Integrating Quality Requirements in Engineering Web Service Orchestrations

Tarek Zernadji[a], Chouki Tibermacine[b], Foudil Cherif[c], Amina Zouioueche[a]

[a]*Computer Science Department University of Biskra, Algeria*
[b]*LIRMM, CNRS and Montpellier University, France*
[c]*LESIA Laboratory University of Biskra, Algeria*

Abstract

Today's Web services are considered as one of the leading technologies for implementing components of service-oriented software architectures for desktop, Web or mobile applications. When designing workflows of activities that involve the invocation of these Web Services, we build either orchestrations or choreographies. The engineering of such applications is an emerging research topic with many challenges. Among them, we can stress out the crucial question of how to answer quality requirements in such engineering processes. This paper, presents a method which aims at assisting software architects of Web Service orchestrations in integrating quality requirements in their artifacts. In order to satisfy a quality requirement, this method suggests a list of service-oriented patterns. We base our work on the postulate stating that quality can be implemented through patterns, which can be specified with checkable/processable languages. This method helps architects to reach concrete architecture changes that can be automatically performed on the orchestration in order to apply a pattern, and thus integrate its associated quality. We experimented our method on a set of real-world orchestrations (BPEL processes) to measure the overhead of using it in engineering such service-oriented applications. The obtained results showed that our method brings a significant gain of time.

Keywords: SOA Pattern, Quality attribute, BPEL

Email addresses: `zernadji@yahoo.fr` (Tarek Zernadji), `tibermacin@lirmm.fr` (Chouki Tibermacine), `foud_cherif@yahoo.fr` (Foudil Cherif), `zou.amina@gmail.com` (Amina Zouioueche)

1. Introduction

In the last two decades, (Restful or SOAP-based) Web services have confirmed their status of one of the leading technologies for implementing components of service-oriented software architectures for desktop, Web and even mobile applications. The growing need for choosing such technology is related to: i) the integrability and portability (independence from programming languages, middleware or operating systems) provided by the published services, ii) the ease of use and efficiency of HTTP as a communication protocol with these services, iii) the security brought by the SSL/TLS layer included in HTTPS, among many other "ilities".

When modeling applications that involve the invocation of Web services, we can build two kinds of compositions of Web services: orchestrations or choreographies. Orchestrations include a central workflow process that implements the main business logic of the modelled application, which invokes operations of "partner" Web services. One of the leading languages used for modeling (and even executing) orchestrations is the OASIS standard WS-BPEL or BPEL (Business Process Execution Languauge [1]). In choreographies, Web services are considered as peers that collaborate in order to implement the application's business logic. One possible language that can be used for modeling choreographies is the OMG's standard BPMN (Business Process Model and Notation [2]).

The emergence of such technology and languages is recent. So, the engineering of these service-oriented applications is not yet mature and raises many challenging questions. Among these questions, we can mention the crucial issue of how to satisfy quality requirements in this kind of engineering processes. In this paper, we present a method named *"SAQIM" (Service-oriented Architecture Quality Integration Method)* which aims at providing to software architects of Web service orchestrations[1] an on-demand assistance in integrating quality requirements in their artifacts. This method has been designed as a multi-step process. It introduces a template for enabling architects to describe quality integration "intents". It then analyzes these intents and helps the architect in satisfying them by suggesting some service-oriented patterns. We base our work on the postulate stating that quality can be implemented through the application of patterns [3, 4], which are specified with

[1]We focus in this paper on such Web service compositions and do not deal with choreographies.

processable languages. After that, the method that we propose simulates the application of different alternative patterns that satisfy the targeted quality requirement, and notifies the architect with its consequences on the other implemented qualities. In this way, it helps her/him to choose the best available pattern that meets her/his needs. It automatically handles some steps in this method in order to reach at the end concrete architecture changes, which are specified using a scripting language called *"WS-BScript"* (for Web Service-BPEL Scripting). It is a lightweight DSL that allows specifying primitive changes making possible the reconfiguration of Web service orchestrations. These changes can be automatically processed on the orchestration in order to apply the chosen pattern, and consequently integrate its associated quality. We experimented our method on several real-world orchestrations (BPEL processes) and we measured the overhead of using this method in engineering such service-oriented applications. The obtained results showed that our method brings a significant assistance to architects.

The remainder of this paper is laid out as follows. In Section 2, we illustrate the problems tackled in our work through some examples. These examples are used in the remaining sections to illustrate our proposals. In Section 3, we detail all the steps of the proposed process. In Section 4, we present an experimentation of our method, and discuss the obtained results. Before concluding and presenting some perspectives to our work, we make in Section 5 an overview of the related work.

2. Illustrative Example

The Web service orchestration, implemented by a BPEL process, that we use as an illustrative and running example here represents a Travel Reservation Service (TRS) of a travel agency. The TRS Service[2] is an example of real-life service for travel organization. This system enables the users to plan and book trips in the Web. For this end, the service interacts with four service partners namely a flight reservation service, hotel reservation service, train reservation service, and a car rental service.

As in any software development the design of the TRS business process is based on requirements which consist of functional requirements (*FR*), non-functional requirements (*NFR*), and technical requirements[3]. The functional

[2]Released with NetBeans from Oracle Website.

[3]We are not interested in our work in this last kind of requirements.

requirements include the main functionality in a travel agency reservation system which are in our example the four service partners.

In addition to the functional requirements, the TRS system has initially the following non-functional requirements:

- **NFR1**: Service consumers are granted access only if they are authenticated, and no direct access to the backend resources of the service is allowed. The transmitted data must not be intercepted by unauthorized service consumers.
- **NFR2**: The TRS system must not deliver any sensitive data that may be used by malicious users which could compromise the integrity of the overall service.
- **NFR3**: The TRS system must ensure that the flight reservation service should be available during the reservation time (8:00 AM-6:00 PM) in the working days. If the service does not respond within 60 seconds the TRS system should notify the system administrator.

The three NFRs are integrated into the orchestration at design time. After the NFRs specification analysis the architect identified the first quality attribute she/he wants to implement in the web service orchestration from NFR1, which is the “access security” (QA1). The second and the third quality attributes, “data security” (QA2) and “reliability” (QA3) are identified respectively from NFR2 and NFR3.

At the beginning, the architect designing this orchestration starts by looking (and/or developing) for candidate service description interfaces that offer the needed functionality of the aforementioned services of the TRS system. After getting the identified service description interfaces, she/he integrates them into the web service orchestration and invokes them in the desired logic.

We will see now some evolution scenarios which target quality requirements of this service-oriented system, in which two additional NFRs emerged after a certain period of time in the system’s lifetime.

After a period of time, the architects realized that the service needs to access additional databases (of different airline companies) having different formats, which resulted in a portability (labelled QA4) quality evolution.

A long time after creating the system, the company providing these services has expanded significantly, and therefore more users requested the TRS system. Consequently, the architect observed that the performance (QA5) of the overall service (TRS Service) has decreased due to a subsequent increasing number of user requests, which imposed managing a large amount of data. As the amount of concurrent usage increases, so does the amount of the generated responses, leading to increased resource consumption of the entire service.

The two new additional NFRs are:

- **NFR4**: The TRS system should be able to support new data formats required by the service partners and therefore, compensates their behavior modifications so that the consumers are not impacted.

- **NFR5**: The TRS system processes and validates a large amount of data. To increase performance, the transmission of unnecessary data to the consumers should be avoided.

In order to satisfy the previous NFRs, several SOA patterns have been applied by the architect in the TRS business process. Figure 1 shows the distribution of patterns in this business process. Its design involved the use of five patterns that are introduced incrementally into the orchestration[4]. Table 1 enumerates each of the embodied patterns and its achieved quality attribute.

A given quality attribute can be implemented using several patterns inside a software architecture. For example, the portability quality attribute can be concretized by three different design decisions: the choice of the *Facade* service pattern, the choice of the *MVC* pattern and the use of abstract APIs. Since the quality attributes that we have to deal with in a service-based system can be listed in an exhaustive way (many quality models exist), their corresponding implementation solutions (SOA patterns) can also be exhaustively listed to some extent (by considering catalogs, such as [5], [6], or [7] which is more specific to SOA). These quality implementations represent recurrent solutions and seem generic enough to be "formally" specified then processed in a semi-automatic way to be used in different quality integration scenarios.

Even if these implementations could be reused, finding a solution (the well suited to be implemented) among several ones for a given NFR is not a trivial task for the architect. There is about eighty-five patterns for service-

[4]We presented four (4) of the patterns in Figure 1 for space limitation.

