MENTA: A Quality Analysis Approach for Self-Adapting Systems

Boris Perez and Dario Correal

University of Los Andes
Systems and Computer Engineering Department
Cra 1E No 19A-40, Bogota D.C, Colombia

Abstract. Self-adaptive behavior is a feature which architects needs to include in their systems in order to improve its reliability. However, despite several ways to get it, it is still hard to implement a self-adaptive system focused on non-functional properties. Difficulties to express quality attributes in the system without combining business logic with the self-adaptation logic and to include new services on runtime are some of them. In this paper we propose a model-driven analysis approach to offer a mechanism which allow the desired quality requirements and adaptation strategies in a system to be expressed in a simple and non-intrusive manner and, to offer a code generation mechanism which takes the models created under the first objective and generates the necessary code for autonomously monitoring and adapting a SOA system.

Keywords: BPM, Metamodel, Self-Adaptation, Software Quality

1 Introduction

Service Oriented Architecture (SOA) is one of the most commonly architectural styles within the context of enterprise information systems [11]. As a part of SOA, web services can meet the same functional requirements, such as recording a sale, but they can also have differences at a non-functional level, in terms of different response times, availability levels or security arrangements [5]. These non-functional properties are also known as *System Qualities* or software quality attributes. Quality attribute scenarios [2] are proposed by precisely defining the quality attributes of an application. This is an activity that should traditionally be performed during the design stage [6]. However, when the application is running, it is not always possible to guarantee achievement of the required quality attributes, due to external factors like faults in 3rd party services.

One way of reacting when faults occur is to stop the running process, make the necessary corrections and restart the system. Though this is not always possible because there are critical business processes that cannot be halted [10]. The ideal solution in this situation would be for the system itself to be capable of reacting when a fault occurs, such as replacing the defective service with another

that meets the desired functional and quality requirements. Ideally, this would take place without any human involvement, in other words, autonomously. According to [4],"a self-adaptive system evaluates its own behaviour and changes behaviour when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible".

Different traditional approaches offer solutions to this. One of them uses Event-Condition-Action (ECA) rules [1]. For ECA-Rules, alternatives like Jess [8] allow definition to functional level. Usually, ECA rules are embedded in application code and will run in the specific order written by the programmer. A further mechanism for implementing self-adaptation consists of managing and controlling exceptions [6]. Exceptions enable the functionality of the system to be controlled if unexpected events occur during execution. According to Garlan [6], exceptions are good for catching an error as it is detected, but are nonetheless weak when it comes to detecting subtle system anomalies. But, these approaches bring with them some limitations; Firstly, it is difficult to precisely express quality attributes in the adaptation conditions to be evaluated. Most of these mechanisms are closely coupled to the application, at code level, and contaminate the business logic with the self-adaptation logic, which worsens maintenance. A second limitation is the problem that exists when it comes to performing a discovery process with respect to new services on runtime. The available alternatives are generally defined in the code.

The main contribution of this paper is to present MENTA, a quality analysis approach for self-adapting systems. MENTA is a method which enables the software architect to use a model-driven strategy for expressing and analysing the desired self-adaptation strategies, based on the quality scenario concept. MENTA is responsible for: 1) looking for the best service in a pool of services meeting the quality needs, 2) calculating the metrics of the invoked service and 3) allowing the inclusion of new services that provide the required functionality. In order to accomplish with this, this proposal performs the following actions: Firstly, it sets out to offer a mechanism that is based on models which allow the desired quality requirements and adaptation strategies in a system to be expressed in a simple and non-intrusive manner. And secondly, it aims to offer a code generation mechanism which takes the models created and generates the necessary code to autonomously monitor and adapt a SOA system. The development and presentation of the proposal will be conducted based on the framework proposed by Salehie [12], in which the decision process for selection of an adaptation strategy in a self-adaptive system based on quality attributes can be defined as the set *<Request (R), Goal Repository (G), Domain (D), Utility (U) >*.

The remainder of this paper is organized as follows: Section 2 expands on the quality attributes concept and the framework proposes by Salehie. Section 3 describes the case study that was used for presenting and validating our proposal. Section 4 reviews the MENTA proposal, based on the reference framework

defined in [12]. Section 5 presents partial results of the experimenting process. Section 6 gives an overview of the related work. Finally, Section 7 contains the conclusions that have been drawn from this proposal.

