A Reference Architecture for Component-Based Self-Adaptive Software Systems

Lorena Castañeda, Gabriel Tamura

Department of Information and Communication Technologies
Icesi University, Colombia
{lcastane,gtamura}@icesi.edu.co

Abstract. Self-adaptive systems are those able to change structure and behavior dynamically and during runtime to overcome unpredictable changes in the context. Software engineers have made significant efforts to propose self-adaptive systems using different approaches based on feedback loops, components and even hybrid architectures, and communication standards. Often, these proposals merge self-adaptation with the target system making very difficult to analyze the adaptation capability. In this paper we describe our design and evaluation of a component-based architecture to implement the basic elements required for self-adaptation: *monitoring, analyzing, planning* and *executing*. Our architecture provides separation of concerns and extensible capabilities in such a way it can be extended to be implemented.

Keywords: Self-adaptive Systems, Reference Architecture, Feedback Loops, Component-Based Software

1 Introduction

Nowadays, businesses based on software systems experiment diverse changes in their requirements of operation. However, along with these organizations, the software systems whose purpose was to support the organization's stability is now required to be flexible enough to support the changes and therefore, to be able to adapt in order to meet the new business goals [1].

For computing systems, the context is unpredictable and it is nearly impossible for software engineers to design an error free system. Therefore, engineers should also monitor and analyze the system every time searching for possible faults. Subsequently, most of the IT budget is spent in prevention and recovery from failures [2].

As a result, it becomes necessary to build systems that can be more self-manageable and must be able to adapt some part of itself. For this adaptation to be performed by the system itself, it is required the execution of four tasks: **monitoring** the new business requirements, as the internal and the external context, **analyzing** the data gathered to identify symptoms and need for adaptation, **planning** a set of adaptive actions, and **executing** those actions over the system in execution [3, 2, 4, 5].

One of the most important challenges in self-adaptation is to create the ability in the system to reason about itself and its context. Control engineering uses feedback to measure and act upon the behavior of the system. Moreover, *feedback loops* become important in the process that the system must follow to monitor and analyze a desired property of the system and then evaluate if it is being reached [6]. Software engineers have made significant efforts to propose self-adaptive systems. These proposals have used different approaches based on feedback loops, components and even hybrid architectures, and communication standards [7, 8, 6, 3, 9–14]. However, all these approaches are not sufficient to fit the different systems, data types, and configurations that are available to build self-adaptive software systems. Often, these proposals merge self-adaptation with the target system making very difficult to analyze the adaptation capability as an independent function, and feedback loops are not explicit in most of these proposals [8, 6, 3, 9–11]. In this scenario, a reference architecture for designing self-adaptive software based on feedback loops is a necessity.

In this paper we design a component-based architecture to implement the basic elements required for self-adaptation: *monitoring, analyzing, planning* and *executing*, independently from the system that is being adapted. Moreover, the components of this architecture can be distributed among different machines providing flexibility and extensibility. Our architecture will provide extensible capabilities in such a way it can be extended to be implemented as in e.g., the Service-Oriented paradigm. To evaluate our proposal, we implement the architecture and evaluate it in an specific case of study.

The remaining of this document is organized as follow: Section 2 describes the context and background of this project, a brief overview of self-adaptive software systems as well as the methodology to derive a reference architecture, and the definition of a component-based software. Section 3 summarizes the two relevant approaches of this project, the IBM's ACRA and DYNAMICO. Section 4 describes de elements to derive our reference architecture and our proposal for the data flow architecture. Section 5 presents our *Reference Architecture for Component-Based Self-Adaptive Systems* with a brief description of its elements. Section 6 presents the evaluation of our proposal in an specific case of study. Finally, Section 7 presents the conclusions of this work.

2 Context and Background

Adaptation refers to the capability of changing the software structure or behavior according to significant alterations in the environment. This adaptation must occur dynamically and at runtime [15, 3]. **Self-adaptive software systems** rely completely on the *context* which can be defined as any information that characterizes the state of entities that affect the execution, maintenance and evolution of systems and where al changes occur [13]. Therefore, the desired software adaptation its meant to be dynamic to adjust itself at runtime, and requires the system to be context-aware to respond to certain changes. In fact, a *component-based design* provides a advantages for composition suitable for adaptive software solutions.

