Adaptation of Service-Based Context-Aware Applications with FraSCAti Platform

Abdelkader Bouguessa
Department of Computer Science
Ibn Khaldoun University
Tiaret, Algeria
abdelkader.bouguessa@gmail.com

Boudjemaa Boudaa
Department of Computer Science
Ibn Khaldoun University
Tiaret, Algeria
boudjemaa.boudaa@univ-tiaret.dz

Leila Amel Mebarki
Department of Computer Science
Ibn Khaldoun University
Tiaret, Algeria
leilamel.mebarki@gmail.com

Abstract – The dynamic adaptation of running service-based applications without stopping has long been shown to be more than a dream for designers and developers. Recently, several works have been proposed in the literature, where most of them do not have a clear and complete process of adaptation or count on a specific plat-form that decreases the use of these adaptable large-scale applications. This paper aims to present a context-aware dynamic adaptation that covers techniques for handling the impact of context changes onto the execution of context-aware applications. We propose a comprehensive solution with software architecture that enables the dynamic adaptation of services built by assembling components, depending on context execution. This architecture is based on SCA (Service Component Architecture) specification, and implemented with the FraSCAti platform. The SCA specification guarantees pure component assembly and the FraSCAti platform guarantees the automatic reconfiguration. A case study of a modern tourism agency is treated by the proposed adaptation solution in order to exhibit its feasibility.

Keywords – Dynamic Adaptation, Context-awareness, Service-based applications, FraSCAti, MAPE.

1. INTRODUCTION

Advances in technology allow us to design efficient applications, where their contextual conditions are different. In order to meet specific needs, components or services of these applications can act synergistically and communicate with each other within one or different operating systems and across two or more connected computers. The agility, reliability, flexibility and reusability that the service-based architecture (SOA) [1] promises are the main raisons of building service-based applications (SBAs) on such an architecture. This development, if properly designed and implemented, will make these SBAs (such as healthcare, travel, aerospace and defense applications) able to discover and compose services at runtime, thus fulfilling ever-changing requirements. Based on the combination of Service-Oriented Architecture (SOA) and Component-Based Software Engineering (CBSE) [2] principles, the evolving paradigm SCA [3] is principally elaborated to realize large-scale systems and distributed applications. Regardless to communication protocols or programming languages, SCA standard aims to define a set of services in one architecture that stands on four specifications: assembly language, component implementations, bindings, and policies [4]. This paper describes a support to the execution of service-based applications that may require adaptation to tackle changing user context, or even altering environment. Due to the situation where the services may become available or not during the execution of the application, this concept helps constructing service-based applications that are capable to autonomously adapting, and consequently involve the evaluation of novel changes to its current design. The goal is to propose a complete process of adaptation that does not count on a specific platform in order to increase the use of these adapt-able large-scale applications within open, dynamic, and distributed environments.

A general definition of context provided by Dey et al. [5] is stated as “any in-formation used to characterize the situation of entities (i.e., whether a person, place, or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves. Context is typically the location, identity, and state of people, groups, computational and physical objects”. The Context-awareness concept that our solution depends on, determines an application’s behavior, it can be also defined as a response component used to manage environmental awareness issues as well as unforeseen changes by providing better information to designers, and hence they would dynamically adjust their applications.

The particularity of our proposal is to consider the work of Dey et al. [5] that introduced the idea of adaptation by defining context-awareness as a leading concept to the automation of a software system, and to use the FraSCAti plat-form as a run-time support for SCA, which offers management features that allow us to easily perform dynamic adaptation. Intended to implement context-aware applications that can dynamically change or adapt their behavior based on their context, our proposed solution is presented through an interesting Tourism Agency System case study, where it seeks mainly to help travelers finding their suitable destinations in the appropriate time based on monitored information provided by various services. The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 proposes a strategy of dynamic adaptation for service-based applications in which a functional architecture based on adaptation module is detailed. Finally, section 4 reports our Tourism Agency System case study while section 5 concludes this paper, and indicates our future work.