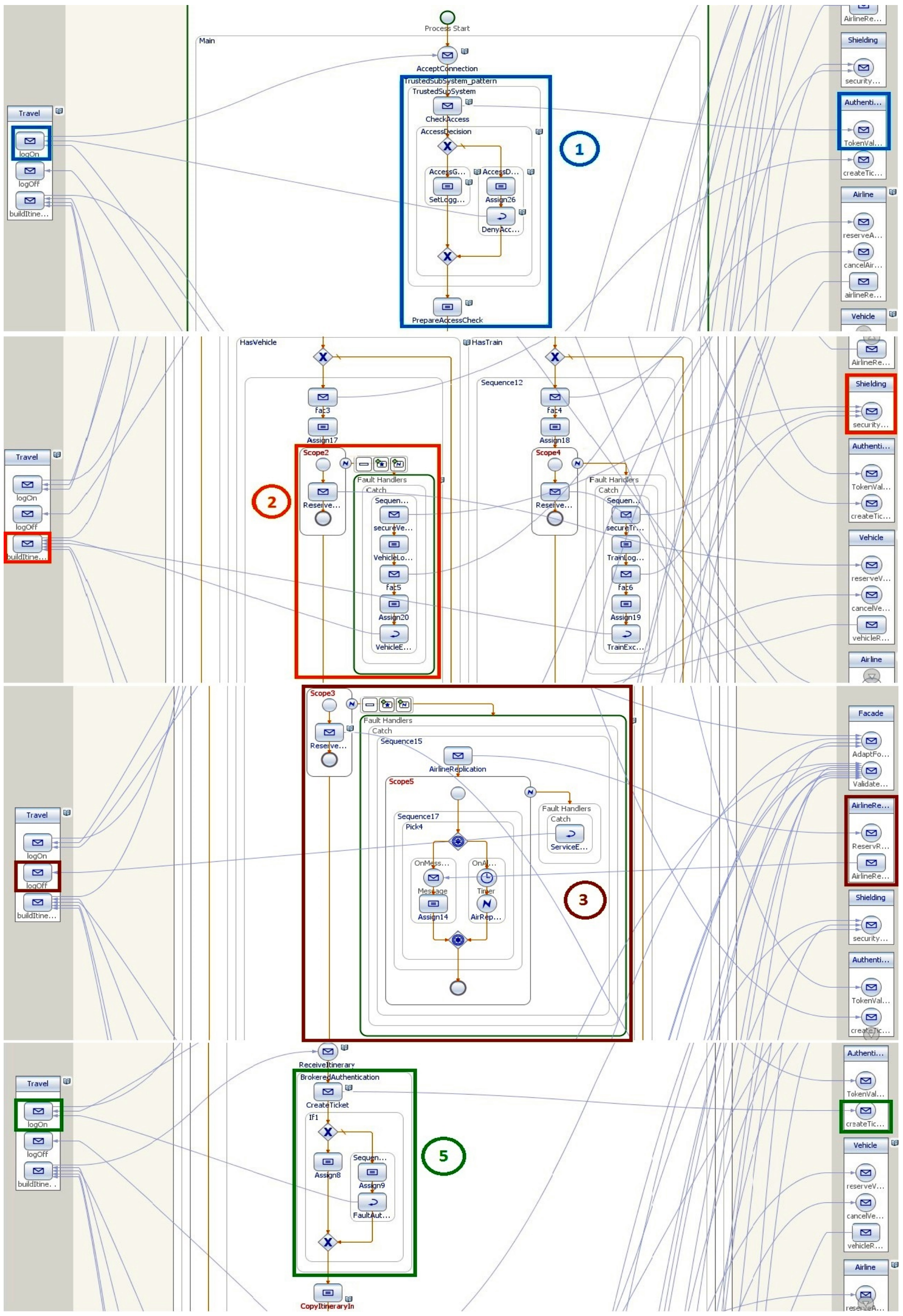


Figure 1: An excerpt of the TRS Business process showing the distribution of the embodied patterns

Table 1: Embodied Patterns and their achieved Quality attributes

Pattern	Trusted Subsystem (1)	Exception Shielding (2)	Replication (3)	Service Facade (4)	Brokered Authentication (5)
Quality Attribute	Access Security (QA1)	Data Security (QA2)	Reliability (QA3)	Portability (QA4)	Access Security (QA1)

based systems that have been described in [7] and the SOA Patterns website[5], about thirty of them [8], each having several variants, can be applied at an architectural level. This makes difficult the decision making for the architect. The reason for that is related to the way each solution concretizes a quality attribute, and what impact it could have on the software architecture. This is especially true, when the architect (a novice one) does not know the existing patterns for a targeted quality attribute or she/he is newly assigned to the software project. For example, the reliability quality attribute can be concretized by the *"Replication pattern"* in different possible ways with different variants namely, *"Naive Replication"*, *"Smart Replication"*, and the *"Passive Replication"*. What is the best possible choice between the three offered solutions? Each of them is suitable for the reliability but one is better than the other depending on the context in which the pattern will be applied. For QA4 and QA2, the architect may not be able to figure out the application of the *"Facade Pattern"* for the portability quality attribute, or the use of the *"Exception Shielding Pattern"* to secure her/his orchestration. Besides, even if the architect is assisted by a collection of reusable patterns, it is difficult for her/him to know the way each of the patterns has to be applied on the software architecture. For example, for satisfying QA1 an architect may not know how to exactly apply the *"Trusted Subsystem Pattern"* in her/his Web service orchestration.

The process we propose in the following section aims to address the aforementioned problems and helps the architects to: i) find one or several SOA patterns to answer an integration of a quality attribute in their orchestration, ii) choose among several ones the most suitable one, and iii) apply a pattern in their architecture (or cancel an existing pattern if the integration consists

[5]http://www.soapatterns.org

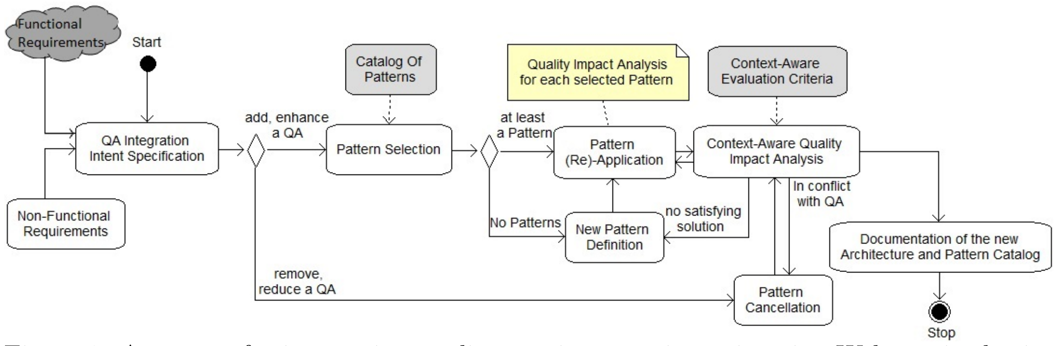


Figure 2: A process for integrating quality requirements in engineering Web service business processes

in weakening or removing an existing quality attribute).

3. SAQIM: a Quality Attributes Integration Method for Service Oriented Architecture

Figure 2 shows the multi-step process that we propose in our work to deal with quality requirements integration. During the process execution, its steps are handled automatically or in a semi-automatic fashion and thus need the architect's involvement. The process steps are detailed in the following subsections.

3.1. Quality Attribute Integration Intent Specification

The architect begins the design usually with functional requirements. We believe that at the design phase some quality attributes are correlated with functional requirements, hence, they have to be processed at the same time with them. For example, in order to integrate the reliability quality attribute, the architect may replicate some service partners. Thus, she/he should look for similar service partners that satisfy the same functional requirement. Consequently, those service partners together allow to achieve the reliability quality attribute.

The architect should first gather the needed information that may help her/him to take decisions correctly while going through the different steps of the process. This information is specified according to a template described in Table 2. The architect provides in this template the quality attribute targeted by this integration activity from the quality requirements specification (*i.e.* the architect wants to implement in the service orchestration). We adopt at

Table 2: Template for Quality Integration Intent Description

Element	Scope	Description
Quality Attribute Integration	What?	State the quality attribute targeted by the integration activity.
Integration Kind	How?	State if the integration targets to add a new quality attribute, enhance, weaken or withdraw the quality attribute.
Related Quality Attribute	Ultimately what?	If the integration kind is withdrawing or weakening the quality attribute, state here the quality attribute which will be ultimately enhanced or added (left empty otherwise).
Architectural Area	Where?	Indicate where in the orchestration changes will occur.

the top level of our specification the ISO 9126[6] quality model to represent quality attributes as quality characteristics and sub-characteristics.

We consider in our work the ISO 9126 quality characteristics mainly as “abstract” quality attributes and sub-characteristics as “concrete” quality attributes which are specializations of the first ones. Some ISO 9126 quality sub-characteristics like “security” are however still considered as “abstract” quality attributes for service-based systems. These sub-characteristics may have several specializations as “concrete” attributes like “Data security” and “Access security”. Additionally, the architect should specify where in the orchestration the changes have to be made. Hence, she/he should identify the architectural area that shows the scope of the change. It represents the architectural elements (or sets of these elements) in the BPEL process concerned by the changes. She/he does not specify an exhaustive list of all these elements, but only the main ones. For example, the architect can identify the `Assign` activities in the BPEL process after which `Invoke` activities should be added to integrate `Authentication`. Besides this, the architect has to indicate in her/his intent specification the integration kind by indicating if she/he wants to add (a new), enhance (an existing), weaken, or withdraw (an

[6]Software engineering – Product quality – Part 1: Quality model. The International Organization for Standardization Website:http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_ detail.htm?csnumber=22749

Table 3: Intent Specification for QII2

Integration Quality Attribute	Security/Data security
Integration Kind	Add
Related Quality Attribute	
Architectural Area	add pattern before ReserveVehicle, ReserveTrain, ReserveHotel Invoke activities

existing) quality attribute. This will determine the assistance type to provide in the following process steps. Additional information should be specified if the architect wants to withdraw or reduce a quality attribute. This is stated in the “Related Quality Attribute” section. Indeed, we argue that each time the architect wants to remove or weaken an existing quality attribute, she/he wants *in fine* to enhance or add another attribute, which is considered here as the “related quality attribute”. For example, when the architect tries to remove “Authentication” for affecting (weakening or removing) “Security”, there is a final goal of enhancing “Performance”. In the other integration kinds (add or enhance), this section is left empty.