The difference between a **software component** and any other software element is the use given by components in the matter of quality features [16]. Moreover, software components hide all the complexity of the software implementation by only exposing the services that the component requires and provides through its interfaces. This easy way of assembling components allows software engineers to reuse, modify or even extend software without the difficulty of development maintenance because as software components are independent from each other, they can be easily replaced [17].

For this proposal, a separation between the adaptation mechanism and the software system is required in order to analyze and maintain each system as its own. Component-based software allow separation but also connection among components through interfaces. Also, a component-based design will provide flexibility and extensibility and the possibility of operation between software systems running on different platforms and written possibly in different programming languages. Hence, our *reference architecture* is in the scope of component-based software systems.

According to Len Bass *et al.* an **architecture in software** defines the elements that compose the system and the relationship between those elements, and it hides the information that is not relevant for the interaction among them [18]. The individual behavior of each element is an important part of the architecture which allows the appropriate communication between them.

The design of a reference architecture follows different stages where each stage represent architectural decisions and architectural binding choices. Fig. 1 shows the relationship among the design elements needed to build a reference architecture.

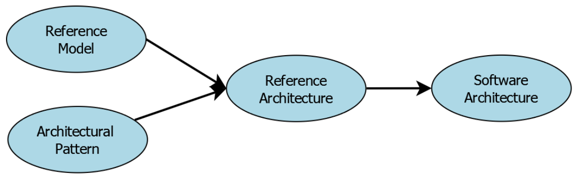

Fig. 1. Reference Architecture derivation relationship between reference model, architectural patterns, reference architectures, and software architectures. (The arrows indicates that subsequent concepts contains more design elements)[18]

The *reference model* represents the decomposition of a well known type of problem in several parts that work together to solve the problem [18]. The *architectural patterns* are a set of elements in component types and the relationships among them and exhibits known quality attributes such as performance, security or availability. The main difference between an architectural style and a reference model, is that the second one is well-known and accepted by a community. Finally, the *reference architecture* is a mapping between a reference model and a set of software elements with the *data flows* among them. Each element in the reference architecture implements totally or partially a function previously identified in the reference model[18].

3 Self-Adaptive Software System Approaches

Some approaches related to self-adaptive software systems based on feedback loops, components, and even hybrid architectures, and communication standards that ground some of the structural and behavioral basis of this proposal [8, 6, 3, 9–11]. However, we'll review two of them which components are important to address our proposal.

3.1 Autonomic Computing Reference Architecture (ACRA)

IBM proposed an architectural blueprint for autonomic computing [7]. According to this proposal, the management of a system involves four common activities: *collect information, analyze it to determine actions that need to be taken, create a plan with those actions*, and finally, *execute the plan.* The IBM's proposal presents a five layer scheme to control self-management systems. However, the most relevant aspects in this proposal for our project are in layers 2 and 3 because include important aspects in the self-adaptation mechanism [7].

The Layer 2 (or *Touchpoints*) are the components responsible to interact with the managed resources, that is the awareness of the state and management of the resource trough two components: *sensor and effector.* The Layer 3 (or *Touchpoint autonomic managers*) are components that implement the *intelligent control loop* over the managed resources, also known as the MAPE-K-loop: Monitor, Analyzer, Planner, and Executer . These four parts communicate and collaborate to each other and exchange appropriate knowledge and information.

Fig. 2 represents these elements. The Monitor collects the information from the managed resources. The Analyzer observes the reported situation, and determines if any change needs to be made. The Planner creates the plan to achieve the changes, and the Executer provides the mechanism to perform the changes over the managed system. The common part of the MAPE-K loop, is the knowledge source, which is shared data among the autonomic managers in the system. In order to do that, the knowledge must by expressed using common syntax and semantic to the system. There are at least three ways to specify knowledge: through policies, activities inside the autonomic system or actions inside the autonomic control loop.