2. RELATED WORK

Various researches found in the literature with the objective to support the dynamic adaptation, notably Context Toolkit [2], SECAS [6], COSMOS [7] and WildCAT [8], have tried to realize frameworks that make applications adaptable. However, most of them proposed incomplete adaptation approaches; where some works performed dynamic adaptation depending on a given infrastructure, and others are limited to an application domain [12]. To the best of our knowledge, SAFRAN [9] and Dynaco [10] are two important works, which present clear dynamic adaptation strategies. SAFRAN enables the creation of self-adaptive applications, and Dynaco offers an adaptation framework, originally designed for applications on grids. According to these two generic frameworks, the adaptation is separated into phases in order to perform several types of adaptations. The MAPE model [11] becomes now a standard reference for specifying a way to divide the different adaptation phases. The limit of these two works is that they are intended to the reconfiguration of component-based applications. Concerning dynamic adaptation of service-based applications, we will consider the generic framework presented by Andrés et al. in [12] to specify different kinds of adaptation in various environments. The authors tackled in detail the four functionalities of the MAPE model [11] (see section C). Nevertheless, the main difference between their proposition and ours is the technologies, techniques and infrastructure. Our selection of FraSCAti platform is motivated by its ability of customization depending on the designer needs; also, it carries runtime reconfiguration capabilities into a SCA environment.

3. A DYNAMIC ADAPTATION OF CONTEXT-AWARE APPLICATIONS

In this section, we will introduce and define the elements of our solution, by mentioning the adaptation fields. We will mention then existing adaptation types, and discuss in detail the ones that we focus on in our work. At the end of this section, we will present our proposed solution, which target dynamic adaptation for context-aware application, and describe each phase of the MAPE model. Cloud Computing, mobile computing, and others are among the research areas that introduce the challenge of applications adaptation, and the ability to reconfigure them so that they will be able to adapt themselves to their executions environment. Adaptation can be considered as the process of modifying an application in order to fit new situations and satisfy specific requirements. An application should be adjusted if its execution context has been changed. Context-aware applications should adapt themselves at run-time, for many different purposes, in particular, to deal with the amount of available resources, and by using more efficient program, the user needs will be better satisfied. Note that introducing facilities for context-aware adaptation in exiting code is a very difficult task.

3.1. Adaptation types

Adaptation types can be defined as mechanisms, which enable the application to react in order to fulfill specific adaptation needs, regarding the evolutions of the context. These are the five possible types of adaptation strategies [9]: parametric adaptation, functional adaptation, structural adaptation, behavioral adaptation, and the environmental adaptation. For the purpose of better control the parametric adaptation [13], our solution will touch only input and output service parameters. The process of modifying those parameters will be not only to ensure the proper functioning but also to leverage new possibilities, which can appear dynamically.

- Input parameter. This parameter comprises the population of operation parameters based on context. One or more parameters values of the request message are replaced with values related to contextual information.
- Output parameter. Response manipulation is also a context-based; where the service responses can be manipulated and modified. The adaptation is done after the service has executed and constructed a response. This can have sorting or filtering operations, for instance.

3.2. Solution overview

This paper presents a novel approach that aims at dynamically adapting the execution of the service-based applications at runtime in case of context changes. As shown in Fig. 1, our proposal relies on four main layers; where, it is important to emphasize that the observation process is done at each layer of the architecture, and equally the adaptation schedules may also be performed at different layers as needed.

3.2.1. OS & hardware layer

This level is the infrastructure that constitutes the context source. For example, in case new OS policies become available, this level may require dynamic adaptation.

3.2.2. FraSCAti Platform layer

At this level, explorer can detect that a service appears or disappears; that means keeping under observation various services' progresses. FraSCAti [14], which is a reflective platform for deploying, hosting, and managing SCA applications, its different subsystems are implemented as SCA components. FraSCAti provides a homogeneous view of a middleware software stack where the platform, the non-functional services, and the applications are uniformly designed and implemented with the same component-based and service-oriented paradigm. This platform has the particularity of facilitating the process of dynamic adaptation of SCA components. Our work is motivated by the fact that FraSCAti is a general model, i.e. it is not specific to an application domain [15]. Moreover, it is an extension of the Fractal model, in other words, it contains the various basic concepts shared by most other models. Technically, the FraSCAti model is very flexible, allowing for many dynamic reconfigurations and especially is designed primarily to be easily extended, and it allows the addition of new features. Likewise, an application built using FraSCAti is seen as a set