Table 3 depicts the specification of a quality integration intent. It shows that the quality integration targets the “Data Security” quality attribute (QA2) which will be potentially added to the orchestration. The architect specified the BPEL elements being involved in the change which is shown in the “Architectural Area” section. This shows that the change will occur before “`Invoke`” BPEL activities in the TRS system which are `ReserveVehicle`, `ReserveTrain`, and `ReserveHotel`.

The integration intent specification is analyzed, and depending on the integration kind two cases are distinguished. These are detailed in the following subsections.

3.1.1. Quality Integration by Adding or Replacing a Pattern

In this case, the architect wants to enhance (replace the existing pattern implementing the quality attribute by applying one or several other patterns) or add a new quality attribute (apply a new pattern) to the orchestration. Therefore, a collection of patterns is suggested to the architect.

3.1.2. Quality Integration by Removing a Pattern

In other situations the architect may have to remove a quality attribute. For example, she/he may weaken or remove the security quality attribute (authentication). There are no proposed patterns from the catalog here since there is no pattern to apply to the architecture. Rather, a cancellation of the pattern implementing the quality attribute is performed. This cancellation is automatically obtained from the scripts for a pattern application. The pattern application and cancellation will be detailed in sections 3.3 and 3.6.

3.2. Pattern Selection

We consider in this work the existence of an "SOA Pattern Catalog", whose structure is detailed later. This pattern catalog is automatically analyzed using the "WS-BScript" toolset and this may result with a collection of patterns related to the targeted quality[7] which are proposed to the architect. The suggested patterns are then applied (Pattern Application step) on the orchestration by the architect in a semi-automatic way by configuring then executing their scripts (using WS-BScript toolset), to evaluate then automatically their impact on the existing qualities (using the WS-BScript toolset). The analysis may also result with no patterns. In this case, the architect is invited to define a new pattern (New Pattern Definition step). The proposed process is based on an "SOA Pattern Catalog", where each pattern is specified according to the model shown in Figure 3.

The pattern's specification includes a "name" with a textual description of its role. It includes also the "quality attribute" (The ISO 9126 quality characteristic or sub-characteristic considered as concrete quality attribute) that the pattern implements. Additionally, the pattern contains in its specification an "architectural script" which describes the way it should be applied in the orchestration. This script is composed of basic architecture changes which are a set of parameterized actions that aim to reconfigure the structure of the Web service orchestration. Actions are specified using a scripting language for Web service orchestration reconfiguration called *"WS-BScript"*. The last section in the description of a pattern contains the "architectural constraints", which are a formal specification of the structural conditions imposed by the pattern and allow the checking of its presence or absence in the orchestration.

Existing SOA patterns are usually presented in the literature following a functional organization (patterns for reliable messaging, patterns for atomic

[7]As stated previously, a quality attribute may be implemented by applying several patterns in different ways.

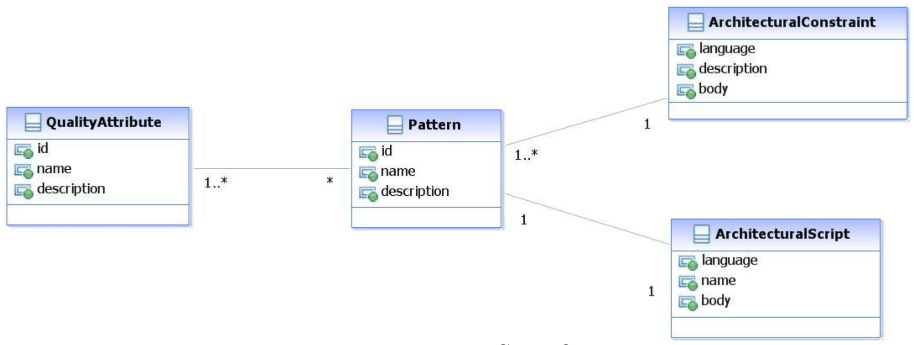


Figure 3: Pattern Specification

distributed service transactions, etc.). This does not answer our needs in this work where we would like to propose a pattern that concretizes a given quality attribute. Consequently, we organize the patterns catalog based on the qualities they implement.

The “SOA Pattern Catalog” is an important artifact in SAQIM. It is partially built before any use of SAQIM. It is then enriched, according to the model presented previously, each time a new pattern is used in the engineering of a given service orchestration using SAQIM. There are two roles associated to this catalog: i) a catalog administrator, whose responsibility is to feed the catalog with new pattern specifications (scripts, constraints, ...), and ii) a catalog user (an architect of a given orchestration), who will not directly manage the catalog, but will just see SAQIM suggesting the application of patterns retrieved from the catalog (or executing cancellation scripts processed from the catalog). As indicated previously, in some cases, the architect has the possibility to feed the catalog with new pattern specifications. In this case, the architect will play temporarily the role of an administrator. It is true that the responsibility of the architect is to design the system, but the fact that she/he is able to enrich the catalog will enable future instantiations of the same pattern, either in the same orchestration or in other orchestrations by benefiting from the automated support provided by SAQIM.

3.3. Pattern Application

This is an important step in the process where the selected SOA patterns are applied on a targeted Web service orchestration by means of some scripts, which specify simple architectural changes expressed with a Web service orchestration scripting language. To the best of our knowledge, there is no scripting language which allows the specification of set of actions that reconfigure WS-BPEL web service orchestrations. Therefore, we have developed a voluntarily simplified language called "*WS-BScript*" (for Web Service-BPEL Scripting). WS-BScript is a lightweight DSL that enables the architect to specify primitive changes making possible the reconfiguration of Web service orchestrations. The idea behind WS-BScript is to formalize some SOA patterns in order to apply them as much automatically as possible in the form of reusable design decisions. This language allows the definition of parameterized "scripts". A script is composed of a set of actions like add, wire, and remove, among others. The basic structure of a script is the following:

script apply<PatternName> (<listOfParameters>)
{ <setOfActions> }

A script declares a set of parameters, which represent the scope of the architectural actions. This set identifies BPEL orchestration elements involved in the changes brought by the elementary actions when applying a pattern. They form a super-set for the elements indicated in the architectural area of the quality integration intents, because generally more elements are needed to apply a pattern (these are requested from the architect). These actions are simple statements. We enumerate them in the following listing:

```
(01) add (BpelElement element, BpelElement AttachedParentelement,
                int  elementPosition)
(02) add (PartnerLinkElement element, String wsdlFileName)
(03) getPosition (String BpelElementName)
(04) create (BpelElement.Kind)
(05) remove (BpelElement element)
(06) wire (BpelElement element,PartnerLinkElement
        element,String PartnerLinkOperationName)
(07) unwire (BpelElement element,String PartnerLinkElement,
        String PartnerLinkOperationName)
(08) ask (String message)
(09) let variableName
(10) variableName = <expression>
(11) for(variableName : OrderedListVar) <actions>
(12) if (<condition>) <action1 or blocOfActions1>
       [else <action2 or blocOfActions2>]
(13) query (String OCLExpression)
(14) scriptCall (String scriptName([parameters])
(15) return (BpelElement element)
```

For instance, the first action adds a BPEL element to an orchestration, namely, Invoke, Assign, Receive, and other BPEL process elements (except `PartnerLink` BPEL element which does not require a position in an orchestration). An element is added in a specific position inside a parent element