3.2 DYNAMICO: A Reference Model for Context-Based Self-Adaptation

As mentioned previously, self-adaptive systems decide autonomously about the changes the system needs to perform. In order to take those decisions, the sys-

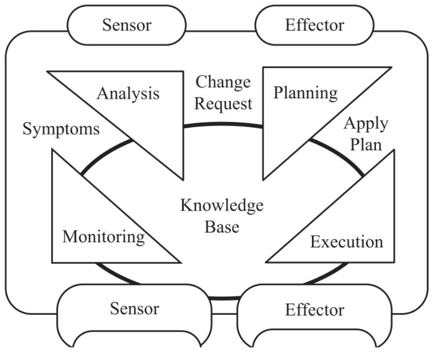

Fig. 2. Functional arrangements of the Autonomic manager (ACRA) [7]

tem must know the output of its actions using feedback loops [15, 19, 12]. The reference model for context-based self-adaptation by Villegas *et al.* illustrated in Fig. 3 is based on the SISO feedback control with explicit functional elements and corresponding interactions to control dynamic adaptation.

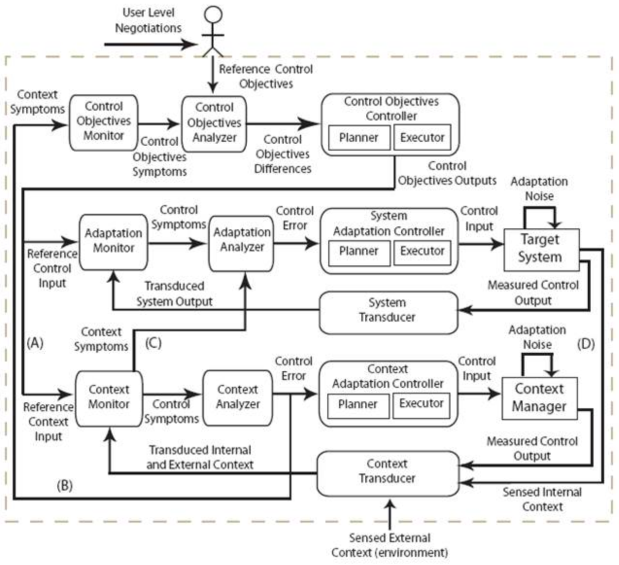

Fig. 3. Reference model for context-based self-adaptive systems [12]

The separation of concerns is a key aspect in this model, which defines three subsystems to achieve self-adaptation [12]: (1) The *control objective manager* that manages the target system's purpose in terms of its control objectives, according to the policies given by administrators. (2) The *context manager* that is responsible for maintaining the pertinence and relevance of the context monitoring infrastructure with respect to the target system under changing conditions of execution. And (3) the *adaptation mechanism* that is responsible for the adaptive actions over the target system according to the evaluation of its behavior. That means, to guarantee the accomplishment of the target system's purpose to satisfy its quality attributes.

Of particular importance in this reference model are the interactions between the three loops. Even though they are designed independently they require to operate together to achieve the system objectives.

4 A Reference Architecture for Component-Based Self-Adaptive Software Systems

As mentioned in Section 2, a reference architecture is the result of the mapping between a reference model and the software elements of a given design and corresponding data flow.

Reference model
The case of study of this project is based on the running example presented by Tamura *et al.* [20] called the Reliable videoconference System (RVCS). In summary, this RVCS is a self-adaptive software system with a server that hosts videoconferences and mobile users which register and attend the conferences.

In this scenario two software elements interact: an *end-user application* in a mobile device and a *server application*. The main three functionalities of the system to be modeled are: (1) the end-user application allows the user to register in the videoconference system, (2) the application allows the user attend to a particular videoconference in execution, and (3) a server application in the videoconference system establishes the service conditions, also known as SLO[1], with the end-user application. The end-user application itself also establishes the conditions with the server, that is why the arrow is bi-directional.

Architectural style
For this reference architecture, three of the David Garlan and Mary Shaw's common architectural styles [21] are reused: *(i)* Pipes and Filters (we have the necessity to transform data from one component to another), *(ii)* Event-based, (we need to execute and/or trigger event over some components such as the target system and the sensors and effectors), and *(iii)* Blackboard (the knowledge base interactions).

Architecture data flow
In our case of study we need to exchange data information among the elements regarded to the adaptation requirement. For this reference architecture's data

[1] SLO (Service Level Objective) is an element of the SLA (Service Level Agreement) between a service provider and a service consumer. The SLO measures the achievement of a quality attribute of the service.

flow, we propose a context model based on the context metamodel proposed by Villegas *et al.* [13]. For our reference architecture, not all the metaclasses of that context metamodel are necessary, excludes the acquisition mechanism for the context and the provisioning mechanism given the scope of this proposal.