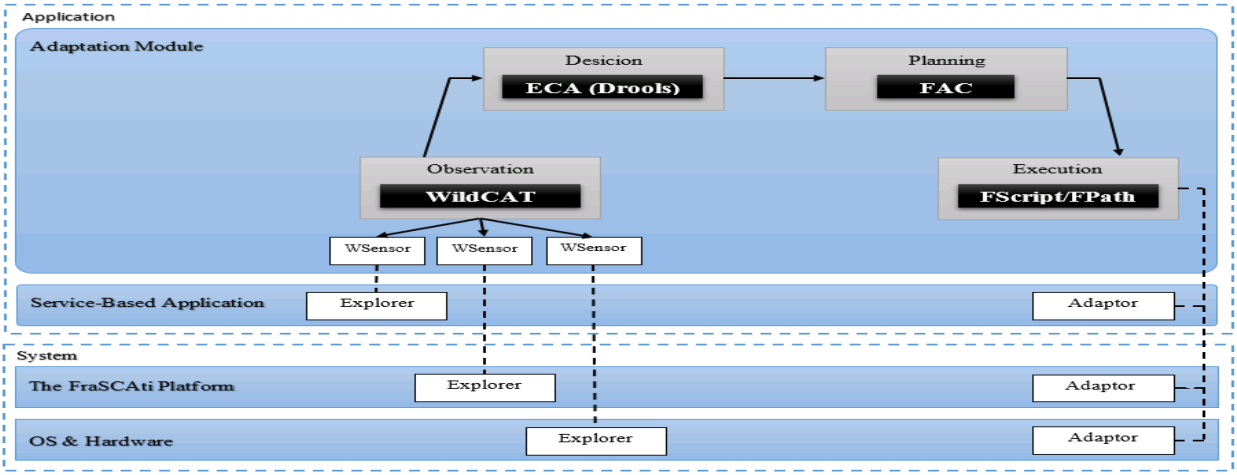

Figure 1: Architecture of the proposed Dynamic Adaptation.

of components and services, these components communicate with each other through referrals. The extension of the FraSCAti model provides better support for synchronization of different application entities during the adaptation process. This dynamic model offers also all the basic functionalities we need without making it more complex than necessary, allowing us to easily integrate our work into the framework of the model itself.

3.2.3. Service-based applications layer

The end user interacts directly with this layer, which represents the application he is using; an adaptation may be triggered via an adaptor, in case the user changes his requirements, which can be detected via an explorer. As obvious in Fig. 1, the present layer contains service-based applications (SBA) with Service Component Architecture (SCA), which defines a general approach that can exploit components, and describe how they work together. In particular, SCA specification defines how to create components and how to combine them into complete applications [16]. Regardless of the language used to build the components of an SCA application and beyond the component technology used, SCA defines a common assembly mechanism for specifying how these elements are gathered within applications. The SCA platform is a framework designed for the development of service-based applications. The main characteristic of this platform is that it supports a wide variety of programming languages for implementing and defining interfaces as well as components. SCA does not offer a new programming language, or a new technology for accessed services, but it can assemble all these existing technologies [17]. Beyond the advantages inherent to this approach, SCA allows easy integration of existing non-SCA development. One more feature of the present platform is the policies that are defined outside the code performing the service may be changed without impact on the application code, which will save time and money by allowing the developer to focus on business logic.

3.2.4. Adaptation module layer

WSensors in this level can listen multiple explorers and push various events directly given to the adaptation module, whose function is not only to keep the application running, but also to make the best reconfiguration based on the new possibilities that may appear during its execution. According to the MAPE model [11], the adaptation module is divided into four main phases: Monitoring or observation, Analysis or Decision, Planning, and Execution.
In the next section, we will describe in more detail the MAPE module as well as techniques and technologies related to each phase of it.