(AttachedParentelement argument) in the orchestration. The second action adds specifically a BPEL `PartnerLink` (`PartnerLinkElement` argument) and links it with a given Web service specified by the `wsdlFileName` parameter. The *"getPosition"* action (Line 03) returns the position of a BPEL element in the orchestration specified by the `BpelElementName` parameter. It is used to identify precisely at what level we should apply a change in the orchestration. The architect may provide the name of the BPEL element, or a qualified name which indicates the path to the element if there are several elements with the same name in the orchestration. Line 04 indicates the *"create"* action which creates a BPEL element instance with some kind (the BPEL element that should be added to an orchestration like, Invoke, Assign, Sequence, Flow, Scope, etc. as defined in the BPEL specification). Line 05 shows the *"remove"* action which eliminates a BPEL process element (except in this case a PartnerLink) from an orchestration such as, a Sequence, an Assign or an Invoke, among others. The *"wire"* action binds a BPEL element to an operation `PartnerLinkOperationName` in a PartnerLink element. The opposite action of *"wire"* is *"unwire"* (Line 07). The *"ask"* action (Line 08) interrupts the execution of the script and waits for some customization values from the architect. It is commonly used in case of complex patterns application, which needs additional parameters that are not fully specified in the quality integration intent. Declaring variables is possible using the *"let"* action (Line 09). A variable can be initialized with an `expression` (Line 10). This expression can be a simple variable, a returned action's value like "getPosition" action, or even a value obtained after evaluation of an arithmetic expression. Variables can be of type integer, string, BPELElement, or Collection type. The *"for"* loop executes repeatedly a given block of actions (Line 11) which should be enclosed between braces. In addition to the *"for"* loop, it is possible to specify *"if-else"* statements (Line12). A condition in an if-else statement is a simple, or a composed boolean expression where we can use conjunction (&&), disjunction (||) and negation (!). The *"query"* action (Line 13) allows to navigate the BPEL meta-model through parameterized OCL [9] expressions and returns the expected result (BPEL elements usually). OCL has been chosen because of its simplicity [10] and the existence of a good tool support (OCL Toolkit [11], Eclipse MDT/OCL [12]). OCL is used as a navigation language in a complementary way with WS-BScript actions to get BPEL elements but without making any change to the orchestration. Composing patterns is possible through the *"scriptCall"* action (Line 14). It allows calling another pattern script by providing, as an argument, the name of the pattern script we want to call and its arguments. The last action (Line 15) can be used inside a script if the architect wants to return a given BPEL element that can be used by the caller script. Calling the return action terminates the script execution.

In this step of the process, the architect will apply one or several predefined[8] scripts (issued from the catalog of patterns) on her/his orchestration. For this end, the architect has to configure the scripts she/he wants to apply by initializing their parameters first and then by customizing them on the fly (through `ask` actions).

In the current implementation, the selected patterns are instantiated (from the pattern catalog which contains the description of patterns) and then applied on the Web service orchestration. It produces at last a new Web service orchestration. The architect is informed about the script application progress by displaying information on the embodied elements composing a pattern instance.

A registry of patterns is created in this step which references all the instances[9] used to build a service orchestration, each of which has a unique identifier. The registry exists during the quality integration assistance process; it is destroyed at the end of the process. Compared to the architecture documentation, it contains all the patterns that have been proposed to the architect for selection while the documentation contains only those chosen and applied on the service orchestration. At evolution time, where future changes may occur on the service orchestration, the registry could be restored from the architecture documentation to assist architects.

Listing 1 below shows a script example of the *Trusted Subsystem Pattern* [7] which implements the "Access Security" quality attribute. It prevents from unauthorized access to the resources of the TRS service by malicious attackers. It adds an authentication service on top of the invocation sequence in the orchestration to secure the service from direct access to the databases.

Before executing the script the architect is asked first to indicate its arguments. She/he has to give first the WSDL file (the `wsdlFileName` parameter) which represents the service. Second, she/he should indicate a specific op-

[8]The patterns scripts are already specified in the patterns catalog, the architect has just to apply them.

[9]Several instances of the same pattern may exist in an orchestration.

eration (the `partnerLinkOperationName` parameter) in the WSDL file representing the service. Then, she/he should state an operation name in the BPEL process (the `ProcessOperationName`) to which a reply is prerformed in case of an authentication failure. Finally, the `BpelElementName` parameter representing the BPEL activity after which a call to the authentication service has to be made is provided by the architect.

```
 1 script applyTrustedSubsystemPattern (String BpelElementName,
 2 String wsdlFileName, String partnerLinkOperationName,
 3 String ProcessOperationName){
 4 let position = getPosition (BpelElementName);
 5 let ocl = "self->closure(eContents().oclAsType(EObject))->select(a|
 6 a.oclIsKindOf(model::BpelType) and a.oclAsType(model::BpelType).name=
 7 'BpelElementName')->collect(a:EObject| a.eContainer())->asSet()";
 8 let elem = query (ocl);
 9 let aAssign = create (BpelElement.Assign);
10 add (aAssign, elem, position+1);
11 let aSequence = create (BpelElement.Sequence);
12 add (aSequence, elem, position+2);
13 let aPartnerLink = create (BpelElement.PartnerLink);
14 add (aPartnerLink, wsdlFileName);
15 let aInvoke = create (BpelElement.Invoke);
16 add (aInvoke, aSequence, 0);
17 wire (aInvoke, aPartnerLink, partnerLinkOperationName) ;
18 let aIf = create (BpelElement.If);
19 add (aIf, aSequence, -1);
20 let aCondition = create (BpelElement.Condition);
21 add (aCondition, aIf, 0);
22 ask(aCondition);
23 let aAssign1 = create (BpelElement.Assign);
24 add (aAssign1, aIf, 0);
25 let aElse = create(BpelElement.Else);
26 add (aElse, aIf, -1);
27 let aSequence1 = create (BpelElement.Sequence);
28 add (aSequence1, aElse, 0);
29 let aAssign2 = create (BpelElement.Assign);
30 add (aAssign2, aSequence1, 0);
31 let aReply = create (BpelElement.Reply);
32 add (aReply, aSequence1, -1);
33 wire (aReply, ProcesspartnerLink, ProcessOperationName);
34 }
```

Listing 1: Trusted Subsystem Pattern application script

The script starts first by looking through the *"getPosition"* action (Line 04) for the position of the BPEL activity (`BpelElementName` parameter) representing the architectural area after which the architect would like to apply the change. The *"getPosition"* action returns the position relatively to a BPEL activity's container. This is why we have to get the container BPEL activity of the `BpelElementName` activity so it could be possible to insert a BPEL activity just after it. To do so, in Lines 05-07 through a parameterized OCL expression with a generic format the script gets the container element of the `BpelElementName` activity. The OCL expression accepts two parameters, the name of the `BpelElementName` activity and the type (`BpelType`) of the activity (namely, Receive, Reply, Invoke, Assign, Sequence, etc. as defined in the BPEL specification). This latter is automatically deduced by the *"WS-BScript"* toolset and injected in the OCL expression. The OCL expression format given in the script example navigates in an *Ecore* implemetation of the BPEL meta-model (see Figure 4).

The OCL expression is executed in Line 08 through the *"query"* action and the result is saved. Then, the script adds in Lines 09 and 10 an `Assign` activity for variables setting before adding a `Sequence` activity (Lines 11 and 12) inside which the remaining BPEL activities composing the pattern will be inserted. We should note that in the *"add"* action, the "0" value means an insertion at the beginning of the container activity and the "-1" value means an insertion at the end, otherwise the architect has to specify the exact position. After that, the script adds a `partnerLink` BPEL activity to the targeted orchestration (Lines 13 and 14). Just after, an `Invoke` activity is added to the orchestration (Lines 15 and 16), having as attribute the `partnerLinkOperationName` parameter which indicates the operation to invoke in the previously inserted PartnerLink. Line 17 binds the `Invoke` activity to the PartnerLink. The script adds If-Else BPEL elements (Lines 18-21, 25 and 26) to specify the case of success, or failure of the authentication for which a `Reply` is intended (Lines 31-33) to answer the consumer a non-granted access. The script interrupts the execution through the *"ask"* action, asks on the fly for additional customization parameters and assists the architect to set the condition of the `If` element (Line 22). This script is executed on the BPEL description of the Web service orchestration which results in a new Web service orchestration implementing the security quality characteristic.

3.4. Quality Impact Analysis

There are two key elements that are used in the Quality Impact Analysis step of the process: i) the use of a quality-oriented assistance service that helps in diagnosing the consequences of any applied pattern on the other implemented qualities, and ii) the use of a Multi-Criteria Decision Making (MCDM) method, named "WSM" [13] (Weighted Sum Model), to evaluate a number of SOA pattern alternatives and to help the architect to select the most satisfactory pattern in a quality requirement integration step.

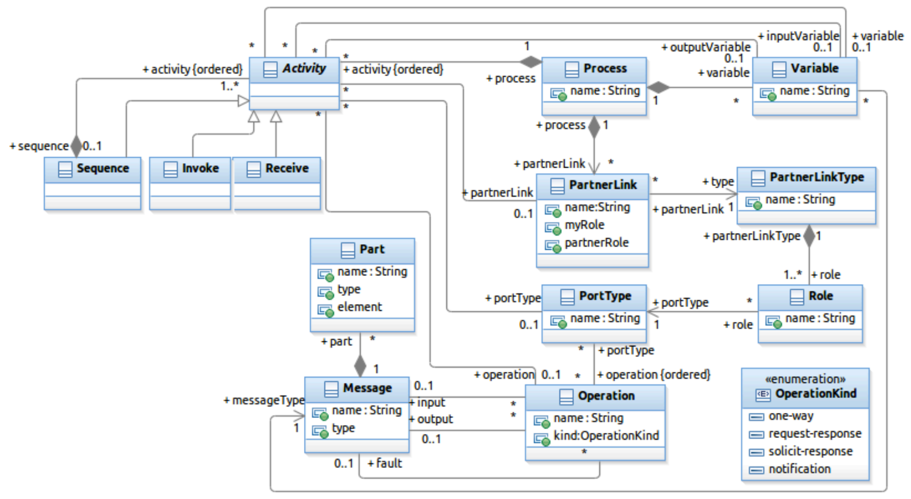


Figure 4: An excerpt of the BPEL/WSDL metamodel

The algorithm 1 shows the behavior of this step. It is composed of several functions. The algorithm is launched after the selected patterns are applied on the service orchestration in the “Pattern Application” step. Each pattern is applied on an instance of the targeted service orchestration.