As noticed in the IBM's proposal, the MAPE-K components are connected to each other as a mechanism for adaptation. Each component receives information about the context, process it and deliver it to the next components in the loop. In our proposal three data types are specified: *(1)* The **context data** is the result of the Sensor component activity. This type of data carries the information of the context events being monitored. *(2)*The **context event information** represents the current state of a context entity according the SLO terms by the performance of a set of operations. And *(3)* the **diagnosis** represents the relationship between two context entities: the context entity in its *current state* and the same context entity in its *desired state*[2].

5 Component-Based Reference Architecture Proposal

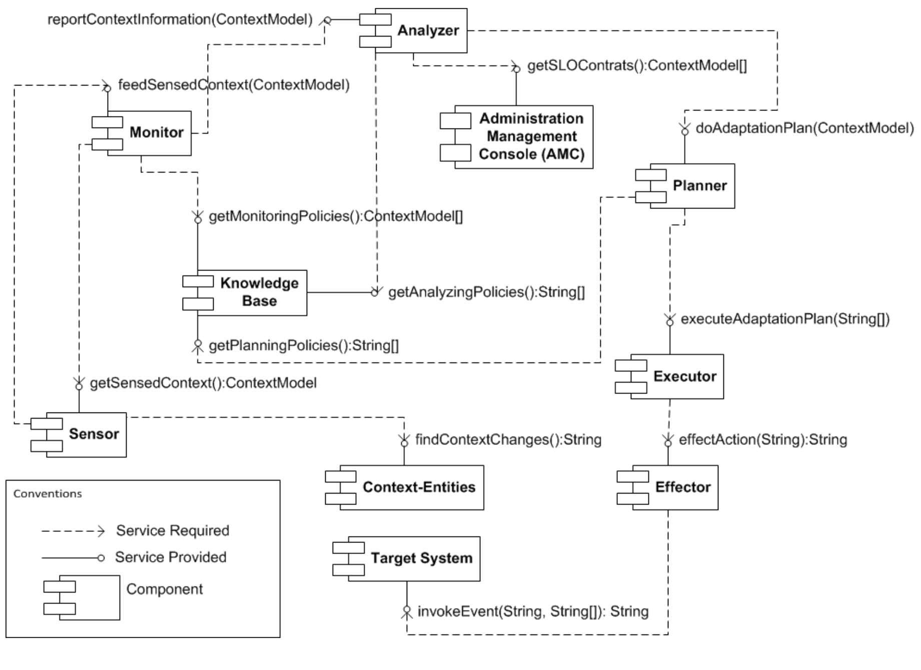
reportContextInformation(ContextModel)
Analyzer
getSLOContrats():ContextModel[]
feedSensedContext(ContextModel)
Monitor
Administration Management Console (AMC)
doAdaptationPlan(ContextModel)
Planner
getMonitoringPolicies():ContextModel[]
Knowledge Base
getAnalyzingPolicies():String[]
executeAdaptationPlan(String[])
getPlanningPolicies():String[]
Executor
getSensedContext():ContextModel
Sensor
findContextChanges():String
effectAction(String):String
Context-Entities
Effector
Conventions
Service Required
Service Provided
Component
Target System
invokeEvent(String, String[]): String

Fig. 4. Component-based reference architecture for self-adaptive systems

[2] See the author's website of the project [22] for the complete details of the Data Flow proposal (i.e., the data types definitions, the model derived, the class diagrams and the Java implementation)

Fig. 4 presents our reference architecture in which the MAPE-K loop is included based on the original IBM's proposal. The components of the architecture are described as follows:

- **Target System:** It is the system we intend to adapt and must be a component-based software system executed in a component runtime system. The Target System in our reference architecture is represented as one component, however, this does not imply that is smaller or less complex than the adaptation loop.
- **Context-Entities:** These components represent the context monitored and whose behavior will affect the target system determining its adaptation. In some cases, the Context Component and the Target System Component are the same, because the system itself is its own context.
- **Administrator Management Console (AMC):** This component allows system administrators to configure the SLOs. This component follows a SLO's general definition of the *Quality of Service (QoS) Contracts* based on the e-graph representation proposed by Tamura *et al.* [23].
- **Sensor:** This component (also known as Monitor Probe) provides a representation of the target system's state in a specific variable of interest and other external systems information, in the form of context entities needed for the adaptation mechanism.
- **Monitor:** This element is responsible to filter all relevant sensed context information. To filter this data, it must configure the *Monitoring Policies* provided by the *Knowledge Base Component.*
- **Analyzer:** This element is responsible to determine the achievement of the SLO, and to decide whether an adaptation is required. To perform this evaluation it must configure the *Analyzing Policies* provided by the *Knowledge Base Component.*
- **Planner:** This element is responsible to build a plan with all the *adaptation actions.* To build this plan, it must follow the *Planning Policies* provided by the *Knowledge Base Component.*
- **Executer:** It is responsible to execute the adaptation plan by translating to commands every step of the plan and guarantee the correct execution over the target system.
- **Effector:** This element is configured to effect the changes needed to alter the target system's structure or behavior according to the adaptation needs.
- **Knowledge Base:** It provisions information, such as policies, for other components. Also it can be used to manage the historical behavior of this adaptation mechanism during time, but this application is out of the scope of this project.

6 Reference Architecture Evaluation

The goal of this evaluation is to determine the practical feasibility of our reference architecture and explore its extensibility, in this case to support the dynamic adaptation of SCA *(Service Component Architecture)* Software applications [24, 25].

Indeed the intention of our proposal is the extension and distribution capability and the contribution for the software engineering community, in which case an open source platform seems more appropriate than a licensed one. In order to support this initiative we chose Java J2SE *(Java Standard Edition)* platform because even though J2EE is meant to be for component-based software development it does not follow an standard for component-based software.

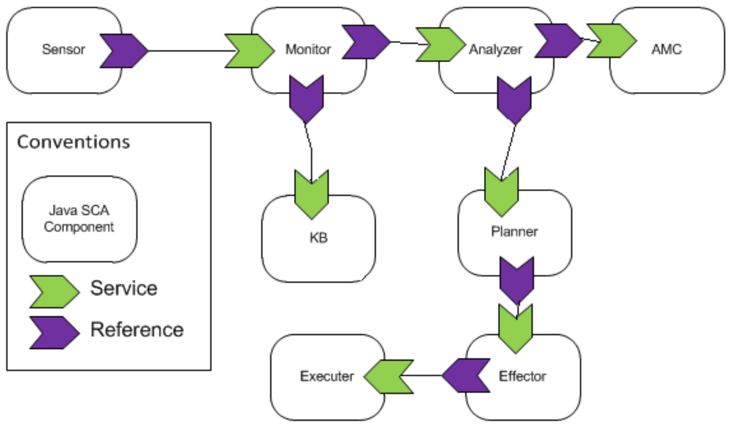

Fig. 5. An instantiation of our reference architecture in terms of SCA Components

Fig. 5 shows an instantiation of our reference architecture in terms of SCA components for a specific implementation. In this figure services *(Green chevrons)* and references *(Purple chevrons)* are explicit between components. It is important to note that not all the services provided and required in the reference architecture are described in this implementation as SCA services and references, the reason is because of the scope of this evaluation. In this case, for example there are no planning or analyzing policies, which means that this implementation uses an static evaluation of the SLO Contracts (Analyzer component) and there is no Adaptation Plan from the Planner component. In addition, for this implementation, each component of the reference architecture will be described by its own composite file to fulfill the extensible capability and distribution goals that were defined for this reference architecture. In our evaluation the components are deployed using an Open Source middleware platform for SCA components named FraSCAti implemented by the OW2 Consortium[3]. The complete set of composite files are available in the website of the project [22].

[3] FraSCAti official web site: http://wiki.ow2.org/FraSCAti/Wiki.jsp?page=FraSCAti

6.1 Java Implementation

Fig. 6 describes the general implementation of the components, which is based on two packages (i) *component.api* where the Java service Interface is defined with the services that the component provides and an Abstract class is defined with the basic implementation for the methods, services and references from other components in order to guarantee an extensible capability for our reference architecture, and (ii) *component.lib* where the Java implementation classes are defined with the specific behavior required by this case of study. As mentioned before, this implementation is meant to be extensible and the interfaces and abstract classes are sufficient to provide this capability. The complete Java specification and implementation is available in the website of the project [22].

