraction. This mechanism supports context management ability and adapting of application, but they do not propose any implementation or code for development.

TriPlet [4] is structured in three core components: a meta-model is called Context-Aware Meta Model that formalizes and abstracts the main concepts for implementing Context-aware adaptation; a reference frameworkis called Context-aware Reference Framework that provides stakeholder support to define, specify and to decide the design for implementing Context-aware adaptation. It can be used before the implementation phase of an application, as an extensive catalogue to guide developers in taking design decisions, or after the implementation phase of an application, to analyze and to evaluate the concepts that were considered, aiding also to identify underexplored areas for future extensions.; and a design space, Context-aware Design Space that supports stakeholders in analyzing, comparing and evaluating the coverage levels of adaptation for context-aware applications. The major drawback of this approach is that it offers only methodology for building applications. They have offered neither implementation nor code nor development or configuration mechanisms to help programmers to develop applications.

CAISDA [7] provides architecture of context-aware framework which is formed of two parties. It describes the development process of a context-aware application by the developer in design time for it is consistent with us framework in run time. In the other hand, it describes the architecture of this framework in run time. In design time, the developer must build a context model of its application to be consistent with meta-model and follow a set of steps guided by mechanisms and tools to generate at the end a context-aware application integrated in the CAISDA framework. In runtime, when the user launches the application, it must be able to capture and to interpret the context elements possible to change during the application execution. This approach is good with applications which use single view but with applications use many different views, the analyze context will become complicate and take more time.

Requirements of Context-aware system are identified depend on each part of system such as Context model, architecture of system, development and deployment of applications.

Firstly, with Context model, some requirements are inspired from works in the previous section such as *Domain independent* [7], *Rich and Dynamics context* [2], *Behavioral and Structural* [12]. Another important requirement of Context model is "*Change and Reuse of views*" that is use to improve adaptation ability, simplify expert's work and develop application. The information in **Table 1** shows that no any approach suppose all necessary requirements of Context Modeling.Some works use meta-model to represent the static and dynamic aspect of the context. However, in case of applications that using many specific views, the combination of views in meta-model is very complex and the work of each developer is not simple. Moreover, if we want to add, delete or replace views that can't be done either by automatic process.It is motivation to our propose use Intermediate Common Model which can adapt requirement about reusable views and improve adaptation capacity of system.

Secondly, the requirements with architecture of system are defined in works in the previous section such as *Adapting of application* [7], *Context Interpreting and Storage* [2], *Separate Context Aware Manager and adaptation*, *Separation of Concerns* [12]. During adaptation process context can be change and system need *Update current context* that relate to correctly of adaptations and reduce the decision time of system with application. *Responsiveness* is ability of system

Table 1. Comparison Context-model of current context-aware frameworks.

Context framework / Requirements with context model	Context toolkit	Henricksen	Personis AD	Costa	Achilleas *et al.*	AOCI	TriPlet	CAISDA
Domain independent	+	+	+	+	+	+	+	+
Rich and dynamics	+	+	+	+	+	-	+	+
Behavioral and structural	−	−	−	+	+	+	+	+
Change and reuse of views	−	−	+	−	−	−	−	−

to adjust quickly to suddenly alter external conditions, as of the changing of current context applications, add or delete views without undue delay. Most of worked propose solution to update current context, but the combination of management, analysing and selecting context elements at run time can make conflict between different applications.

Table 2 summarizes all supporting of literature mentioned with requirement of architecture system. Based on the analysis information provided in Table 2, we can conclude that no single approach has the features to address all the issues that were identified. We propose an approach that is usingAuto Adaptive Systemto take care current context without care to manage context. Moreover, it also support the separation of concerns and responsivemess which necessary to improve adaptation capacity.

Thirdly, with requirement about *Rapid development and deployment of application* [7], our approach use independent common model at design time combination with two independent execution cycles of the Context Aware Manager and the Auto Adaptive System at run time which can address all issues with development and deployment of application.

3. Hypotheses and Architecture of System

When we want to improve the adaptation capacity by use more Adaptation rules (Adaptation Rules) to drive the decision process such mentioned in introduction section. We must solve problem conflict between Adaptation Rules, but solving that problems at design time are even more sensitive to the increased context that we would like to cover and thus increasing the number of rules. More rules, more potential conflicts between them are importance. We must develop automatic methods of detection and conflict resolution. But this is much more difficult to do at runtime.

