Requirements Monitoring for Adaptive Service-Based Applications

Marc Oriol[1], Nauman A. Qureshi[2], Xavier Franch[1], Anna Perini[2], and Jordi Marco[1]

[1] Universitat Politècnica de Catalunya, Barcelona, Spain
{moriol,jmarco}@lsi.upc.edu, franch@essi.upc.edu
[2] Fondazione Bruno Kessler - CIT, Trento, Italy
{qureshi,perini}@fbk.eu

Abstract. [Context and motivation] Adaptive Service Based Applications (SBAs) need to cope with continuously changing environments. Monitoring becomes a key requirement for engineering Adaptive SBAs. **[Question/problem]** Ongoing research on Requirements Engineering (RE) for Adaptive SBAs strives to answer challenging questions such as how to monitor changes affecting user's requirements? and how the monitored information helps in adapting to the candidate solutions? **[Principal ideas/results]** Existing approaches and techniques to specify requirements monitoring for Adaptive SBAs are either formal or specialized to a particular domain. A convenient and easy approach to specify requirements monitoring for Adaptive SBAs is still missing. In this paper, we focus on this issue. **[Contribution]** We describe a systematic approach for deriving requirements monitoring specifications for the running Adaptive SBA. We use a running example from a travel domain case study to elaborate our approach.

Keywords: Requirements Monitoring, Self-Adaptive Systems, Services-Based Application.

1 Introduction

Service-Based Applications (SBA, hereafter) reply on third party services while operating in an open environment (such as the Internet) [1]. In such a dynamic environment, SBAs must adapt in response to changing end-user's needs and preferences (e.g. book travel using different services), changes in context (e.g. wifi service is available in downtown, but is not available in a mall nearby) or variation in the availability of resources to exploit such solutions (e.g. mobile battery went down) or the availability of the service (e.g. travel service is not available due to server maintenance). Research on self-adaptive systems has started to gain considerable attention from the research community [2]. However, research on Requirements Engineering (RE) for Adaptive SBAs has received less attention.

Existing works in the field of service-oriented computing aims at architectural aspects when focusing on service monitoring and discovery [3]. In the context of RE, requirements monitoring has been tackled as a way to observe the deviations in the running system by instrumenting the code [4,5,6]. However, these approaches anticipate changes that might occur at runtime, which makes them limited in the case of adaptive

B. Regnell and D. Damian (Eds.): REFSQ 2012, LNCS 7195, pp. 280–287, 2012.

SBAs. Recently it has been pointed out that to cope with unanticipated changes that can occur dynamically at runtime, self-adaptive systems need to be aware of their own requirements and end-user's needs at runtime [7]. Taking this vision to adaptive SBAs, in several cases, the decision on how to adapt in response to changes and what to monitor can be postponed to runtime as well with respect to the real environment involving the end-user. In the context of Self Adaptive Systems, there are many instances where adaptation decisions cannot be determined at design time. For instance, if a flight is delayed (unanticipated event) the Self Adaptive System may choose to rebook a similar flight, cancel the flight and hotel booking or explicitly involving the end-user asking for what to look for (e.g. travel by train, rent a car, etc). Such decision cannot be pre-fixed, as dynamic changes may occur at run-time.

In this work, we consider changes that pertain to end-user requirements, operational contexts and variability in resource's availability posing challenging questions to the field of RE. In particular, we aim to address the following research questions: (1) how to systematically obtain and configure monitors from end-user's requirements? (2) how to configure an adaptive SBA to adapt at runtime in response to changes in operating context, availability of resources and by involving the end-user if needed?

To address these questions, we envision a novel approach to systematically derive monitoring specifications from the user's requirements for a running adaptive SBA. We adopt an operational pattern based on Event-Condition-Action to configure adaptive SBAs to monitor changes and adapt at runtime.

The paper is organized as follows. Section 2 describes the related work and highlights the challenges. Section 3 briefly recalls the baseline of our proposed work [8,9] and describes our envisioned approach on requirements monitoring for adaptive SBAs. Section 4 summarizes the next steps.