3.3. Adaptation module

This section will give some details about our implementation that relies on the adaptation module. This last is responsible for managing the dynamic adaptation of context-aware service-based application. The MAPE model [11] divides this module into four phases (Fig. 1):

3.3.1. The observation phase

The first phase of the MAPE model corresponds to the observation. It consists of monitoring the context of the application in order to detect changes that need adaptation. This context can change from available system resources (bandwidth, CPU, etc.) to the user preferences and environmental properties (weather, time, etc.) [15]. Monitoring the environment (execution context) of the application and/or the application itself is the detection of the occurrence of certain circumstances requiring adaptation. This step can be implemented using context-aware adaptive application. The technology we used in this phase is WildCAT, a Generic Framework for Context-Awareness [9]. This Framework's objective is to offer a pragmatic approach to ease the creation of Context-Aware applications. Fig. 2 depicts the main function of the WildCAT programming interface [8], which allows discovering the characteristics of the context, reason on it and automatically be notified when specific events occur. The main reasons behind our choice of this system is the simplicity of its use by application programmers, and it's considered generic, i.e. it's not tied to a specific application domain, moreover this system enables and allows an efficient and custom implementations for particular domains. The Adaptation module is connected with each layer through two elements; an Explorer that measures the level's state and extracts new information appearing in that layer, for example at the OS & Hardware layer. The Explorer can determine this level's properties, such as the number of available processors as well as RAM capacity.

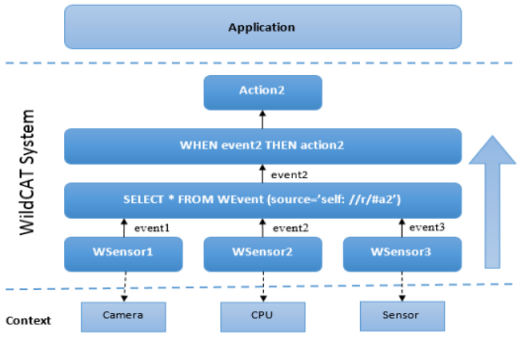

Figure 2: WildCAT overview.

Second element is an Adaptor that is charged of performing adaptation within layers if necessary. In our case every Explorer is linked by a "WSensor". In order to monitor the execution context and detect its changing over time, WildCAT gives a special implementation to the used monitors and after a certain period, the system collects the new contextual information so it will be notified by the occurred events, hence the application adapts to those changes taking into account the new contexts.

3.3.2. The decision phase

Considering the current state of the application and depending on the new contextual circumstances detected in the first phase, the application must decide reconfiguration operations to perform so it adapts to new situations. To do this, the second phase of the MAPE model consists on reactivating an adaptation policy, detecting events occurring in the context and according to these events as well as the current state of the system, an adaptation has to be done for this application. How decision is made and which actions must be performed to execute the reaction are closely related to the information received from the previous phase and to the goal given to dynamic adaptation. Adaptation policies can be realized by a rules-based inference engine, in our case they are implemented using the Drools system. Drools [18] is a Business Rule Management System with a rule-based engine, more correctly known as a Production Rule System, which execute actions based on conditions. Drools has an enhanced and optimized implementation of the Rete algorithm. One of the main reasons of using Drools in our case, is that it is flexible enough [18] to match the semantics of the dynamic adaptation problem. Moreover, the various parts that compose a Drools rule may be extended to lend any domain- specific sense to the rule. As depicted in Fig. 3, a Drools rule has the following structure: A set of Events, which is the conditional parts of the rule, and a set of Actions, which is a block that allows specific actions to be done. The decision phase is the start point of our solution to perform adaptation; it relies on gathered contextual information to decide whether the current situation requires adaptation or not. Once this phase has decided that an adaptation should be done, it transmits reaction to the planning phase. The reaction describes what kind of adaptation should be performed.