During the quality integration process, the information encapsulated in the architecture documentation (See section 3.7) is exploited by the assistance algorithm in order to assist architects. The main purpose is to drive software architecture change to a situation where the quality integration intent is satisfied and the existing quality is minimally affected. This is done in three main steps: i) constraint evaluation and data collection; ii) pattern ranking, and iii) result reporting. The algorithm starts first by looking for the architecture documentation associated to the service orchestration which has been changed. Then, in the first step the algorithm checks each constraint (Line 20) in the documentation (by calling a function which is detailed in section3.4.1) and collects a part of the necessary data to partially configure the ranking system (WSM). In the second step, the ranking system collects first the remaining data required to complete its configuration then computes and returns the ranking scores of all the patterns (ADs) in a descending order (Lines 21- 26). It is obvious that there is no need for the ranking system if there is only one selected pattern. In the last step, the results are reported (Line 27) to the architects to allow her/him to choose a pattern from the selected ones. After that, the developer is asked to pinpoint the architecture decision and the quality attribute associated to the changes, if any (Lines 28- 31). At last, if the changes generate a new architecture decision

Algorithm 1: Quality Integration Assistance

```
begin
    let AE := Architectural Element;
    // a service orchestration;
    and AD := Architectural Decision;
    and AC := Architectural Constraint;
    and QA := Quality Attribute;
    and AT := Architecture Tactic;
    // a couple composed of a QA and an AD;
    and Doc:= architecture documentation associated to changed AE;
    and let wsmParams:= { };
    // an empty list of WSM system parameters (Aij, Wj);
    and let rankedPatterns:= { } ;
    // an empty list of pairs (AD, score);
    Function main(){
    begin
        after Pattern Application {
        foreach (AT in Doc) do
            QA := QA in AT;
            AD := AD in AT;
            $checkArchitecturalConstraint(AD)$;
             let A2j= ask for the context-suitability decision criterion
            value;
            $wsmParams := wsmParams + (A2j, W2)$;
            let score := runWsmSystem( );
            rankedPatterns := rankedPatterns + (AD,score);
        end
        $sort(rankedPatterns)$;
        $displayResults()$;
        let newAD := ask for AD associated to the new architecture, if
        any;
        if $newAD \neq null$ then
            let newQA := ask for the QA associated to newAD;
        end
        $addNewArchitecturalTactic(newAD, newQA)$;
        }
    end
    }
end
```

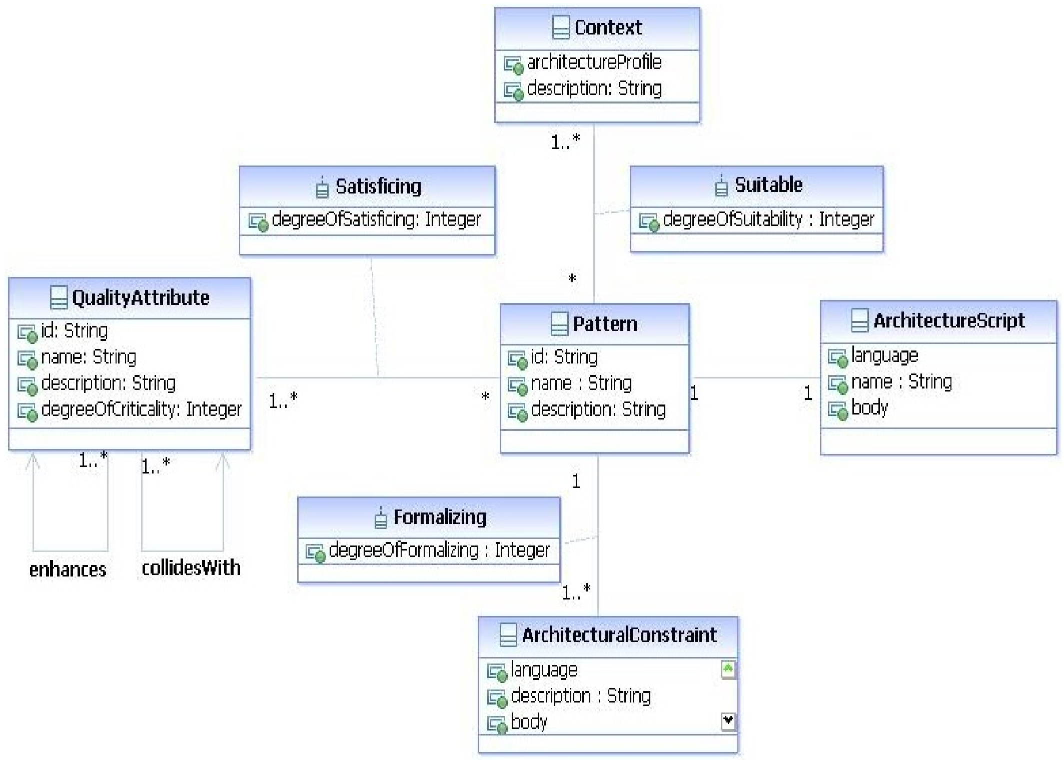


Figure 5: Links between Architecture Decisions and Quality Attributes

(the choice of a pattern), the algorithm adds (`addNewArchitecturalTactic (..)` function) to the documentation the couple composed of this new decision associated to its quality attribute, which is called an architectural tactic (Line 32).

We note here that the patterns (as design decisions) are previously documented by the architect according to the model proposed in [14]. This model introduces some fine-grained information (see Figure 5) namely, the criticality degree (C_1) of a quality attribute which represents its importance in the architecture, the formalization degree, which represents the extent to which some checkable constraints (present in the documentation) formalize the pattern, and the satisfaction degree, which represents the degree to which a design pattern contributes to satisfy a quality attribute. The documentation is enriched with a context-suitability degree (C_2), which is specified and documented at quality integration time because it depends on the pattern's suitability to a given situation and to the orchestration. This degree cannot be reused in different service orchestrations. It can however be reused in the future evolutions of the same service orchestration.

3.4.1. Quality-Oriented Assistance Service

The first element of the quality-related impact analysis step is an assistance service which aims to notify the architect of the consequences of the applied pattern on the other qualities. It indicates what are the related qualities that may be altered when applying the pattern which implements the new quality attribute. This assistance is mainly based on the evaluation of some OCL constraints that we used to specify SOA patterns parameterized architectural constraints [15] for Web service orchestrations. These constraints are defined using OCL and navigate in a metamodel of BPEL.

The function `checkArchitecturalConstraint(..)` detailed in the algorithm 2, checks the constraints associated to a given architecture decision received as an argument. It starts by checking the constraint expressions associated to the decision. If the checking does not succeed for a given constraint, a set of warnings are displayed to the architect by the `AffectedQAsNotifier (..)` function (Line 5). The displayed information includes the architecture decision, the exact architectural element impacted by the change, the degree of formalization of the decision, the quality attribute, its degree of satisficing and its criticality degree[10]. In addition, it shows to the developer the list of quality attributes which are eventually impacted by the change. For doing so, it uses the recorded information in the architecture documentation namely, the "related-quality" attribute (See section 3.7 for un example). It notifies also the developer (Lines 6 and 7) when adding a quality attribute to the service orchestration, about the quality attributes which are indirectly impacted (*i.e.* the quality attribute that its constraint still hold and is related to the added quality attribute). For example, when adding the portability quality attribute, the change may not invalidate the constraints formalizing the performance quality attribute implemented in the service orchestration but, this latter could be in a conflicting conceptual relationship with the portability.

Finally, the `checkArchitecturalConstraint(..)` function collects from the architecture documentation for each applied pattern a part of the necessary data for the ranking system (WSM) configuration (Line 8). This data is the criticality degree value (C_1) of the directly impacted quality attributes[11].

[10]For more details about the displayed information see [14]

[11]The change produced by the application of an architecture decision (a SOA pattern) may impact several quality attributes

Algorithm 2: Architectural constraints checking

```
Function checkArchitecturalConstraint($AD$)
begin
    let result := check AC ;
    if $result == false$ then
        $AffectedQAsNotifier(AD)$;
        warn "Other QAs may be in conflict with "+QA+": ";
        + QA_Relationships (QA,"collidesWith","both");
        $wsmParams := wsmParams + (A1j, W1)$;
    end
end
```

3.4.2. Weighted Sum Model for Pattern ranking

The "WSM" method is the second key element of the quality impact analysis step and is used only when the "Pattern Selection" step results in a collection of patterns for a targeted quality attribute. Its goal is to give a ranking on the selected patterns to choose the best alternative (having the highest WSM score).

Concerning this element, the MCDM problem we want to solve can be expressed as following: "what is the pattern that impacts the less the most important quality attributes, and is the most suitable to the architect preferences (context suitability, e.g., price, applicability related conditions, etc.)?" We have formulated the MCDM problem as follows:

- Alternatives are some selected patterns we want to classify;
- Decision criteria are defined as follows:
    1. Criticality of the impacted quality attribute (C_1);
    2. Context-Suitability of the pattern (C_2).