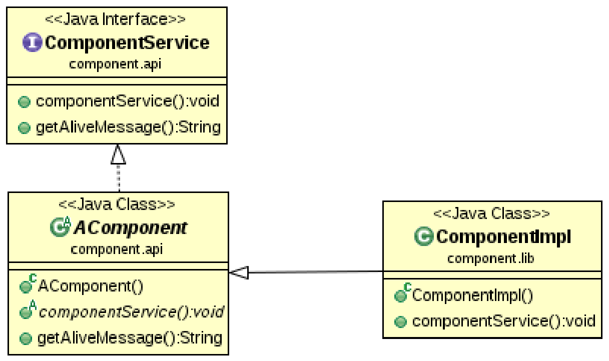

Fig. 6. The general Java implementation classes and packages for the MAPE-K components

6.2 Application to the Case of Study

This case of study is based on the running example presented by Tamura *et al.* which uses a simplified version of a Reliable Video-Conference System (RVCS) that we use to illustrate the requirements for dynamic reconfiguration [20].

The RVCS comprises two software applications in which self-adaptation takes place. Both applications establish the **Quality-of-Service (QoS) conditions** to be fulfilled by the videoconference: (1) Confidentiality, (2) Availability and (3) Throughput. In this case of study, we guarantee the **Throughput** quality attribute in the RVCS Server application in which the number of registration requests are computed as transactions per minute and the maximum limit that the server is capable to attend is delimited by the day of the month in the SLO Contract (for this implementation it was defined a representation of the SLO Contracts as Java Objects, the complete details are available in the website of the project [22]).

Table 1 summarizes for this contract the context events and the service level objectives be fulfilled (SLO). For this case a Context Event is the day of the month and the SLO is the maximum of transactions per minute allowed for the server. In the case in which this maximum is surpassed the adaptation mechanism must reconfigure the RVCS Server, otherwise the adaptation mechanism performs no operation.

Table 1. QoS contractual conditions and corresponding SLO for Throughput as presented by Tamura *et al.* [20]

Context Events	**SLO**
Day of the month	Transactions per minute
1-15	100
16-23	150
24-27	300
28-31	900

As mentioned previously, it is necessary that the adaptation mechanism must be separated from the system itself. The deployment configuration includes three applications separately executed: *(i)* The RVCS, *(ii)* the Adaptation Mechanism (AM), and *(iii)* the SoapUI Tester used to simulate RVCS clients[4]. The scope of this test does not provide the adaptation instructions but just a message indicating whether adaptation is required and some dummy instructions to guarantee the data flow through the Planner, Executer and Effector components. The adaptation Plan and Execution requires a more complex and specific work.

6.3 Test Cases

For this configuration the MAPE-K monitors the SLO Throughput Contract. The general test case is described in Table 2 in which the **input** is the Day of the month *(d)* and the quantity of transactions received in the last minute *(x)*, the **evaluation** is the match between input with the current day range and the maximum transactions per minuted allowed, and the **output** depends on the result of the evaluation that can be either "Adaptation Required" or "No adaptation required". For this evaluation, Adaptation is required when x is greater than the maximum allowed.

6.4 Test Results

In this test implementation, SoapUI stresses the RVCS system by creating a number of hypothetical attendants previously created in the RVCS system and all these attendant clients will register to all the conferences available in the

[4] Refer to the website of the project for the complete implementation of the SOAP UI and the configuration files [22]

Table 2. General conditions for the Test Cases for the MAPE-K evaluation corresponding the SLO for Throughput

Input Context Information	**Evaluation** SLO condition	**Expected Output** MAPE-K Response
Day:d, Tx/min:x	$d \in$ Range. If $x <$Max Tx/min	No adaptation Required
Day:d, Tx/min:x	$d \in$ Range. If $x =$ Max Tx/min	No adaptation Required
Day:d, Tx/min:x	$d \in$ Range. If $x >$Max Tx/min	Adaptation Required

system. While the RVCS is running and attending these requests, the MAPE-K is also running and querying every minute for this type of requests registered in the database table. Table 3 describes the information and conditions during the test of the MAPE-K in this RVCS scenario. For this implementation RVCS applications and MAPE-K are running in the same machine. However, RVCS distribution does not affect the test because the context information is in the RVCS database.