3.3.3. The planning phase

This third phase consists on establishing a plan to apply reactions given from the Decision phase, this plan is a set of actions of various types (environmental, structural, functional, behavioral, or even parametric) able to change the current state of the application into its new state. Before applying reconfiguration operations, we must associate those operations to their components, this step is called weaving of adaptation policies. To do so, our approach is based on the aspect paradigm. Aspect-Oriented Programming (AOP) [19] gives us mechanisms to achieve an adaptation as a Cross-Cutting Concern, which means to separate the adaptation code from the functional code of the application. The aspect includes a code that can be grafted (or weaved) in a source code through a program called Weaver and cut-off points or actions. The FraSCAti platform considers the Aspect as a component named FraSCAti Aspect Component (FAC for short). FAC is the mapping of the general model [20] (aspect component (AC), aspect domain, and aspect binding) on the FraSCAti components model. Fig. 3 illustrates how to implement an AC by a sample source code. ACs apply the component methods shown by the client and server interfaces. An implementation of the IntentHandler interface is required when declaring an AC. The invoke method describes the behavior of the aspect. The parameter of the Invoke method is a reification of a FraSCAti interface invocation. It provides a set of methods for a joint point introspection. The argument of the invocation can be changed. The code written in this method will be executed around the join point. The “proceed” call is the original method call.

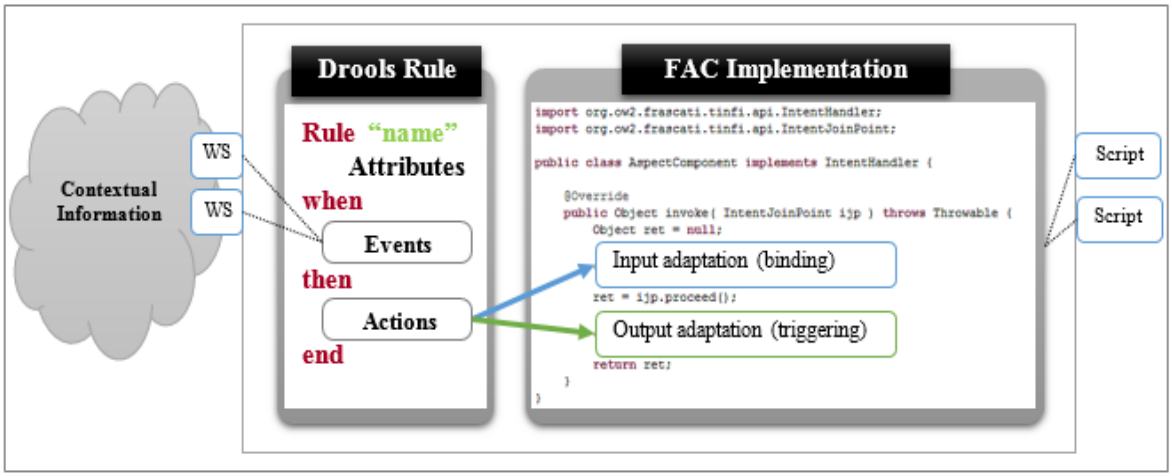

Figure 3: Overview of the implementation process.

The main advantage of considering our solution as an Aspect system is that after the observation of changes that are significant enough, and the decision to react to those changes, the Aspect Component model in particular, FraSCAti component will be able to make a plan given to the execution phase, which addresses the dynamicity of the execution environment.

3.3.4. The execution phase

Once the system knows to what state it has to evolve, we must define how it should do it (execution plan) and run the adaptation. The implementation plan will define the list of operations to be performed by the component to achieve its new state. Thus the order of executing these operations. This plan can be managed by a set of rules for an initial state and a final state given a list of actions to perform. The implementation of the adaptation is the most sensitive part. At this level, the component knows the order of execution of adaptation operations. Although it supports all the features we need, the FraSCAti platform specified as a set of application programming interfaces (APIs). This form does not fit the way we use: it is impossible to guarantee the consistency of reconfigurations performed, the corresponding code is very complex and incomprehensible and specific concepts of FraSCAti does not exist directly in Java. These limitations have led us to choose FScript [21] a language dedicated for the specification and the execution of application and FraSCAti components reconfiguration, FScript includes FPath [21] another language used to navigate inside FraSCAti architecture. FPath can only write expressions and does not have control structures. An FPath expression evaluates without side effects and always returns a value, which can be one of the following types: general classic types (numbers, string and Boolean) and specific types for FPath (homogeneous set of components of interfaces or attributes). A path is a specific expression FPath. This type of expression allows browsing and selecting items inside FraSCAti architectures, with another word FPath is a way to select a set of node that meets certain criteria by navigating along the arcs. To do this, a path consists of a set of successive steps, separated by a slash. Each of these steps is itself divided into three parts: an axis, a test and a collection of predicate possibly empty. The concrete syntax of a path is:

axe1::test1[pred1]/axe2::test2[pred2]/...