In addition, these methods take, due to their complexity, considerable time. This has the effect of greatly reducing the responsiveness of the process of adaptation to the context. Or if this time is too long, the context may change. Then, we are in a situation where the performed adjustment does not make sense and it is necessary to calculate a new one. This can result in significant disruption of

Table 2. Comparison architecture of current context-aware frameworks.

Context framework / Requirements with context model	Context toolkit	Henricksen	Personis AD	Costa	Achilleas *et al.*	AOCI	TriPlet	CAISDA
Adapting of application	−	+	+	+	+	+	+	+
Context interpreting and Storage	+	+	−	+	−	−	+	+
Separate CAM & adaptation	−	−	−	+	+	−	−	+
Update current context	+	−	+	+	+	+	−	+
Separation of concerns	+	−	−	−	−	+	+	−
Responsiveness	−	+	+	−	+	−	−	+

operation of the application.

Building on this, we propose here a hybrid approach that will increase the number of Adaptation Rules and thus improve the consideration of the context while ensuring a reasonable time to adapt to the application.

In firstly, we assume that current Auto adaptive systems (Auto Adaptive System) are enough good to manage limited number of situations (defined by a short list of Adaptation Rules). So we make the choice to only manage this Auto Adaptive System by controlling their list of Adaptation Rules, and by relying on them for the adaptation functions.

We get architecture (as shown in **Figure 2**) where a new Context Aware Manager has been introduced to manage the Auto Adaptive System. In addition to managing the list of Adaptation Rules to be deployed on the Auto Adaptive System, this architecture preserves the independence of the execution cycle of the Auto Adaptive System. The execution cycles of the Context Aware Manager and the Auto Adaptive System are independent. This allows the Auto Adaptive System to take care of changes in the current context. As the Context Aware Manager handles, depending on its own rhythm, context switching requires Auto Adaptive System reconfiguration.

While this is not the object of our study, it is also desirable that the Auto Adaptive System proceeds in the same way, *i.e.* the life cycle and the operation of the application are independent of the adaptation process. Thus, the application can continue to operate during the decision stage (calculating adaptation functions) and is interrupted only during the implementation of the adaptation plan.

Secondly, we assume that address challenges of context awareness means managing various views. By views, we mean different concerns taken into account in the design of an application as business processes, privacy, human tasks, security, etc.

Few studies take into account different views and most focus on the contextualization of one view (e.g. by extending the business process [13]). Or, the views are not managed independently and are mixed in a single model: the model of context [7].

But like we say in introduction, it is very difficult to produce at design time a unique model including all views. Therefore, we believe that each expert should be able to work on his domain of expertise independently of others experts. For

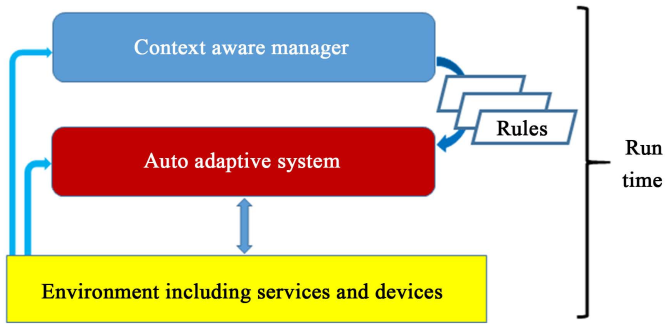

Figure 2. The runtime architecture model.

example, a domain expert can produce a BPMN [14], an ergonomist produces scenarios [15], whereas a security expert can focus on others specific models [12]. And we must be able to dynamically combine these views in them during the application runtime.

Figure 3 presents the design part of our architecture. It preserves the independence of the different views during the design.

In view of the diversity and unpredictability of developments in ubiquitous environment, to solve conflict between Adaptation Rules at design time does not reasonable today. Because it would mean that designer should able to evaluate all the possible combinations (between Adaptation Rules) that could meet a user. Indeed, if we analyze the company's business processes, we quickly see that they only cover the nominal case. Unexpected situations, such as their qualifier suggests, are not taken into account.