Table 3. Contract Information for the Test Case scenario

	Test No.				
Information	**1**	**2**	**3**	**4**	**5**
Day of the month	15			16	
Conferences in the RVCS System	7				
Max Tx/min SLO	100			150	
Test Tx/min *(tx)*	tx = 0	tx <= 100	tx >100	100 <tx <= 150	tx >150
RVCS deployment	Single Machine				
MAPE-K deployment	Single Machine				

Table 4 presents the results of the 5 tests cases performed over the described scenario in which the MAPE-K responses are expected showing the expected behavior of the MAPE-K's functionality (More information about the output for some of the executed tests in the website of the project [22]).

Table 4. Test results for the MAPE-K according the SLO for Throughput.(Conventions: T=Number of threads, RR=RVCS Registration requests, E=Evaluation result, L=Less, G=Greater, NA= No adaptation, A=Adaptation)

T	**SOAPUI Info**	**SLO Info**	**E**	**Output**
1	T = 0, RR = 0	Nothing to evaluate		
2	T = 10 , RR = 70	15 $\in$ Range:1-15 $\rightarrow$ Tx/min:100	L	NA
3	T = 20, RR = 140	15 $\in$ Range:1-15 $\rightarrow$ Tx/min:100	G	A
4	T = 20 ,RR = 140	16 $\in$ Range:16-23 $\rightarrow$ Tx/min:150	L	NA
5	T = 30 , RR = 210	16 $\in$ Range:16-23 $\rightarrow$ Tx/min:150	G	A

7 Conclusion

In this project we have developed a reference architecture that provides a general framework to implement adaptation mechanisms for software systems. This reference architecture allows to maintain a clear and complete separation of concerns between adaptation mechanisms and target systems, enabling the evaluation, maintenance and analysis of the adaptation process in a self-adaptive system. Moreover, each action in the adaptation process can be analyzed independently as part of a specific component in the reference architecture.

Based on components, this reference architecture also provides component reuse, the ability to be extensible and interoperable, which can help software engineers to specialize, extend and grow the functionalities of each component, according to the necessities of the specific target system. Moreover these components can be reused independently, as far as the target architecture provides the services required by them, which multiplies the reuse possibilities beyond the scenarios considered in this project.

As an SCA-compliant implementation, our reference architecture fulfills the goal of being extensible, which is one of the major goals of this project. Also, each SCA component of the reference architecture is defined in its own composite file in order to fulfill the distribution capability, which is the second major goal of this project.

Building a reference architecture for self-adaptive systems was one of the challenges proposed by Villegas et al. [12]. However, in this project we have identified an additional set of challenges worth of future work, such as the following:

- The need of sensors being embedded in the target system to analyze its behavior and then report to the MAPE-K about any change in the context.
- The need of security between the Target System and the MAPE-K, as two separated systems, to be sure that this external component is reliable and can be trusted.
- The implementation of the AMC to translate SLO contracts from human language to objects requires a specific software implementation suitable for any user in charge of this task in the software system.
- A quality assurance mechanism, as mentioned by Brun *et al.* [5] and Tamura *et al.* [23] to ensure that after adaptation the system is left in a valid state and the adaptation process finishes as expected.
- The Planner and Executer implementation. The Planner component tasks include the planning policies and techniques to generate the adaptation plan that eventually will modify the behavior of the target system. The Executer must transform the adaptation plan to commands that the Target System interprets to change itself according the new desired state.