2 Background

2.1 Quality of Service Attributes and Quality Attribute Scenarios

In [7] the author proposes that quality of service attributes can be classified into two categories: *Certain attributes and uncertain attributes*. Certain attributes, such as security, can be fixed and they are unvaried. On the other hand, uncertain attributes, such as response time, can be varied due to the dynamic heterogeneous environment that the service is operating. Response time, availability and reliability are the generic QoS criteria for web service selection and guarantee the successful use of the service. The characteristics of these three QoS attributes are uncertain. In our proposal we will focus on the uncertain attributes, because they are constantly changing and the customer has no direct control.

On the other hand, a quality attribute scenario, as seen in Bass [2], is a quality-attribute-specific requirement. It consists of six parts: *Source of stimulus* (a human or any other actuator that generated the stimulus), *Stimulus* (a condition that needs to be considered), *Environment* (the system may be in an overload condition or may be running when the stimulus occurs), *Artifact* (some artifact is stimulated, may be the whole system or some pieces of it), *Response* (the activity undertaken after the arrival of the stimulus) y *Response Measure* (measuring the response).

2.2 Decision-Making Process in Self-Adaptive Software

According to the definition given in [12], the decision-making process for selecting an adaptation strategy in a self-adaptive system based on quality attributes can be defined as a set $<R,G,D,U>$, where *Request (R)* is the reason for the change being demanded (violation of a quality attribute). *Goal Repository (G)* corresponds to a deposit in the system containing the required quality properties (quality scenarios). *Domain (D)* corresponds to a deposit with structural information and information about the behaviour of the software that is to be adapted. Finally, *Utility (U)* is a repository with information about stakeholders' preferences for carrying out an adaptation.

3 Case Study

This case study is based on a company (InAlpes) that works in the property purchase, sale and rental field, offering its clients houses, apartments and offices.

InAlpes would like to implement a service-based system (SOA). In view of the company's business objectives, a process automation strategy has been proposed, and since it is envisaged that the processes will be used intensively, it is hoped that the availability level will be high. This system is expected to be used on a large scale, and it is therefore not viable to make adaptations to the system on-line. In other words, the system should adapt itself if faults are detected, or problems arise with services. Figure 1 shows that InAlpes has a Service Oriented Architecture (SOA) consisting of a set of services displaying the capacities of existing data sources and applications, to support the business process detailed above. This technology architecture was implemented with the aim of having a set of highly reusable and flexible services available.

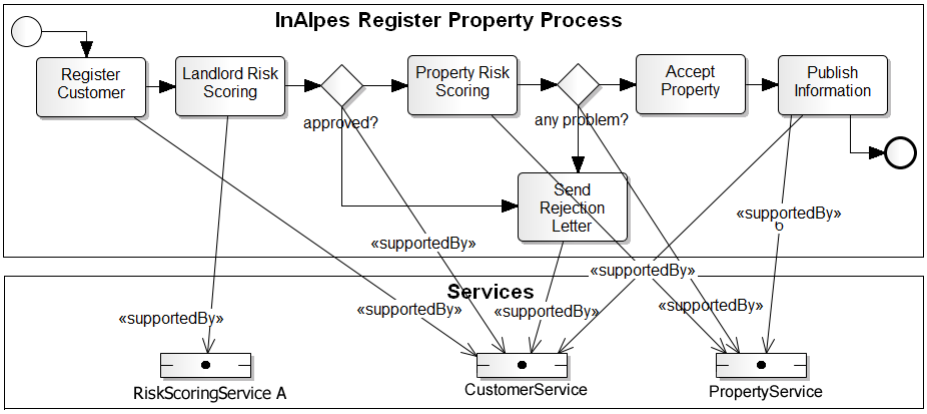

Fig. 1. The InAlpes Register Property Process

The Register Property business process consists of publishing property details, in order to be rented by a customer. We will pay attention to the *Landlord Risk Scoring* activity. In this example, *RiskScoringService A* is responsible for checking the customer's credit history and whether he is blacklisted or not. This activity is critical, and if faults occur, the system should adapt itself in order to find an alternative service, with a view of guaranteeing the uninterrupted operation of the system. In the example, *RiskScoringService B*, *RiskScoringService C*, and *RiskScoringService D* all offer the necessary functionality, but with different quality characteristics.

4 MENTA: A Quality Analysis Approach for Self-Adapting Systems

MENTA is a method based on model-oriented engineering principles for supporting the design of self-adapting SOA systems. The purpose of this proposal is

to create a pool of services that offer the same functionality, the system finds a service that not only meets the required quality scenarios, but also has the best quality of service from this set of services. In general terms, the process proposed by MENTA consists of three steps. The first defines the architecture models to be adapted and the quality scenarios to be accomplished. The second defines the rules for adapting the system. The third creates a model relating the system to the adaptation rules and generates the code for the self-adapting system.