The reason for this is that the analysis and context modeling is currently global way. Designers define a single large model of context, which still has the advantage of resolving conflicts as possible during the design phase. But making the task long, tedious and even impossible if we increase significantly the context we wish to support.

Instead, each expert can specify its own view, without having to know the views specified by the other experts. In fact, the work of each expert is simpler and can therefore be described in more detail without increasing the working time.

In addition to ease of modeling, we believe that the separation into independent views will increase the reuse of views between different applications.

We associate a set of views based on adaptive capacities that we want to provide to our application. This set may change over time (add, delete or replace views). This can be done either by automatic process or by the end user and without the need to bring in an expert.

4. Intermediate Common Model

We currently introduce an intermediate common model (Intermediate Common

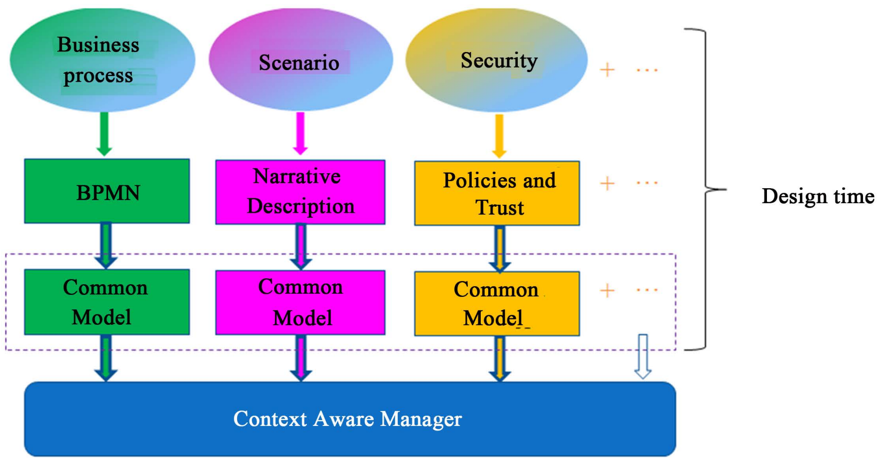

Figure 3. The design architecture model.

Model) on **Figure 3**, between expert specific models and the Context Aware Manager. The goal of Intermediate Common Model is to simplify the management of models by the Context Aware Manager.

4.1. Intermediate Common Model Description

In design time, the developer must transform each special view into XML description to provide data for Context Aware Manager. **Figure 4** shows overviews of Intermediate Common Model approach in design time.

The input of each Intermediate Common Model is a special view such as BPMN, Scenario, etc. The Intermediate Common Model is an XML description of a Moore automaton described by the following 6-tuple (S, S0, Σ, Λ, δ, λ):

- S is a finite set of states.
- S0, with S0 ∈ S, is a start state.
- Σ is an input alphabet defined as a finite set of predicates used to identify the states in S. N predicates leads a maximum of 2N possible states.
- Λ:Λ → (Ω, ϕ) is an output alphabet defined as a pair of finite Adaptation Rules sets. The first set Ω contains observation rules used by Context Aware Manager for its observation *i.e.* the evaluation of Σ predicates. The second set ϕ contains the application rules that will be deployed on Auto Adaptive System as shown in **Figure 1**.
- δ:S x Σ → S' is a transition function mapping a set of state S and the input alphabet Σ to the next set of state S'.
- λ:S → Λ is an output function mapping each set of state S to the output alphabet Λ.

Relying on XML extensibility property, the Common Model does not require the type of adaptation rules that it contains the description. Similarly, it allows you to specify different way predicates and therefore how the Context Aware Manager will perform observation.