The axis is an indenting from a finite set, which corresponds to the possible labels for arcs. The test is either a name (identifier) or a star *. Predicates are complete FPath expressions evaluated for their Boolean values. We now describe the FScript language. Compared to FPath, FScript adds the ability to define new functions (used in FPath expressions) and especially reconfiguration actions, which unlike functions are able to modify the architecture of FraSCAti components. For this, FScript supports a number of control structures and especially enriches the FPath "standard library" with a set of primitive actions that act on FraSCAti components. An FScript program consists of a set of functions and actions. The difference between the two is that the functions are strictly no side effect on the architecture FraSCAti: Only actions can invoke other actions. The only

syntactic difference between functions and actions appear in their definitions. Functions are introduced by the keyword function while actions use the keyword action. The body defining a process consists of a sequence of instructions.

4. Tourism agency case study

Aimed at improving our proposed architecture and its implementation, we have chosen “Tourism agency System” as an illustrative case study, which is one of the most important and required services in the market at the global scale. This web application shown in Fig. 4, allows the visitor to find a destination among different touristic regions or events belong to the visited country, which is located by the system based on the visitor IP address. It also provides a chronological classification for each touristic site so the visitor can easily move to the selected destination. To get there, he can choose a conveyance from a given list with the price of each one, after that the total of the tour is calculated automatically by the system. In order to strengthen and develop a context-aware Web application (reconfigurable), we have added other functions to the previous features by using the power of social networking tools (Twitter or Facebook in our case) to gather the user’s contextual information based on his account preferences. At the end of the whole process, the application offers a conversion of the calculated total price using the currency of the visitor in accordance with the exchange rate. To more clarify the implementation of this runtime adaptation, Fig. 4 illustrates the tourism agency services composite, which covers four components representing the following services: The "Tourism Agency Component" encapsulates all calls to other services; it also contains two properties that describe the expected user input: Twitter or Facebook account, plus the date of the visit. The second service "Visiting Service Component" displays the list of the touristic areas according to the user preferences and his current detected location from his IP address. It also suggests indoors and outdoors places in a specific order referring to the user favorites, and time conditions; for example if the condition time is good it proposes both of places, while only indoors places will be recommended if the condition time is bad. The "Travel Service Component" displays the list of available means of transport leading to the selected site and a map between his current location and the selected destination. The last service "Banking Service Component" converts the calculated total, which consists of two prices: the visit price, and the transport price.

Fig. 5 demonstrates two possible scenarios seamlessly supported by our new architectural solution. In the first scenario (Fig. 5.Case1), the application starts by the observation phase and shows (Fig. 5.A) in the home page. When the user introduces no input information, i.e. no