4.1 First Step - Modelamiento Arquitectural y Comportamental

Modeling the Domain (D) Repository. This repository contains information about the system to be adapted. In our case, this repository contains the models which represent the services architecture of the solution. The service architecture representation is based on Archivol [3], a meta-model for defining solution architectures. By using Archivol, relationships can be established between the elements of an architectural style and the associated functional requirements. In this model the relationships between the consumer *Landlord Risk Scoring* and the provider *RiskScoringService A* are defined. The service provider is modeled as a component that exposes an interface. This interface has a set of operations, which serve to identify the features you want to offer to the service consumers. This interface has two major goals in this approach: Define the operations exposing the service and serve as an identifier for the set of alternative services.

Modeling the Goal Repository (G). This repository contains the information of the quality properties (quality scenarios) required in the system. Each scenario has a specific requirement for a quality attribute and its validation will depend on the quality properties of the services or components that make part of the application. Quality scenarios will be described in Archivol [3]. A quality attribute scenario model comprises a *performanceScenario* quality requirement. This quality requirement depends on the quality attribute *Response Time*. In this quality attribute scenario two interactions are specified, *Stimulus* and *Response*. Stimulus interaction is of type input, i.e., the event triggering the scenario evaluation. The metrics for this interaction are called *Request*, and it is a measure of the invocation. Moreover, the Response interaction is output type, i.e., what is expected to assess the response of this interaction. In this case, the answer given is expected to be between 0 and 50 milliseconds. Just as this was defined, other quality requirements may be defined as well.

4.2 Second Step - Definition of Restrictions

Once the architecture of the system has been modelled (Domain Repository and Goal Repository), the architect needs to turn his attention to the desirable

behaviour of the self-adapting system. He has to think of the quality scenarios he wants to control for each consumer-provider relationship. This relationship can be expressed by rules. When a rule has been broken, the action that is taken involves locating a new service which offers the same functionality and also accomplish the quality scenarios that have been modelled. In the context of the model defined by [12], adaptation rules will use the information stored in the Goal Repository (G).

To establish the adaptation rules, we defined a Domain-Specific Language (DSL) called *MENTADSL*, which allows an association to be specified between a consumer service, a service provider interface, and the envisaged quality characteristics when the service is being executed. The motivation for the DSL design is based on ECA rules, with a specialisation for considering quality scenarios when conditions are being evaluated. The set of rules described for defining the self-adaptable behaviour of the system are transformed into a model.

Listing 1.1. Fragmento de Self-Adaptability Definition Language

```
1 rule { rule1ForLandlordRiskScoring }
2   for serviceConsumer { LandlordRiskScoring }
3   consuming interface { iCustomer }
4   fulfilling qualityScenarios { performanceScenario,
5     availabilityScenario };
```

The Listing 1.1 gives a DSL example. On line 1, a rule is created and a name assigned. Line 2 specifies who will be the service consumer to be controlled. Following the case study, the consumer will be the business activity *Landlord Risk Scoring*. Line 3 associates the interface to be used to identify the relationship between a service consumer and any alternative services that best behave. Lines 4 and 5 specifies the quality attribute scenarios you want to validate for all alternative services that offer the interface in line 3. In this fragment, the services are required to meet *availabilityScenario* and *performanceScenario* scenarios.

Modeling the Utility (U) Repository. The Utility Repository (U) contains information relating to adaptation-objective preferences. According to the DSL, when a failure on the quality attribute scenarios is detected, *Alternative Services* are evaluated in order to replace the service where the fault has occurred. This selection is made with a set of services implementing the same functional contract of the original service. Our approach gives priority, when an alternative service is being selected, to the greatest number of quality scenarios that the particular alternative service complies with. If faced with two alternative services that comply with the same number of quality scenarios, selection will be based on the order in which the service was defined, prioritizing the older ones.

The failure to accomplish the quality scenarios of a service is calculated by comparing the quality service metrics against defined quality constraints on quality scenarios. When the value of the response time for an operation is greater

than defined in the quality scenario, for example, when the service response time is greater than 50 milliseconds.