2 Related Work and Baseline

Relevant works on requirements monitoring are briefly recalled here below. In [4,5] a formal language (Formal Language for Expressing Assumptions - FLEA) is proposed to express the assumptions about the environment that has to be monitored as prerequisite in order to apply remedial actions if the related requirements are violated. Similarly, in [10] a monitoring framework, named ReqMon, is proposed for monitoring web service requirements expressed using a goal-oriented language (KAOS). KAOS model of requirements is used to analyze obstacles for specifying monitors. Another framework to monitor and diagnose failures of software requirements has been proposed in [11]. The framework logs the execution of the system, and a diagnostic component identifies if there has been any violation of the requirements by means of propositional formula in CNF and using SAT solvers. In [12] an approach to deal with self-adaptation of BPEL compositions by means of adaptive goals, which are responsible for the evolution/adaptation of the goal model, is presented. Using the KAOS goal model they transform the obstacles and additional conditions into the languages of two monitoring systems: ALBERT, Dynamo which are used to evaluate properties of a BPEL process.

A comparison between these works and our envisioned approached is shown in table 1.

Table 1. Comparison with the related work

Framework	Usage of a Goal Model	Adaptation support	Different Monitoring Configurations	Goal reasoning (bottom-up)	General / Service Based
Fickas et al. [4,5]	Yes: *KAOS*	No	No	Partially *based on activities*	General
Robinson[10]	Yes: *KAOS*	No	Partially *Implemented by the designer*	-	Service
Wang et al.[11]	Yes	No	Partially *Testing not supported*	No	Service
Baresi et al.[12]	Yes: *KAOS*	Yes	No	-	Service
Our approach	Yes: *ARML*	Yes	Yes	*Future work*	Service

These works on requirements monitoring tend to consider only changes that can be anticipated at design-time. This limits their applicability in case of adaptive SBAs. Many decisions need to be postponed to runtime while engineering adaptive SBAs. In context of RE for adaptive SBA, requirements monitoring demands a flexible approach to derive and configure application monitoring with respect to the changes in the requirements or in the environment. An easy and convenient approach to support the analysts at design-time to derive and configure application monitoring with respect to the requirements and later provide the support to the running adaptive SBA at runtime to automate it monitoring and adaptation with respect to the changes. To address this target, we envision a convenient and systematic approach that enables the analyst to express monitoring specification from requirements (without obfuscating the requirements specification using a complex formal language), and provide supporting features to re-configure at runtime.

The baseline of our envisioned approach is our ongoing works on the Continuous Adaptive Requirements Engineering (CARE) [8] Framework and the SALmon Framework for Monitoring SLA [9].

The CARE framework attempts at bridging the gap between design-time and runtime RE. At design-time requirements model is constructed using the concepts (i.e. goals, tasks, context, resources etc.) defined in the revised core ontology of RE for self-adaptive systems in [13]. The resulting instances of the requirements specification (i.e. candidate solutions to the requirements problem) are stored in the requirements database. At run-time, the CARE is instantiated by a running adaptive SBA, performing RE by itself. It exploits the requirements specification instances for runtime refinement of requirements by involving the end-users, if needed, to satisfy their needs exploiting the available services.

SALMon is a framework focused on monitoring the quality of service (QoS) of web services, evaluate them accordingly to stated conditions, and notify violations to the interested parties. For this project, SALMon has been extended with new measurement capabilities, such as monitoring the change of status of a service, which goes beyond QoS. SALMon is able to combine both passive monitoring and testing approaches accordingly to the preferences of the user. The framework has been implemented as a

SBA itself, providing hence easy integration with other frameworks. It provides the following two services: the Monitor, responsible to retrieve the data of the target services; and the Analyzer, responsible for the evaluation of conditions.

3 Requirements Monitoring Framework

In this section, we elaborate our overall envisioned approach to derive monitors from the requirements, as well as the rules that guides the system adaptation in response to changes detected from the monitoring data. We exploit a Event-Condition-Action pattern to operationalize the requirements as a monitoring specification which is used to configure the running adaptive SBA with respect to the requirements.