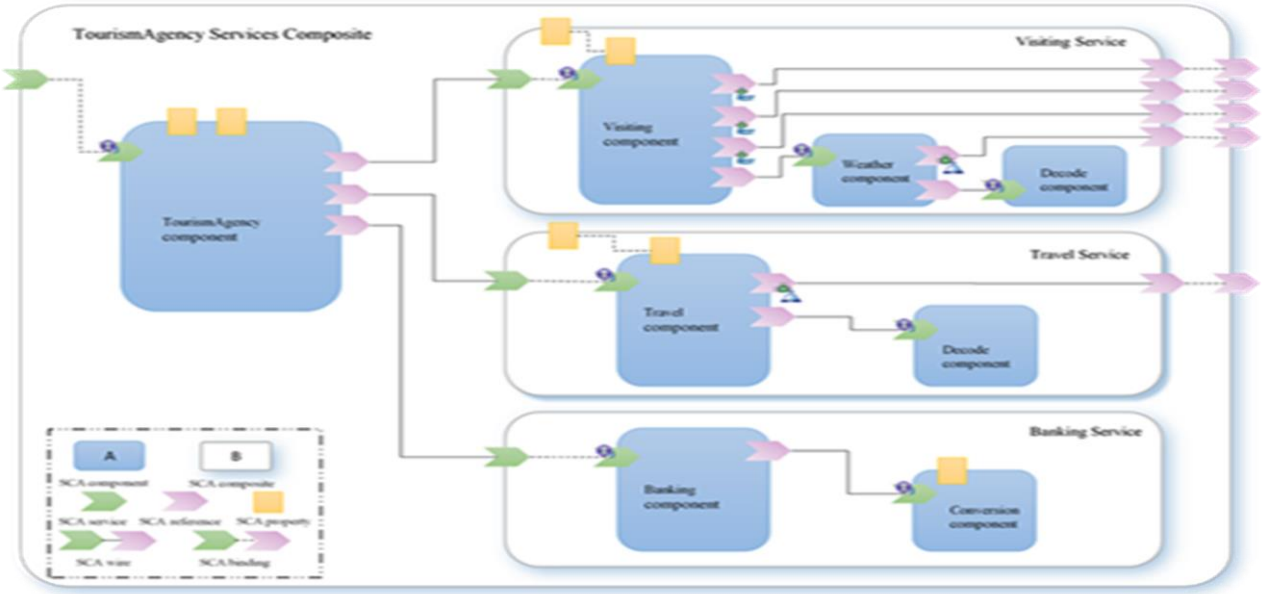

Figure 4: General Architecture of Tourism Agency Composite.

contextual changes are applied; the system displays his current location detected from his IP address, as well as a brief description of that area and its corresponding weather information. After selecting the suitable site (Fig. 5.D); which represents a contextual change for the application, this last reacts by showing some information about the located area and a table of the current conditions and forecasts including seven day outlook (Fig. 5.E). When the user chooses a day from this showing table, which means that the application has detected another contextual change and needs to be adapted again, the process of adaptation is applied for the second time. The application shows a list of means transports with their prices, a map that helps the user to reach the visiting location will be displayed as well (Fig. 5.F). The total prices will be shown (Fig. 5.G) only after the selection of a mean traveling. This scenario depends always on the user interaction.

Figure 5: Demonstration of the running application.

In the second scenario (Fig. 5.Case2), the user introduces his Facebook or Twitter username as an input, which allows detecting contextual changes during the first phase of the MAPE model (the observation phase) as long as the user updates his account settings and preferences and attempt to use our web application. Liked pages, visited areas, tweets, and others factors can permit the application to decide (at the decision phase) automatically and then plans a new action (at the planning phase) to propose when using the application. In the end those plans will be executed (Fig. 5.B) at the execution phase. The second one that covers more dynamicity explains our implemented solution (Fig. 5.C) which consists of auto-charging different displayed information such as the selected touristic area, the preferred visiting day, and the mean of transport. All these features will help the visitor to make suitable decisions, save time and money.

The strength of the services offered by the "Tourism Agency" resides on the fact that defining at design time all possible contexts and their evolutions is not obligatory needed. That means that the agency system will be drastically able to plan and react even with the unexpected context changes, hence specific user's requirements will be satisfied according to the adaptation strategy.

The main contribution of this proposal is the strength architecture that has been illustrated with the above-described case study, in which we applied the MAPE Model for developing adaptive service-based applications. We also demonstrated how the specific features of the FraSCAti platform could be exploited during the application's adaptation process, and then how the defined dynamic adaptation strategy can be used to develop high quality, flexible, and scalable service-based applications.

5. Conclusion

The present work stands on the FraSCAti Platform used to develop and adapt dynamically the SCA-based context-aware applications. SCA is a standard for the distribution of Service-Oriented Architectures (SOA). The novelty of FraSCAti is to bring the adaptation of applications during their execution according to

changes in their context. The solution provided in this work is performed by a case study on tourism services by presenting a variety of choices in order to satisfy customer's needs. Aiming at making the application benefit from appearing contexts. In addition, other examples can also verify the feasibility of the proposed solution and demonstrate the effectiveness of the dynamic adaptation using FraSCAti. Our future work is to strengthen the flexibility, and the semantic of the decision phase by utilizing other rule-based engines as SWRL rules combining OWL, and RuleML scripts [20]. Moreover, we want to fully achieve other aspects of contextual adaptation by addressing the behavioral aspect (changes of SCA component's behavior), and the structural aspect of a SCA composite (changes in the structure of the components).