The goal was to build a model flexible enough to support our various experiment

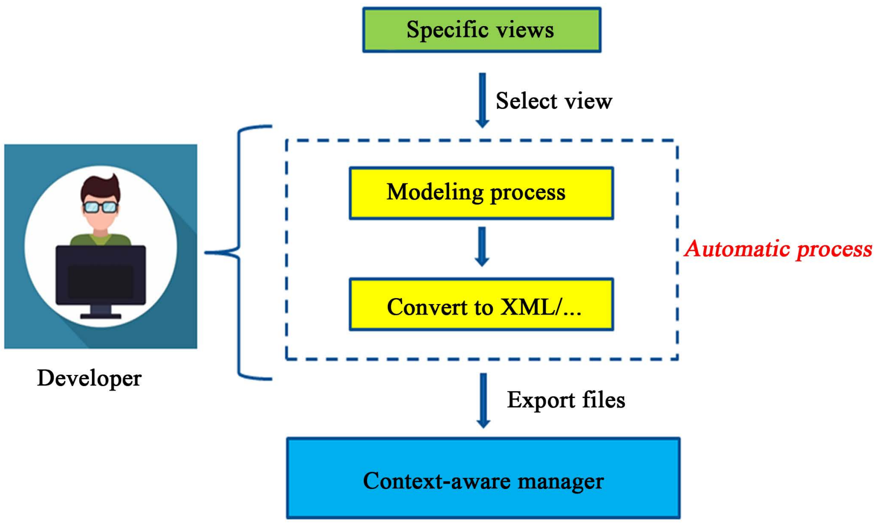

Figure 4. Overview of design time.

and research tracks and to adapt to the different implementation of Context-Aware Manager and Auto Adaptive System.

-Figure 5 shows graphic of XML schema that uses to define Intermediate Common Model.

4.2. Example

We use two different views *Security* and *Process* to analyze information from the worker and then active status of Tool and display Worker's task. With P is authenticate from worker and S is a finite set of states, we have:

-In view: Security.