References

1. Truex, D., Baskerville, R., Klein, H.: Growing Systems in Emergent Organizations. Communications of the ACM **42 No. 8** (1999) 117 – 123

2. Ganek, A., Corbi, T.: The dawning of the autonomic computing era. IBM Systems journal **42**(1) (2003) 5–18
3. McKinley, P.K., Masoud Sadjadi, S., Cheng, B.H.: Composing Adaptive Software. IEEE Computer (July 2004)
4. Kephart, J., Chess, D.: The vision of autonomic computing. IEEE Computer **36 No.1** (2003) 41 – 50
5. Cheng, B.H., Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Serugendo, G.M., 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. Software engineering for self-adaptive systems **LNCS 5525** (2009) 1 – 26
6. Hellerstein, J.L., Diao, Y., Parekh, S., , Tilbury, D.M.: Feedback Control of Computing Systems. John Wiley and Sons (2004)
7. IBM: An Architectural blueprint for autonomic computing. Autonomic computing (2005)
8. Kramer, J., Magee, J.: Self-managed Systems: an Architectural Challenge. IEEE Computer (2007) 259 – 268
9. Garlan, D., Chen, S.W., Huang, A.C., Schmerl, B., Steekiste, P.: Rainbow: Architecture-based self-adaptation with reusable infrastructure. IEEE Computer **37**(10) (2004) 46–54
10. Parra, C., Blanc, X., Duchien, L.: Context Awareness for Dynamic Service-Oriented Product Lines. In McGregor, J., Muthig, D., eds.: 13th International Software Product Line Conference. Volume 1., San Francisco, États-Unis (August 2009) 131–140 Acceptance rate: 30/83 (36%). Rank (CORE) : B.
11. Solomon, B., Ionescu, D., Litoiu, M., Mihaescu, M.: A real-time adaptive control of autonomic computing environments. CASCON (2007) 124 – 136
12. Villegas, N.M., Tamura, G., Müller, H.A., Duchien, L., Casallas, R.: DYNAMICO: A Reference Model for Governing Control Objectives and Context Relevance in Self-Adaptive Software Systems. In: Software Engineering for Self-Adaptive Systems 2. Volume 7475 of LNCS., Springer (2012) 282 – 310
13. Villegas, N., Müller, H.: Context-Driven Adaptive Monitoring for Supporting SOA Governance. Proceedings of the 4th International Workshop on a Research Agenda for Maintenance and Evolution of Service-Oriented Systems (MESOA 2010) **Carnegie Mellon University Software Engineering Institute** (2010)
14. Tewari, V., Milenkovic, M.: Standards for Autonomic Computing. Intel technology journal **10 No.4** (2006) 275 – 284
15. Müller, H..A., Kienle, H.M., Stege, U.: Autonomic computing: Now you see it, now you dontdesign and evolution of autonomic software systems. Lecture Notes in Computer Science **5413** (2009) 32 – 54
16. Heineman, G., Councill, W.: Component-based software engineering. Addison-wesley (2007)
17. Szyperski, C.: Component Software. Beyond Object-Oriented Programming. Addison-wesley (2007)
18. Bass, L., Clements, P., Kazman, R.: Software Architecture In Practice. Addison-Wesley (2003)
19. Giese, H., Brun, Y., Serugendo, J.D.M., Gacek, C., Kienle, H., Müller, H., Pezzè, M., , Shaw, M.: Engineering self-adaptive systems through feedback loops. Springer-Verlag **LNCS 5525** (2009) 47 – 69

20. Tamura, G., Demarey, C., Casallas, R., Duchien, L.: QoS-CARE: A Framework for self-reliable QoS Contract Preservation through Self-Reconfiguration. Preprint submitted to JSS Special Issue State of the Art in Self-Adaptive Systems (2012)
21. Garlan, D., Shaw, M.: An Introduction to software architecture. School of computer science. Carnegie Mellon University (1994)
22. Castaneda, L.: A Reference Architecture for Component-Based Self-Adaptive Software Systems (Website) http://lcastaneda.com/mgit/.
23. Tamura, G., Casallas, R., Cleve, A., Duchien, L.: QoS contract-aware reconfiguration of component architectures using e-graphs. In: Proceedings of the 7th international conference on Formal Aspects of Component Software. FACS'10, Berlin, Heidelberg, Springer-Verlag (2012) 34–52
24. Organization, O.: Service Component Architecture (SCA) Specification http://www.oasis-opencsa.org/sca (Last visited: Jan/2012).
25. Beisiegel, M., Blohm, H., Booz, D., Edwards, M., Hurley, O.: Service Component Architecture, Assembly Model Specication. Specication Version 1.0. Open Service Oriented Architecture Collaboration (2007)