The "WSM" is considered as one of the most widely used methods for its simplicity [16]. If there are M alternatives and N criteria, then the best alternative (pattern) is the one that satisfies (in the maximization case) the following formula [13]:

$$A_i^{wsm} = max_i \sum_{j=1}^{N} a_{ij} w_j, for \qquad i = 1, 2, 3, ..., M. \tag{1}$$

$\sum_{j=1}^{N} w_j = 1$ and $w_j > 0, j = 1, ..., N$

a_{ij} is the value of an alternative "i" (pattern) in terms of a decision criterion "j". Weights represent the importance of each criterion according to the architect's preferences in the quality integration process.

In our approach, we choose the "Pairwise Comparison" method introduced in the "AHP" (Analytic hierarchy Process) method [17] to derive the data. AHP is highly mature, has a shallow learning curve (simple to learn within a reasonable length of time), uses quantitative measures and has clear-cut steps [18]. "Pairwise comparison" is known to have a good theoretical foundation and is easy for decision makers to understand [16]. In this approach the decision maker has to express her/his opinion about the value of one single pairwise comparison at a time by using the scale proposed by Saaty [17] depicted in Table 4. Pairwise comparisons are represented in a decision matrix. In our MCDM problem the data consist in the criteria weights (W_j) as well as the criteria values themselves (C_1 and C_2). This data constitute the parameters for the WSM ranking system. Weights should be derived in advance by the patterns catalog administrator, that means before using the proposed method. The criticality degree values (C_1) of the quality attributes defined in the adopted quality model[12] are derived by developers when expressing their preferences over quality attributes. The data (C_1) creation is done in the context of a service orchestration which may make a quality attribute more desirable than another (for example, security may be more advantaged than portability). Additionally, the criticality degree values should be also prepared beforehand and should be available to be used in the proposed method. They are automatically extracted after executing the scripts of the patterns being evaluated, because it depends on the criticality degree of the impacted quality attributes. If there is only one impacted quality attribute we take its criticality degree, if there are many, we take the sum of the criticality degrees of the impacted quality attributes.

Figure 6 shows an example of a decision matrix which represents the architect's preferences for the quality attributes defined in a service-oriented system project quality plan. An entry in the matrix, labeled a_{ij}, indicates how much the criticality for quality "i" is higher (or lower) than that for quality "j". Each quality has a value of "1" when compared to itself. Figure 7

[12]A company may define its quality attributes based on the developers experience.

Table 4: Scale of relative importance

Intensity of importance	**Definition**
1	Equal importance.
3	Weak importance of one over another.
5	Essential or strong importance.
7	Demonstrated importance.
9	Absolute importance.
2, 4, 6, 8	Intermediate values between the two adjacent judgments.
Reciprocals of above nonzero	If activity i has one of the above nonzero numbers assigned to it when compared with activity j, then j has the reciprocal value when compared with i.

shows the derived values for C_1[13].

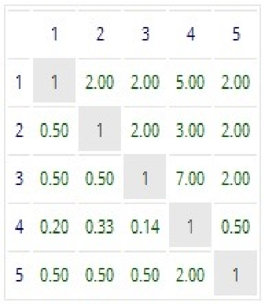
In AHP, the pairwise comparisons in a decision matrix are considered to be consistent if the corresponding *"consistency ratio (CR)"* is less than 10% [17]. The CR derived for the values in the below decision matrix is 5.4%. Finally, the context-suitability values (C_2) are derived when patterns are selected to be applied on the service orchestration. The data is specified before executing the pattern script because it is not documented yet since it is a context-dependent value and should be specified at design time.

- An example of the weights vector: W_1= 0.750, W_2= 0.250 respectively for C_1 and C_2 (prioritizing criteria weights show that the architects give more importance to C_1).

- The criticality degree weights vector (Figure 7) for the five quality attributes defined in the project quality plan: C_1Q_1= 0.348, C_1Q_2= 0.246, C_1Q_3= 0.224, C_1Q_4= 0.058, C_1Q_5= 0.124 respectively for QA1, QA2, QA3, QA4 and QA5.

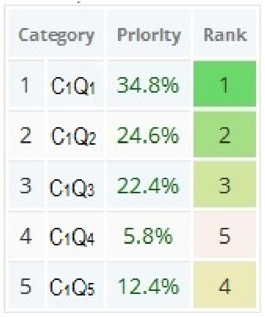
Hereinafter, an example in the selection process when dealing with the reliability quality attribute (QA3). The proposed solution (Pattern Selection step) for ensuring Reliability (QA3) was the *"Replication Pattern"* with its three different variants namely, the *"Naive Replication (RP_1)"*, the *"Smart*

[13]We used an online AHP priority calculator to calculate weights based on pairwise comparisons: `http://bpmsg.com/academic/ahp_calc.php`



	1	2	3	4	5
1	1	2.00	2.00	5.00	2.00
2	0.50	1	2.00	3.00	2.00
3	0.50	0.50	1	7.00	2.00
4	0.20	0.33	0.14	1	0.50
5	0.50	0.50	0.50	2.00	1

Figure 6: Decision Matrix.

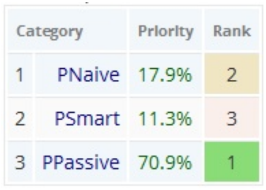


Category		Priority	Rank
1	C_1Q_1	34.8%	1
2	C_1Q_2	24.6%	2
3	C_1Q_3	22.4%	3
4	C_1Q_4	5.8%	5
5	C_1Q_5	12.4%	4

Figure 7: Weights for C1.

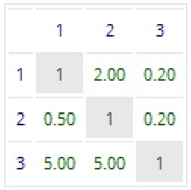
Replication (RP_2)", and the *"Passive Replication (RP_3)"*. The Replication pattern considers multiple implementations (as backups) of a service actively used, thus representing a point of failure in the system architecture. The architect decides to design a rescue system by the use of a backup service for the Airline service, which is used sequentially. A call to the second service is planned only if the first does not answer. The architect prefers the last variant of the pattern since its design solution organizes the service invocations in a hierarchical way, while the first two variants plan parallel invocations (the first waits for the first answer then continues, the second waits for all answers then picks the best one). Therefore, she/he gives a score (Pattern Application step) for the Context-Suitability Degree which is more important than the other patterns. Another advantage of the last criterion (context-suitability) is to distinguish between pattern variants suitability for a specific situation and a specific orchestration. Even if the same pattern variant is applied again on the same orchestration it would not have the same impact because the context is frequently not the same. The architect could have a preference for the "Smart Replication" if it is a matter of price of the delivered service. The architects proceed by configuring the WSM system with Context-Suitability criterion values (C2) of each pattern based on its preferences. Figures 8 and 9 show the derived values for C_2. For example, in Figure 9, in the row 1 column 2 of the matrix the architect slightly favors the "Naive Replication" over the "Smart Replication", hence she/he puts her/his judgment value "2". In the row 1 column 3 of the matrix, when comparing the "Naive Replication" with the "Passive Replication" the architect strongly advantages the latter, hence she/he puts the reciprocal value of "5" (0.20). The architect has just to fill (in case of manually doing the calculation) one half of the matrix (the upper half). The other half represents the reciprocal values.

When the WSM method is applied on the previous data, the scores of



	Category	Priority	Rank
1	PNaive	17.9%	2
2	PSmart	11.3%	3
3	PPassive	70.9%	1

Figure 8: Weights of C2 for the replication pattern.



	1	2	3
1	1	2.00	0.20
2	0.50	1	0.20
3	5.00	5.00	1

Figure 9: Decision Matrix.

the three alternatives are:

- PNaive (WSM score)= 0* (0.750) + 0.179* (0.250) = 0,04475
- PSmart (WSM score)= 0* (0.750) + 0.113* (0.250) = 0,02825
- PPasive (WSM score)= 0* (0.750)+ 0.709* (0.250) = 0,17725

The notification report shows a higher score of the *"Passive Replication (RP_3)"* with no impacted related qualities (those directly impacted and their related quality attributes in the orchestration), followed by RP_1 then RP_2. The results that yield the application of the WSM method are considered as the satisfaction degrees of each applied pattern for a quality attribute. Note that values between parentheses are weights. All variants have had no impact on any implemented quality attribute in the orchestration, which explains the "0" values for the first criterion (C_1).

3.4.3. SOA Patterns Architecture Constraint Specifications

After applying a pattern on a service orchestration we have to be sure that its structure is respected when making future changes, by imposing some architectural constraints. These latter are part of the pattern specification (see Figure 3) and serve to verify if an architecture conforms to the pattern or not. Since the pattern implements a quality attribute in the service orchestration, the non-conformance of its structure to the specified constraints implies an altered quality attribute.

To make these architectural constraints reusable artifacts, we build the SOA pattern catalog with parameterized constraints that can be configured then checked when applying a pattern into a service orchestration.

We give now an architectural constraint of the *"Passive Replication Pattern"* brought from our implemented catalog of SOA patterns. It is worth noting that constraints have been tested[14] on an *"Ecore"* implementation of

[14]Tests were held on an enriched version of the NetBeans travel agency application.