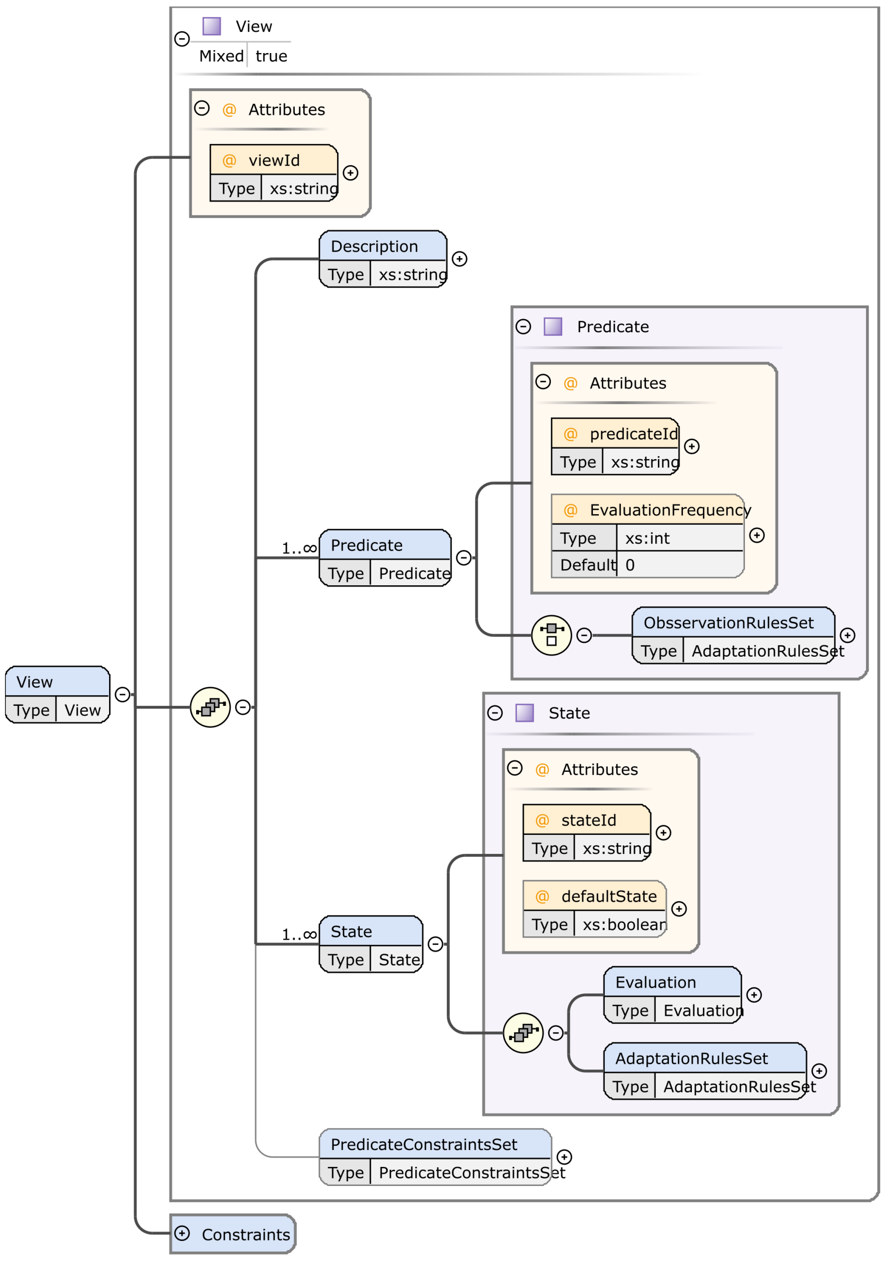

Figure 5. Schema of XML program.

S1: no authenticate! P1→disable Tool.

S2: have authenticate P1→activate Tool.

-In view: Process.

S1: task not selected→!P1→disappear Task.

S2: task selected + worker absence→P1 & !P2→activate Tool and information.

S3: task selected + worker at place→P1 & P2→activate Tool and Task.

-XML description of "Security" view:

```
<?xmlversion="1.0"encoding="UTF-8" ?>
<Viewxmlns="gdc"xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="gdc ViewSchema.xsd"viewId="Security">
<Description>Example of a simple security view</Description>
<PredicatePredicateId="P_authenticate">
<ObservationRulesSet> ...  </ObservationRulesSet>
</Predicate>
<StatestateId="S_not_authenticate"defaultState="true">
<Evaluation>
<PredicatePredicateId="P_authenticate"value="false"/>
</Evaluation>
<AdaptationRulesSet> ... </AdaptationRulesSet>
</State>
<StatestateId="S_authenticate">
<Evaluation>
<PredicatePredicateId="P_authenticate"value="true"/>
</Evaluation>
<AdaptationRulesSet> ... </AdaptationRulesSet>
</State>
</View>
```

-XML description of "Process" view:

```
<?xmlversion="1.0"encoding="UTF-8" ?>
<Viewxmlns="gdc"xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="gdc ViewSchema.xsd"viewId="BusinessProcess">
<Description>Example of a simple Business Process view</Description>
<PredicatePredicateId="P_taskSelected">
<ObservationRulesSet> ...  </ObservationRulesSet>
</Predicate>
<PredicatePredicateId="P_workerOnSite">
<ObservationRulesSet> ...  </ObservationRulesSet>
</Predicate>
<StatestateId="S_task_not_selected"defaultState="true">
<Evaluation>
<PredicatePredicateId="P_taskSelected"value="false"/>
</Evaluation>
<AdaptationRulesSet> ... </AdaptationRulesSet>
</State>
<StatestateId="S_worker_not_on_site">
<Evaluation>
<PredicatePredicateId="P_taskSelected"value="true"/>
<PredicatePredicateId="P_workerOnSite"value="false"/>
</Evaluation>
<AdaptationRulesSet> ... </AdaptationRulesSet>
</State>
<StatestateId="S_worker_on_site">
<Evaluation>
<PredicatePredicateId="P_taskSelected"value="true"/>
<PredicatePredicateId="P_workerOnSite"value="true"/>
</Evaluation>
<AdaptationRulesSet> ... </AdaptationRulesSet>
</State>
</View>
```

4.3. Context Definition

In summary, we distinguish the design time from the runtime.

-At design, designers independently write views that correspond to the contextual concerns of each of these designers. Each view decomposes into a set of states defined by a set of predicates specific to each view. In each state, we also associate a list of adaption rules to adapt the application to the current state?

-At runtime, the evaluation of the predicates of each of the views makes it possible to identify the current situation (*i.e.* the set of current states of each view). Depending on this situation, a list of adaptation rules will be applied.

It can then be said that the context is the set of possible situations for a given set of views.