6. REFERENCES

[1] N. Bieberstein, S. Bose, M. Fiammante, K. Jones, and R. Shah, "Service-Oriented Architecture Compass : Business Value, Plan-ning, and Enterprise Roadmap," Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[2] J.Q. Ning, "Component-Based Software Engineering (CBSE)," In 5th International Symposium on Assessment of Software Tool (SAST'97), 1997, pp. 34-43.

[3] M. Beisiegel, D. Booz, A. Colyer and K. Team, "Service Component Architecture," Novomber. 2007.

[4] D. Fournier, P. Merle, and al., "ANR SCOrWare Project: WP1 – SCA Platform Specification," April. 2009, www.scorware.org/.

[5] A. Dey, G Abowd and D Salber, "A Conceptual framework and toolkit for supporting the rapid prototyping of context-aware applications," 2001, pp. 97-166.

[6] T. Chaari, "adapt applications to new contexts of use". SECAS Project: INSA Lyon, 2006.

[7] C. Sophie, C Denis, and T Cahantal, "Service Management Context COSMOS," ADAPT09, September. 2009.

[8] P.C. David and T. Ledoux, "Wildcat: a generic framework for context-aware applications," in MPAC '05: Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing. New York, NY, USA: ACM, 2005, pp. 1–7.

[9] P.C. David and T. Ledoux, "An aspect-orienteach for developing selfadaptive fractal components," in Soft-ware Composition, ser. Lecture Notes in Computer Science, vol. 4089. Springer Berlin / Heidelberg, 2006, pp. 82–97.

[10] J. Buisson, F. André, and J.L. Pazat, "Supporting adaptable applications in grid resource management systems," in 8th IEEE/ACM International Conference on Grid Computing, Austin, USA, pp. 19-21, September. 2007.

[11] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," Computer, vol. 36, no. 1. 2003, pp. 41–50.

[12] F. André, E. Daubert, and G. Gauvrit, "Towards a Generic Context Aware Framework for Self-Adaptation of Service-Oriented Architectures," in 5th International Conference on Internet and Web Applications and Services, Barcelona, Spain, 2010, pp. 309–314.

[13] M. Kapitsaki, A. Kateros, N. Prezerakos, S. Venieris, "Model-driven development of composite context- aware web applications," Information and Software Technology 51, 2009, pp. 1244–1260.

[14] R. M´elisson, P. Merle,D. Romero,R Rouvoy, and L Seinturier. "Reconfigurable run-time support for distributed service component architectures," Automated Software Engineering, Tool Demonstration, Antwerp Belgique, 2010. URL http://hal.inria.fr/inria-00499477/en/.

[15] L. Seinturier, P. Merle, "A Component-Based Middleware Platform for Reconfigurable Service-Oriented Architectures," April. 2012.

[16] C. David, "Introducing SCA," white paper, Chappell & Associates, July. 2007.

[17] L. Seinturier, P. Merle, D. Fournier, N. Dolet, V. Schiavoni, and J.B. Stefani, "Reconfigurable SCA Applications with the FraSCAti Platform," In Proceedings of the 6th IEEE International Conference on Service Computing (SCC'09), pp. 268–275, September. 2009.

[18] Drools docs. http://docs.jboss.org/drools/release/5.2.0.Final/droolsexpert-docs/html/ch05.html.

[19] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.M. Loingtier, and J. Irwin, "Aspect-Oriented Programming," In Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP'97), volume 1241 of LNCS, pp. 220–242. Springer, June. 1997.

[20] N. Pessemier, L. Seinturier, L. Duchien, and T. Coupaye, "A Model for Developing Component-Based and Aspect-Oriented Systems" In Proceedings of the 5th International Symposium on Software Composition (SC'06), volume 4089 of LNCS, pp. 259–274. Springer, March. 2006.

[21] P.C. David, "Développement de composants Fractal adaptatifs : un langage dédié à l'aspect d'adaptation," PhD thesis, University of Nantes / Ecole des Mines de Nantes, 2005.