WS-BPEL meta-model, and that is why *"Ecore"* related details was removed for clarity.

Listing 2 shows the architectural constraints of the *"Passive Replication Pattern"* which is one of the three variants of the *"Replication Pattern"* that we have specified in the SOA patterns catalog. This pattern serves the *"Reliability"* quality attribute. Its design solution organizes the service invocations in a hierarchical way, a call to another replicated service is planned only if the first does not answer. The identified structural conditions characterizing this pattern are listed below:

i) The service to be replicated should be wrapped in a `Scope` BPEL activity. This guarantees to isolate the service that could eventually fail and allows to handle (through a `faultHandlers` BPEL activity) its failing in a `Catch` BPEL activity. This latter is defined inside a `faultHandlers` BPEL activity.

ii) In all the `Catch` activities attached to the `Scope` it should exist only one `Reply` BPEL activity. This latter represents the fault response case of all the replicated services and should be in the last `Catch`.

iii) The number of `Invoke` BPEL activities (representing the calls to the replicated services) where each one is contained in a `Catch`, equals the one of `Catch` activities minus one. The last `Catch` intercepts the failure case of the last replicated service.

iv) The service invocations are organized in a hierarchical way.

```
Context TRS: Process inv:
let scp :Set(Activity)=
 self->closure(oclAsType(Activity))->select(a:Activity|a.oclAsType(Scope).
     name='aScope') in
--The service to be replicated should be wrapped in a 'Scope' activity
 scp.oclAsType(Scope).activity->exists(b:Activity| b.oclAsType(Invoke).
     name='serviceTobeReplicated')
and
let cth :OrderedSet(Activity)=
 scp->closure(oclAsType(Activity))->select(c:Activity|c.oclIsKindOf(Catch)
     )->asOrderedSet() in
 let rep: Set(Activity)=
 cth->closure(oclAsType(Activity))->select(c:Activity|c.oclIsKindOf(Reply)
     ) in
--In all the 'Catch' elements attached to the 'Scope' it should exist only
     one 'Reply'. This latter represents the fault response case (if any)
     of all the replicated services and should be in the last 'Catch'
     element
 rep.oclAsType(Reply)->size()=1 and cth->last()->exists(c:Activity|c.
     oclIsKindOf(Reply))
and
 let ink: Set(Activity)=
15  cth->closure(oclAsType(Activity))->select(c:Activity|c.oclIsKindOf(Invoke
        )) in
16 --The number of 'Invoke' activities equals the one of 'Catch' activities
        minus one. The last 'Catch' intercepts the failure case of the last
        replicated service if any.
17  ink.oclAsType(Invoke)->size()>=1 and ink.oclAsType(Invoke)->size()= cth.
        oclAsType(Catch)->size()-1
18 and
19  let fhandlers :OrderedSet(Activity)=
20  scp->closure(oclAsType(Activity))->select(c:Activity| c.oclIsKindOf(
        FaultHandler))->asOrderedSet() in
21 --The service invocations are organized in a hierarchical way
22  if fhandlers->size() > 1 then
23  fhandlers->excluding(fhandlers->last())->forAll(aa,bb:Activity|aa.
        oclIsKindOf(FaultHandler) and
24  aa->exists(bb.oclIsKindOf(FaultHandler))) else false endif
```

Listing 2: Passive Replication Pattern Architectural constraint

Firstly, this constraint checks that the service invocation to be replicated should be wrapped in a `Scope` which the name is given as a parameter in the constraint (aScope in Line 3). This allows to establish a recovery after failure system by attaching to the `Scope`, a `faultHandlers` element offering the possibility to handle the failure of the service (serviceTobeReplicated in Line 5) in a `Catch` element. In the second part of the constraint (Line 12), we check that there is only one `Reply` that should be placed at the end of the `Catch` element hierarchy. The third part of the constraint (see Line 17) ensures that there is a `Catch` element which does not encompass a service invocation. Additionally, it ensures the existence of at least one invocation to a replicated service. Finally, the last part checks that the invocations to the replicated services are hierarchically structured since each `faultHandlers` element encompasses another one and, each one encompasses a `Catch` (Lines 22- 24). The fact that `Invoke` activities are encompassed by `Catch` activities ensures that a service is called only if its predecessor has failed. The failing of a service throws an exception which is intercepted by a `Catch`; this way we ensure passively the execution of one service at a time.

3.5. New Patterns Definition

The choice of a specific pattern or its rejection is the responsibility of the architect. If the architect is not satisfied with any of the proposed patterns, then she/he can define (as a pattern catalog administrator) new patterns, which she/he is asked to document according to the proposed specification (Figure 3). They will be considered as new reusable architecture design decisions that could potentially be applied on some architecture descriptions in the future.

After that, the architect is redirected to the "Pattern Application" step to simulate the effect of the new catalogued pattern. This is an important transition backward in the process, especially if the architect who catalogued the pattern is not the one who chose the patterns that are implemented in the architecture, and therefore, potentially did not know them. Consequently, she/he does not know the impact of the new pattern application on the other implemented qualities in the architecture. Hence, returning back to the "Pattern Application" step is necessary to assist the architect.

3.6. Pattern Cancellation

As we have mentioned in Section 3.1.2, the architect may want to remove or weaken a given quality attribute. In this case, the process execution takes another path, as illustrated in Figure 2. The process goes through the pattern cancellation step where an elimination of the concerned pattern is performed. This is done by deducing the opposite effect of the pattern's architectural actions, hence avoiding to the architect the burden of doing it manually or specifying the cancellation script. The generated cancellation script is then executed on the Web service orchestration. The generation of a cancellation script is handled automatically (by the "*WS-BScript*" toolset) following a bottom-up approach starting by the last action in the script and going up to the first one, by respecting some specific rules which are enumerated hereinafter: 1) keep the script parameters specified in the original script; 2) maintain the loops and if-else statements as they are; 3) ignore the "ask", "return", "create", "query" actions; 4) replace the "add" action by the "remove" action, and the "wire" action by the "unwire" action; 5) replace a script call by its corresponding cancellation script; and 6) replace the remove (BpelElement element) action by two primitives:

```
i) let element= create(BpelElement.Kind), and
ii) add (BpelElement element, BpelElement AttachedParentelement,
          int  elementPosition)
```

In the following listing 3 we show the cancellation script of the "Trusted Subsystem" pattern, whose application script is given in Section 3.3:

```
script cancelTrustedSubsystemPattern (String BpelElementName,
String wsdlFileName, String partnerLinkOperationName,
String processOperationName) {
 unwire (aReply, ProcesspartnerLink,processOperationName);
 remove (aReply);
 remove (aAssign2);
 7 remove (aSequence1);
 8 remove (aElse);
 9 remove (aAssign1);
10 remove (aCondition);
11 remove (aIf);
12 unwire (aInvoke,aPartnerLink,partnerLinkOperationName) ;
13 remove (aInvoke);
14 remove (aPartnerLink);
15 remove (aSequence);
16 remove (aAssign);
17 }
```

Listing 3: Trusted Subsystem pattern cancellation script

The script presented above cancels the application of the "Trusted Subsystem" pattern by reversing its actions from the last one to the first one. Line 04 unbinds the "Reply" activity from the "PartnerLink" before removing it (Line 05). Similarly, the other BPEL elements are removed in the opposite order they were added (Rule 4 stated above). The script parameters remain unchanged (rule 1).

The cancellation of a pattern from a service orchestration involves the following steps: i) looking for all the pattern instances the architect wants to remove from the pattern registry, and listing them to the architect; ii) the architect should choose manually the pattern instance to cancel; iii) if the pattern cancellation script has been already generated, apply the script, otherwise, generate the script and add it to the registry then apply it, and finally iv) manage pattern intersections (handled automatically by *"WS-Bscript" toolset*) by showing to the architect the BPEL elements pertaining to other pattern(s) that could be eventually removed when applying the script. If it is the case, it is up to the architect to validate the corresponding actions or not.

The consequences of removing the pattern instance implementing a quality attribute are reported to the architect by the quality impact analysis using the quality-oriented assistance service (architectural constraint checking), and it is the architect's responsibility to validate the change and hence documenting the new architecture, or repeat again the different steps of the process for a new architecture decision.