3.1 Scenario

We elaborate our approach exploiting a scenario from a Travel Companion exemplar case study (adopted from [8]). Travel Companion is an adaptive SBA, responsible for managing users' travel booking by maintaining users' goals. In this scenario, the user must be notified about changes about her flight itinerary i.e. flight booking status (e.g. flight status changes to delayed or canceled). The notification message about her flight booking status must be sent on her device (e.g. mobile phone) instantly (e.g. with in less than 5 mins) exploiting the available services (e.g. the Internet wifi, flight booking service, SMS service etc.), keeping in view her preference (e.g. send email on a corporate mail account while in office) and context (e.g. location: outdoor, indoor).

3.2 The Framework at Design-Time

We describe our envisioned approach that supports the analyst at design-time to conveniently specify requirements and derive monitoring specification that is used to configure the components of the running adaptive SBA as shown in Fig. 1. We use the above scenario to help to clarify the elements of our approach.

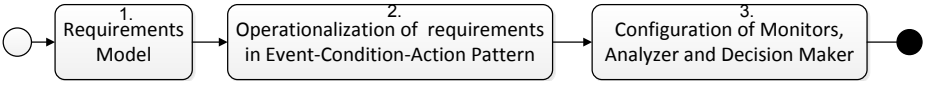

Fig. 1. Design-time process for deriving and configuring Monitor, Analyzer and Decision Maker

1. Requirements Model: The requirements model is defined by exploiting the concepts and relations defined in the revised core ontology of requirements for self-adaptive systems in [13]. Concepts includes: goals, softgoals, tasks, resources, domain assumptions (i.e. conditions considered to be true for the correct behavior of the system), quality constraints (i.e. requirements that expresses conditions over the expected quality of service), context (i.e. information that defines the system state, user's presupposed

information about a requirement etc.) and relations includes: preferences (i.e. defining priorities over mandatory or optional requirements), conflicts (i.e. inconsistent set of requirements), inference (i.e. a generalized relation over decomposition such as AND/OR in goal models). The resulting model describes the requirements specification, which not only the mandatory requirements but also encompass monitoring specification, evaluation criteria and alternative candidate solutions for the intended Travel companion SBA.

2. Operationalization: To operationalize the given requirements, there exist several alternatives, such as using the Object Constraint Language (OCL) [14], Event-Condition-Action (ECA) [15] or Temporal Logics [5] [16], beyond others. We adopt a convenient Event-Condition-Action (ECA) pattern that helps expressing the adaptive requirements specification. Although, ECA pattern for expressing requirements is not the most compact and only form. We chose this pattern to provide a straight forward operationalization of adaptive requirement, thereby capturing the feedback loop functions (i.e. monitoring specification, evaluation criteria and adaptation/trigger actions, making them explicit using ECA rules). The operationalization of these requirements is as follows:

Specifying Events: The analyst can include either goals or tasks to monitor. The framework navigates through the given defined element in the requirements model until it reaches the leaf tasks that implement the functionality and generate the events to observe. For instance, from a high-level goal 'changes over the flight itinerary being monitored', the framework reaches the task 'invoke flight status' and monitor the events of this task. The current framework supports the generation of monitors for web services. In order to automate the generation of monitors, the analyst annotates these tasks with the required information (i.e. endpoint, WSDL and SOAP action). The invocation of these tasks are the events to monitor. The concrete properties to monitor on each event are obtained from the Quality Constraints defined in the requirements model that applies over the task. The requirements model includes also a set of preferences i.e. Preference Requirements (PRs). PRs specify preferences regarding how the monitoring should be performed (i.e. actively invoking the service every time-interval or passively observing the interaction between the system and the end-user). This information is used to automatically generate a Monitoring Specification, an XML file that describes what is to be monitored, and is used in order to generate the monitors accordingly.

Specifying Conditions: The list of elements in the condition specify the rules of the system to analyze. These rules are checked on runtime to detect if the behavior of the system fulfills the expected functionality with the desired performance. The given elements involved in the Condition section are the Quality Constraints (QCs) that specify the conditions to check, and the runtime data obtained as Resources (i.e. the results of the monitored events). This information is used to automatically generate of the Condition Specification, which specifies the conditions to be checked at runtime.