4.3 Third Step - Implementing The Self-Adaptive System

The self-adapting system introduces one element that is fundamental to facilitating system self-adaptation, namely *Virtual Service (VS)*. Each VS element is implemented in the form of dual components, called *Membrane* and *Validator*. The Membrane is generated as a web service that has two objectives. The first is to deal with service user requests, while the second is to ask the Validator component for the reference of a specific service to use in order to answer to a request. When the Validator component returns the service reference, the Membrane is responsible for routing the request to this service through HTTP. Thus, becoming transparent to the consumer on the service he is using. The Validator component is responsible for selecting the service which best adheres to the quality scenarios at a given moment and besides, it best meets those requirements from the set of alternative services. To execute the service selection, the Validator saves information about the status of each available service that implements a particular business interface. The Validator performs a periodic scan of each service in order to determine their respective statuses - for example, if the service is alive and responds within the time limits established. The Validator allows the inclusion of a new service that implements a business interface. This business interface will define the operations required to support a feature. Thus, all alternative services must offer at least the operations described in the interface. This verification is done through signing the methods. During execution, it may add new services on condition that they provide at least the operations in the interface.

Figure 2 gives an example of how our Virtual Service works. In this case, the *LandLord Risk Scoring* activity invokes a VirtualService. The VS selects the most appropriate service in that moment (RiskScoringService C), and connects the activity request with this service. Suppose that the *RiskScoringService C*, doesn't respond with the latency constraint defined in some quality scenario, in this case, the VS selects another service when a new request arrives. The selection is made following the criteria defined in the Utility Repository. It is important to note that the VS does not suspend the current requests. If a violation of a quality scenario is detected while carrying out the processing of a request, it is left to finish, and will be followed by requests using the new service has been selected by the Validator component.

5 Experimentation

For the development of our proposal we used the case study presented in the Section 3, in which attention is focused on the *Landlord Risk Scoring* business

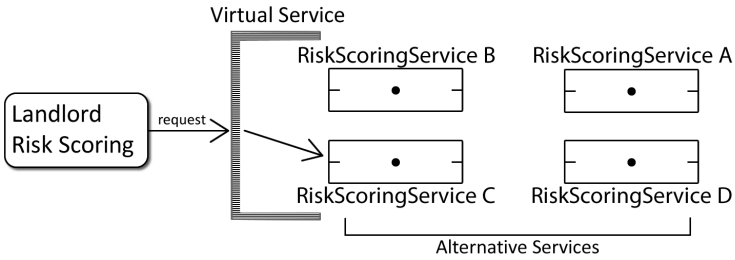

Fig. 2. Menta Operation

activity. The first step in our experiment was to model the business process using Archivol, along with the specification of a quality scenario. For our validation only the activity *Landlord Risk Scoring* was identified as self-adaptive. The second step, modelling the adaptation rules, was developed using the DSL described in Subsection 4.2. After the first two steps, we proceeded to build the self-adaptive system model and its subsequent transformation into code.

This experiment was intended to demonstrate that VS is able to maintain a quality requirement, embodied in the form of a quality scenario. The quality attribute scenario is associated with the attribute *response time* and requires the attribute value to not exceed 50 milliseconds. This can be seen in Listing 1.1. The outcome of this experimentation must show two things: first, keep the response time less than 50 milliseconds, and secondly, to do so, different alternative services shall be used. The latter is related to the fact that the VS located not only the service that meets the quality scenarios described in the rules, but also looks the best among those that satisfy the scenario. The evaluation consisted of a process executed in Bonita BPM process engine. Where we created two instances, and for each instance 50 requests are launched. The 50 requests are 50 runs of the process, resulting in 100 invocations over the VS which is responsible for administoring the alternative services for *Landlord Risk Scoring* business activity. The result can be seen in Figure 3.

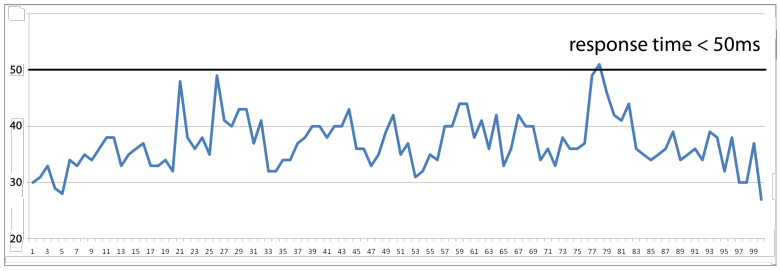

Fig. 3. Response times vs requests

However, an external mechanism to handle self-adaptation introduces some latency. To verify this, we performed a test that consisted of measuring the time from the client perspective. Invocations passing through the VS and the invocation of a service directly. The test consisted of four groups of 10 each invocation. At the end, we found that in general the latency introduced by our proposal is 44.47 milliseconds.