Our context definition therefore does not approach this notion according to the type of entities or attributes observed but rather as a set of photographs of the elements components world where the applications designers have the freedom to observe what interests them depending on their concerns.

5. Application and Current Implementations

Our implemented solutions are based on an auto adaptive system, named W-Comp. This Auto Adaptive System supports the adaptation of application during runtime depending on the presence or absence of devices [8]. A performance model is available in [16], allowing to estimate the responsiveness of the adaptation based on the number of rules deployed by the Context Aware Manager.

The current Context Aware Manager implementations support an independent way of management views. Each view is described according to Context Model. Context Aware Manager is able to set up observations mechanisms to identify the current situation (via predicates described in the Context Model). Once the situation of each view, identified, it deploys all the relevant Adaptation Rules on the Auto Adaptive System. The Auto Adaptive System is in charge of enforcing the rules when it can and managing conflicts that could occur [17].

5.1. First Solution

We have focused on our currents explorations on observation part. Our first Context Aware Manager implementation (**Figure 6**), used in the project ANR CONTINUUM [18], is based on a knowledge base for evaluating predicates. The predicates are then modeled inside the Context Model as SP Adaptation Rules QL queries whose answer is true or false. The knowledge base is in charge of observing the environment, to infer new knowledge and responding to requests from Context Aware Manager.

The problem with this solution is the need to lock the knowledge base during the evaluation of predicates so as to have answers based on the same knowledge.

However, during the lock, the base does not update its knowledge (that is the purpose of the lock). But this causes problems if you want to share the base between several Context Aware Managers, the base may be locked continuously.

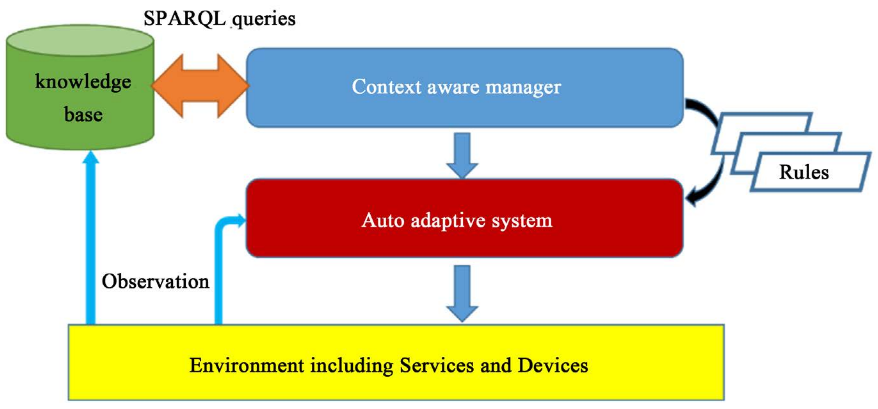

Figure 6. Our first solution of Context aware manager.

5.2. Current Solution

After several experiments on modeling of predicates, we have taken up the idea of contextual observation chains that introduced by Rey [8]. Each predicate is an application modeled by way of an assembly of components. Context Aware Manager (Figure 7), to set up his observation will deploy an application for each predicate that it wants to evaluate. During this deployment, the Context Aware Manager set how it wants to receive the evaluation of the predicate (requests, notifications ...). To do this we rely on our WComp framework.

Two solutions are then possible:

-Either a single container is used for all applications, simplifying management for Context Aware Manager and saves resources.

-Or each application has its own container, simplifying the sharing of applications (thus predicates) between different Context Aware Manager.

The advantages of this solution are many.

-First, there is no central point of failure as was the case with the knowledge base.

-Second, the sharing of observation mechanism in several Context Aware Managers is facilitated. Because each predicate is managed independently, the lock is not necessary. But the Context Aware Manager needs to manage data synchronization.

-Finally, as we used our WComp framework for the creation of applications, we can make them benefit from its capacity for self-adaptation. Each application is able to adapt to variations of the devices in the environment. This improves observation by making it more robust.