Specifying Actions: This part consist on the execution of an action over the defined elements in the model. There are several kind of actions that can be performed in order to correct or mitigate the malfunction of the system. Currently we have focused on two kind of actions to perform over the requirements model. Namely, SELECT and INVOKE. Operationalizing the SELECT(task): the element included as a parameter in

the SELECT function is a composite task that can be met by several alternatives. This action defines the preferred alternative to execute at runtime. For instance, in the given scenario, there is a task 'NotifyUser' composed of several alternatives (e.g. NotifyByEmail, NotifyBySMS,etc). When a condition over these tasks is not met, the action SELECT(NotifyUser) is triggered, which updates the selection of the most convenient device to notify the user. INVOKE(task): the element included in the INVOKE function is a task that is executed by the system as a result of the failure of the condition. For instance, if the flight has been delayed, INVOKE(NotifyUser) notifies the user to his most convenient device that the flight has been delayed. The set of defined actions are used to generate the actions specification.

3. Configuration: From each generated specification using the ECA rules, the components of the running adaptive SBA i.e. Monitor, Analyzer and the Decision Maker are configured. Here we exploit monitor of SALMon framework, which is configured from the Monitoring Specification, providing hence at runtime the monitored information of the target services to the Analyzer. The Analyzer, which is configured from the Condition Specification, checks if the rules are fulfilled or not and notifies any violation to the Decision Maker (i.e. part of the adaptive SBA itself, which instantiate CARE framework). The Decision Maker is configured by means of the Action Specification, which triggers the defined actions.

3.3 The Framework at Run-Time

In this section, we describe our runtime architecture that combines both CARE's runtime process (instantiated by Travel Companion) and SALMon that provides runtime monitoring information to Travel Companion.

Monitoring the Events: The resulting Monitor Specification is used as the input to configure the monitor of SALMon accordingly. The monitor can be configure in either passive or active way (i.e. by passively observing the invocation over the defined services or by invoking systematically the target services in different time intervals). Once the service is invoked, the monitor retrieves the desired information, which can be the value of a quality metric or the result of the invocation.

Analyzing the Conditions: The Analyzer is configured to check the conditions stated in the Condition Specification. During execution, the analyzer is subscribed to the new values that the monitor retrieve. That is, for each new monitored value, the analyzer checks the fulfillment of the conditions. Currently the conditions are stated as a tuple of $< property, operand, value >$. If the conditions are not met, the Analyzer notifies the violation to the Decision Maker.

Triggering the Actions: The Decision Maker retrieves the failure of a condition, and triggers the defined actions. The Decision Maker is composed of several decisions modules, each one responsible for a concrete kind of action to perform. As stated previously, we have defined two kind of actions, namely SELECT(task) and INVOKE(task). The SELECT action is achieved by means of updating the model with the preferred concrete task that will realize the composed task. The trigger action is achieved by means of invoking the specified task. To this aim, the given task is implemented as a

service, and the invocation is performed as a SOAP-based message invocation. In the given scenario, the status of the flight is monitored actively by the monitor through a web service interface. For each invocation, the analyzer checks if the status of the flight is 'OK'. In case the status is 'Delayed' or 'Canceled' the Analyzer triggers the Decision Maker, which performs the action INVOKE(NotifyUser).

4 Conclusions and Future Work

In this paper, we have proposed a systematic tool-supported approach for deriving monitoring specifications from the users requirements for a running Adaptive SBA. Our proposal provides a tool set that allows linking requirements models with more concrete operational artifacts, i.e. adaptive requirements expressed as ECA rules, and deriving monitoring specifications from requirements model elements. Such specifications are used to implement and configure our monitoring framework, which is flexible enough to accommodate changes (e.g. changes in monitoring specification), and to configure an adaptive SBA to adapt in response to observed runtime changes. We adopted Event-Condition-Action pattern in order to operationalize the requirements specification. ECA rules are then used to specify and configure automatically the different components of the adaptive SBA presented in the framework at design-time. At runtime, the monitors provides observed data to analyze the execution of the adaptive SBA. Realizing this framework will help bridging the gap between the design-time and run-time, which exists in the current approaches. To implement monitors and analyzer we exploited SALMon (for monitoring the events and evaluation the conditions) and for decision maker, we exploited Companion SBA, which instantiate CARE (for triggering the defined actions).