6 Related Work

In [13] a method based on quality attribute scenarios to find and analyse potential points of self-adaptation in software architecture during the design stage is proposed. Extend an ADL called ABC / ADL, to store the architecture information. Information is used directly by a reflective-based middleware architecture, called PKUAS, for making self-adjustments in implementation. Some limitations in this proposal are using EJB components. While the use of these components is not really a limitation at the functionality level, it certainly is at a interoperability and scalability one. In a word, this proposal is tied to implementation of technology. This proposal is not clear about the possibilities of including new components. The system design is separate from the implementation, so the implementation may be different from the solutions proposed in the architecture. In practice this should not happen, but it is a risk taken to manage this separation.

In [9] they focus on non-functional reconfigurations in a SOA system. These reconfigurations refer to the action of replacing some of the services with others with better quality characteristics. From a restriction of Quality of Service (QoS), this approach seeks to replace individual services according to the their critical factors. If a replacement for an individual service is not able to find any, then it replaces multiple services until they find a satisfactory solution. Some limitations in this proposal are Algorithm resolutions. The current proposal focuses on their algorithms, however, they fail to specify the entire implementation process. The proposal does not give details about the mechanisms or tools used for the architecture modelling process. Neither does it talk about the implementation process. Retaining the proposal at the architectural level, but leaving out elements of the implementation could lead to differences such as a lack of mechanisms for adding new services. If the proposal is not clear about the possibilities of including new services. A self-adaptive system should have the ability to add or remove services running. No mechanisms are defined to express the quality constraints.

7 Conclusions and Future Work

In this article we have introduced a way of modelling a self-adaptable system in a service-based architecture. The contribution our approach makes is to work with models, which not only allows for independence from the platform but also

for quality attributes to be taken into account in order to carry out dynamic adaptations and generate the source code therefor. To achieve this, the architect has to model his current architecture, then describe the self-adaptation rules that will govern the new system, based on quality scenarios. The architect uses a DSL for this that is provided by the proposal, meaning that he does not need to know how to model the self-adaptability, rather just how to describe it. Finally, with the information provided by the architect, the system proceeds to generate the new system with self-adaptive characteristics. In the future, we will consider including the Certain Attributes [7] in order to get a complete self-adaptation system that considers all the qualities of service attributes.

References

1. An approach to grid resource selection and fault management based on eca rules. Future Generation Computer Systems 24(4), 296 – 316 (2008)
2. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice, Second Edition. Addison Wesley (2003)
3. Correal, D.: Model oriented software architectures (2011), `http://moosas.uniandes.edu.co`
4. DARPA: Self adaptive software. DARPA, BAA 98-12, Proposer Information Pamphlet (December, 1997)
5. Diamadopoulou, V., Makris, C., Panagis, Y., Sakkopoulos, E.: Techniques to support Web Service selection and consumption with QoS characteristics. JOURNAL OF NETWORK AND COMPUTER APPLICATIONS 31(2), 108–130 (APR 2008)
6. Garlan, D., Schmerl, B.: Model-based adaptation for self-healing systems. In: Proceedings of the first workshop on Self-healing systems. pp. 27–32. WOSS '02, ACM, New York, NY, USA (2002)
7. Kulnarattana, L., Rongviriyapanish, S.: A client perceived qos model for web services selection. In: Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, 2009. ECTI-CON 2009. 6th International Conference on. vol. 02, pp. 731 –734 (may 2009)
8. Laboratories, S.N.: Jess, the rule engine for the java^{TM}platform, `http://http://www.jessrules.com/`
9. Li, Y., Zhang, X., Yin, Y., Wu, J.: Qos-driven dynamic reconfiguration of the soa based software. Service Sciences, International Conference on 0, 99–104 (2010)
10. Lin, K.J., Zhang, J., Zhai, Y., Xu, B.: The design and implementation of service process reconfiguration with end-to-end qos constraints in soa. Service Oriented Computing and Applications 4, 157–168 (2010), `http://dx.doi.org/10.1007/s11761-010-0063-6`, 10.1007/s11761-010-0063-6
11. Oriol Hilari, M., Marco Gomez, J., Franch Gutirrez, J., Ameller, D.: Monitoring adaptable soa-systems using salmon. pp. 19–28. GESSI - Software Engineering for Information Systems Group, Madrid, Espaa (2008)
12. Salehie, M., Tahvildari, L.: A quality-driven approach to enable decision-making in self-adaptive software. In: Companion to the proceedings of the 29th International Conference on Software Engineering. pp. 103–104. ICSE COMPANION '07 (2007)
13. Zhu, Y., Huang, G., Mei, H.: Quality attribute scenario based architectural modeling for self-adaptation supported by architecture-based reflective middleware. In: Software Engineering Conference, 2004. 11th Asia-Pacific. pp. 2 – 9 (2004)