6. Limitation

The current work is based on few assumptions that could be seen as limitations. We estimate that all the thinking models (views) used at design time can be projected in our context model. To validate this, we must experiment with other views and evolve our intermediate common model according to these experiments. For now, we only carried out some tests and experimentally validated the

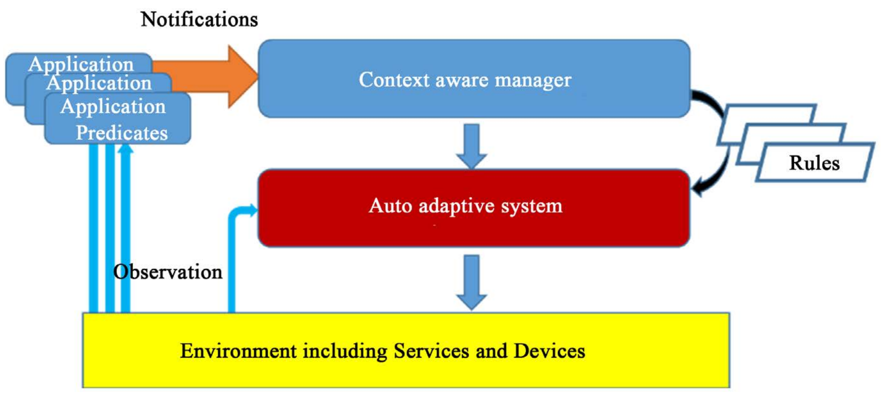

Figure 7. Our current solution of Context aware manager.

projection, on our model, of the BPMN (U-INSITHER project with EDF R & D) and the narrative description [18] (CONTINUUM project with SUEZ Environment and Lyonnaise des Eaux).

On the other hand, the current approach strongly links the state of a view (the current situation) with the rules to be deployed (adaptation rules). Our first experiments (carried out in the projects mentioned above) showed that the designers of the views are not able to write these rules. That is why we are interested in the intentional approach. The aim would then be to associate intent with a state of a view and to be able to rediscover the adaptation rules to be implemented according to the set of intentions identified as required.

7. Conclusions and Future Work

This article presents a new architecture for context management using independent view models to design specification and using 2 independent levels of adaptation at runtime.

Preserving the independence views at design time, this approach simplifies the work of designers and increases the reusability of views between different applications.

At runtime, the use of two independent levels, this allows Auto Adaptive System to focus on the adaptation of the application without sacrificing responsiveness, whereas, at the same time, Context Aware Manager focuses on identifying the current context to change the behavior of Auto Adaptive System.

In the future, we will rely on our experimental results to develop our model. These changes aim to ease the transition from specific expert models and to detect and to resolve possible conflicts between views or Adaptation Rules by the Context Aware Manager.

Acknowledgements

Thank you to all partners of the Continuum Project [18] which collaborated on the design and implementation of the first solution. Also thanks for EDF R & D, which has supported us through the U-Insither Project (collaborative project between the Université Nice Sophia Antipolis and EDF R & D/STREP).

References

[1] Weisser, M. (1991) The Computer for the Twenty-First Century. *Scientific American*, **265**, 94-104. https://doi.org/10.1038/scientificamerican0991-94

[2] Dey, A.K., Abowd, G.D. and Salber, D. (2001) A Conceptual Framework and Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction*, **16**, 97-166.

[3] Dockhorn Costa, P. (2007) Architectural Support for Context-Aware Applications—From Context Models to Service Platforms. CTIT PhD Thesis, 021(TI/FRS/021), The Netherlands University, Enschede.

[4] Motti, V.G. and Vanderdonckt, J. (2013) A Computational Framework for Context-Aware Adaptation of User Interfaces. *Proceedings of the 7th International Conference on Research Challenges in Information Science,* Paris, 29-31 May 2013, 1-12. https://doi.org/10.1109/rcis.2013.6577709

[5] Assad, M., Carmichael, D.J., Kay, J. and Kummerfeld, B. (2007) PersonicAD: Distributed, Active, Scrutable Model framework for Context-Aware Services. In: LaMarca, A., Langheinrich, M. and Truong, K.N., Eds., *Pervasive Computing. Pervasive* 2007. *Lecture Notes in Computer Science*, *Vol.* 4480, Springer, Berlin, Heidelberg, 55-72.