Currently we have implemented the generation of monitors from the requirements model. As an ongoing work, we plan to validate the overall process by realizing and evaluating our envisioned framework. We aim to conduct empirical studies which demonstrate the suitability of our envisioned approach. By one hand, we will conduct tests to assess the performance of the implemented framework, by the other, we plan to perform an evaluation of the usability by means ofstudents using the framework.

Acknowledgments. This work has been supported by the research project ADICT, TIN2007-64753, MCyT, Spain. Marc Oriol has a FPI grant bound to the project TIN2007-64753.

References

1. Di Nitto, E., Ghezzi, C., Metzger, A., Papazoglou, M., Pohl, K.: A journey to highly dynamic, self-adaptive service-based applications. Automated Soft. Eng. 15(3-4), 313–341 (2008)
2. Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Di Marzo Serugendo, G., Dustdar, S., Finkelstein, A., Gacek, C., Geihs, K., Grassi, V., Karsai, G., Kienle, H.M., Kramer, J., Litoiu, M., Malek, S., Mirandola, R., Müller, H.A., Park, S., Shaw, M., Tichy, M., Tivoli, M., Weyns, D., Whittle, J.: Software Engineering for Self-Adaptive Systems: A Research Roadmap. In: Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (eds.) Self-Adaptive Systems. LNCS, vol. 5525, pp. 1–26. Springer, Heidelberg (2009)

3. Baresi, L., Ghezzi, C., Guinea, S.: Smart monitors for composed services. In: ICSOC 2004: Proceedings of the 2nd International Conference on Service Oriented Computing, pp. 193–202. ACM, New York (2004)
4. Fickas, S., Feather, M.S.: Requirements monitoring in dynamic environments. In: RE 1995: Proceedings of the Second IEEE Intl. Symp. on Req. Eng., p. 140. IEEE CS (1995)
5. Feather, M.S., Fickas, S., Lamsweerde, A.V., Ponsard, C.: Reconciling system requirements and runtime behavior. In: IWSSD 1998: Proceedings of the 9th Intl. Workshop on Software Specification and Design, p. 50. IEEE CS (1998)
6. Robinson, W.: Monitoring web service requirements. In: Proceedings of 11th IEEE International Requirements Engineering Conference, pp. 65–74 (September 2003)
7. Sawyer, P., Bencomo, N., Whittle, J., Letier, E., Finkelstein, A.: Requirements-aware systems a research agenda for re for self-adaptive systems. In: 18th IEEE Intl. Requirements Eng. Conf., Sydney, Australia, pp. 95–103 (2010)
8. Qureshi, N.A., Perini, A.: Requirements engineering for adaptive service based applications. In: 18th IEEE Intl. Requirements Eng. Conf., Sydney, Australia, pp. 108–111 (2010)
9. Oriol, M., Franch, X., Marco, J., Ameller, D.: Monitoring adaptable soa-systems using salmon. In: Workshop on Service Monitoring, Adaptation and Beyond (Mona+), pp. 19–28 (2008)
10. Robinson, W.N.: A requirements monitoring framework for enterprise systems. Requirements Engineering Journal 11(1), 17–41 (2006)
11. Wang, Y., McIlraith, S.A., Yu, Y., Mylopoulos, J.: Monitoring and diagnosing software requirements. Autom. Softw. Eng. 16(1), 3–35 (2009)
12. Baresi, L., Pasquale, L.: Live goals for adaptive service compositions. In: ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2010 (2010)
13. Qureshi, N.A., Jureta, I., Perini, A.: Requirements Engineering for Self-Adaptive Systems: Core Ontology and Problem Statement. In: Mouratidis, H., Rolland, C. (eds.) CAiSE 2011. LNCS, vol. 6741, pp. 33–47. Springer, Heidelberg (2011)
14. Souza, V.E.S., Lapouchnian, A., Robinson, W.N., Mylopoulos, J.: Awareness requirements for adaptive systems, Technical Report DISI-10-049, DISI, Universit‘a di Trento, Italy (2010)
15. Knolmayer, G., Endl, R., Pfahrer, M.: Modeling processes and workflows by business rules. In: Business Process Management, pp. 16–29 (2000)
16. Baresi, L., Pasquale, L., Spoletini, P.: Fuzzy goals for requirements-driven adaptation. In: 18th IEEE Intl. Requirements Eng. Conf., pp. 125–134 (2010)