[6] Achilleos, A., Yang, K. and Georgalas, N. (2010) Context Modelling and a Context-Aware Framework for Pervasive Service Creation: A Model-Driven Approach. *Pervasive and Mobile Computing*, **6**, 281-296.

[7] Jaouadi, I., Raoudha, B.D. and Hanene, B.A. (2015) Approach to Model-Based Development of Context-Aware Application. *Journal of Computer and Communications*, **3**, 212-219. https://doi.org/10.4236/jcc.2015.35027

[8] Rey, G. and Coutaz, J. (2004) The Contextor Infrastructure for Context-Aware Computing. *Workshop on* "*Component-oriented Approaches to Context-Aware Computing held*" *ECOOP*'04, Oslo, 14 June 2004.

[9] Lavirotte, S., Rey, G., Rocher, G. and Tigli, J.-Y. (2015) A Generic Service Oriented Software Platform to Design Ambient Intelligent Systems. *UbiComp*/*ISWC*'15 *Adjunct*, Osaka, 7-11 September 2015, 281-284. https://doi.org/10.1145/2800835.2800843

[10] Henricksen, K. and Indulska, J. (2006) Developing Context-Aware Pervasive Computing Applications: Models and Approach. *Pervasive and Mobile Computing*, **2**, 37-46.

[11] Soldner, G., Kapitza, R. and Meier, R. (2011) Providing Context-Aware Adaptation Based on a Semantic Model. In: Felber, P. and Rouvoy, R., Eds., *Distributed Applications and Interoperable Systems*. *DAIS* 2011. *Lecture Notes in Computer Science*, Vol. 6723, Springer, Berlin, Heidelberg, 57-70. https://doi.org/10.1007/978-3-642-21387-8_5

[12] Joshi, A., Finin, T., Kagal, L., Parker, J. and Patwardhan, A. (2008) Security Policies and Trust in Ubiquitous Computing. *Philosophical Transactions of the Royal Society A*, **366**, 3769-3780. https://doi.org/10.1098/rsta.2008.0142

[13] Yousfi, A., de Freitas, A., Dey, A. and Saidi, R. (2015) The Use of Ubiquitous Computing for Business Process Improvement. *IEEE Transactions on Services Computing*, **9**, 621-632. https://doi.org/10.1109/TSC.2015.2406694

[14] Dumas, M., La Rosa, M., Mendling, J. and Reijers, H. (2013) Fundamentals of Business Process Management. Springer-Verlag, Berlin.

[15] Carroll, J.M. (2000) Five Reasons for Scenario-Based Design. *Interacting with Computers*, **13**, 43-60. https://doi.org/10.1016/S0953-5438(00)00023-0

[16] Tigli, J.Y., Lavirotte, S., Rey, G., Ferry, N., Hourdin, V., Fathallah, S., Vergoni, C. and Riveill, M. (2012) Aspects of Assembly: From Theory to Performance. In: Leavens, G.T., Chiba, S., Haupt, M., Ostermann, K. and Wohlstadter, E., Eds., *Transactions on Aspect-Oriented Software Development IX*, Springer, Berlin, Heidelberg, 53-91. https://doi.org/10.1007/978-3-642-35551-6_2

[17] Abdenneji, S., Lavirotte, S., Tigli, J.Y., Rey, G. and Riveill, M. (2012) The Dynamic Composition of Independent Adaptations Including Interferences Management. *Proceedings of the 7th International Conference on Software Engineering Advances* (*ICSEA*), Lisbon, 18-23 November 2012, 678-684.

[18] ANR Continuum. Programme VERSO, Continuum ANR-08-VERS-005, 12-2008/09-2012.

Scientific Research Publishing

Submit or recommend next manuscript to SCIRP and we will provide best service for you:

Accepting pre-submission inquiries through Email, Facebook, LinkedIn, Twitter, etc.
A wide selection of journals (inclusive of 9 subjects, more than 200 journals)
Providing 24-hour high-quality service
User-friendly online submission system
Fair and swift peer-review system
Efficient typesetting and proofreading procedure
Display of the result of downloads and visits, as well as the number of cited articles
Maximum dissemination of your research work

Submit your manuscript at: http://papersubmission.scirp.org/
Or contact jcc@scirp.org