3.7. Documentation of the New Architecture

In this step, the chosen pattern is applied to the orchestration and added in the architecture decision documentation as a new design decision. This documentation contains all design decisions (SOA pattern choices) that was made to build the architecture. In addition, the architect has to complete a part of this documentation, namely the formalization degree of the pattern, and also the related qualities of the quality attribute. The criticality degree of the quality attribute the pattern implements, and the satisfaction degree of the pattern for the quality attribute are automatically added to the documentation by the *"WS-BScript toolset"*. This information is necessary for the futur quality integrations especially in the patterns selection process (quality impact analysis step). We show below an excerpt of the TRS system's architecture documentation. Its architecture documentation is presented in a synthetic way (in order to not be too verbose with its original XML-based description) in the listing below:

```
Architecture-Documentation :
1. Architecture-Tactic :
    This tactic ensures the Access Security quality requirement by using
    a Trusted subsystem pattern
    - Quality-Attribute name="Access Security" degreeOfCriticality="34,8"
    - Related-Quality name="Availability" relationship="Enhances"
                               relationType="weak" influence="negative"
    - Architecture-Decision name="Trusted subsystem pattern"
                                         degreeOfSatisficing="18,15"
                                         degreeOfContext-suitability="72,6"
    - Architecture-Constraint profile="BPEL" degreeOfFormalizing="90"

2. Architecture-Tactic :
    This tactic ensures the Data Security quality requirement by using
    a Exception Shielding pattern
    - Quality-Attribute name="Data Security" degreeOfCriticality="24,6"
    - Related-Quality name="Portability" relationship="Enhances"
                               relationType="weak" influence="negative"
    - Architecture-Decision name="Exception Shielding pattern"
                                         degreeOfSatisficing="75"
                                         degreeOfContext-suitability="80"
    - Architecture-Constraint profile="BPEL" degreeOfFormalizing="90"

3. Architecture-Tactic :
    This tactic guarantees the Portability quality requirement by using
    a Service facade pattern
    - Quality-Attribute name="Portability" degreeOfCriticality="5,8"
    - Related-Quality name="Performance" relationship="CollidesWith"
                               relationType="tight"
    - Architecture-Decision name="Service facade pattern"
                                        degreeOfSatisficing="90"
                                        degreeOfContext-suitability="95"
    - Architecture-Constraint profile="BPEL" degreeOfFormalizing="80"
```

The architecture documentation contains three architectural tactics. They document the links between architectural decisions (SOA pattern choices) and their corresponding quality attributes (QA1, QA2, QA4). In this documentation we can see among others the different relations between quality attributes (`Related-Quality` element in the listing above). For example, in the third tactic, the `Related-Quality` element shows that the portability and performance quality attributes are colliding and are tightly coupled.

4. Experimentation

We can distinguish two main roles of SAQIM. First, it is a system that provides an automated support for the integration (application and analysis) of SOA patterns into service orchestrations. Second, it is a recommendation system of SOA patterns satisfying quality attributes for service orchestrations. Due to the actual size of the pattern catalog which includes eleven patterns, we will focus on the evaluation of the first role. Indeed, it is not pertinent for example, to calculate the "precision" and "recall" as metrics to measure the efficiency and thus evaluate the research and selection aspects in SAQIM with the actual size of the catalog. Thereby, we addressed in particular the following research question:
"Compared to a manual quality integration, does the automated support provided by SAQIM give substantial help to architects?".
To answer the research question, we pursued the steps detailed in the following subsections.

4.1. Methodology

We compared some measures (presented later) obtained by using SAQIM with those obtained "without using" it[15]. To do so, we simulated quality integration (with and without SAQIM) by using a collection of 16 patterns: eleven of them are real patterns, and the remaining five are "imaginary"[16]. These latter, are unreal patterns in which we have varied randomly the number of BPEL elements (for scripts time specification), and the number of tokens (for OCL constraints time specification) to estimate their specification time by following a specific protocol (explained in the following subsection).

[15]We mean by "without using SAQIM" that the architect has to choose him(/her)self the patterns and uses the NetBeans BPEL designer to apply them manually.

[16]We used approximately the half of the real patterns total number.

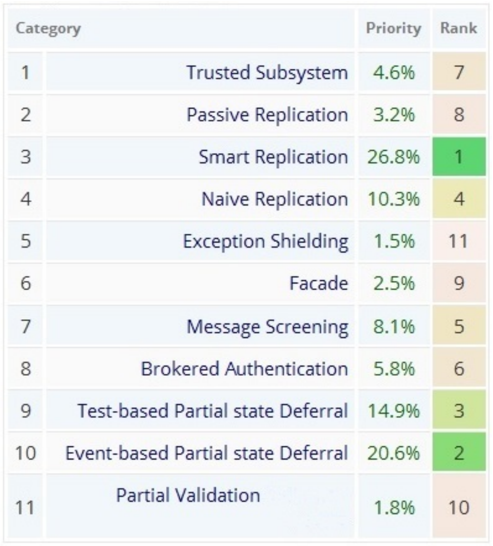
Table 5: OCL constraints specification results

Pattern	Tocl (min)	Var	CD	1-CD	Tocl*(1-CD)	Uocl	Uocl-Umec	NbTokens
Facade(1)	40,25	1,79	0,046	0,975	39,24	0,072	0,062	154
Trusted Sub-System(2)	78,5	0,16	0,032	0,954	74,89	0,058	0,048	366
Passive Replication(3)	65,5	4,04	0,268	0,968	63,40	0,055	0,045	333
Smart Replication(4)	115,5	5,54	0,103	0,732	84,55	0,049	0,039	497
Naive Replication(5)	68	0,66	0,015	0,897	61	0,044	0,034	394
Exception Shielding(6)	36	1,16	0,025	0,985	35,46	0,044	0,034	239
Message Screening(7)	63,5	2,79	0,081	0,919	58,36	0,043	0,033	365
Brokered Authentication(8)	56	1,54	0,058	0,942	52,75	0,041	0,031	363
Test-based Partial state Deferral(9)	62,25	1,62	0,149	0,851	52,97	0,036	0,026	459
Event-based Partial state Deferral(10)	75,5	2,16	0,206	0,794	59,95	0,036	0,026	461
Partial Validation(11)	30	1,29	0,018	0,982	29,46	0,034	0,024	213

Imaginary patterns are introduced in the experimentation to represent a relatively acceptable number of SOA patterns that we can find and use in a real development process. From the other hand, they allow to run simulations with a configurable number of patterns in the catalog so that we can evaluate our method in a reliable way. The experiment was conducted following the next steps:

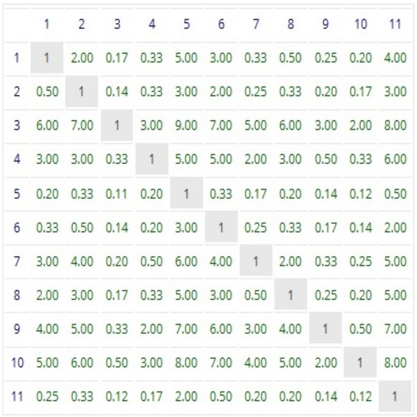
4.2. Data Collection

We have invloved in our experiment three Ph.D students in software engineering and programming languages. They have had the task of applying the patterns using NetBeans BPEL designer then measuring and recording the approximative time spent for each pattern. They were also asked to record the time spent in understanding each pattern after reading a textual documentation (retrieved from the literature). In addition, they were taught examples about the *WS-BScript* language. In addition to the first task, they were asked to specify patterns by writing OCL constraints and scripts for the eleven real patterns. These Ph.D students have basic OCL skills, a good knowledge of frameworks, styles and basic patterns of software design. The students were separated and were not told about the final goal of the experiment. Additionally, they were not told about their recorded results to ensure confidentiality. Moreover, students were selected with a relatively similar level of knowledge and background.



Category		Priority	Rank
1	Trusted Subsystem	4.6%	7
2	Passive Replication	3.2%	8
3	Smart Replication	26.8%	1
4	Naive Replication	10.3%	4
5	Exception Shielding	1.5%	11
6	Facade	2.5%	9
7	Message Screening	8.1%	5
8	Brokered Authentication	5.8%	6
9	Test-based Partial state Deferral	14.9%	3
10	Event-based Partial state Deferral	20.6%	2
11	Partial Validation	1.8%	10

Figure 10: Weights for OCL constraints.



	1	2	3	4	5	6	7	8	9	10	11
1	1	2.00	0.17	0.33	5.00	3.00	0.33	0.50	0.25	0.20	4.00
2	0.50	1	0.14	0.33	3.00	2.00	0.25	0.33	0.20	0.17	3.00
3	6.00	7.00	1	3.00	9.00	7.00	5.00	6.00	3.00	2.00	8.00
4	3.00	3.00	0.33	1	5.00	5.00	2.00	3.00	0.50	0.33	6.00
5	0.20	0.33	0.11	0.20	1	0.33	0.17	0.20	0.14	0.12	0.50
6	0.33	0.50	0.14	0.20	3.00	1	0.25	0.33	0.17	0.14	2.00
7	3.00	4.00	0.20	0.50	6.00	4.00	1	2.00	0.33	0.25	5.00
8	2.00	3.00	0.17	0.33	5.00	3.00	0.50	1	0.25	0.20	5.00
9	4.00	5.00	0.33	2.00	7.00	6.00	3.00	4.00	1	0.50	7.00
10	5.00	6.00	0.50	3.00	8.00	7.00	4.00	5.00	2.00	1	8.00
11	0.25	0.33	0.12	0.17	2.00	0.50	0.20	0.20	0.14	0.12	1

Figure 11: Decision Matrix.

Because the level of the Ph.D students skills is close one to another, we notice an insignificant variance (see Column 3 in Table 5 and Table 6) in the measured times across students for each pattern. Therefore, the recorded times were "homogeneous" and this is why we took the average time. Now, to estimate the specification time for both OCL constraints and scripts for a number of imaginary patterns in a reliable way, we followed a specific protocol. The aim is to estimate the pattern catalog specification overhead from the one hand, and to simulate quality integration with a configurable

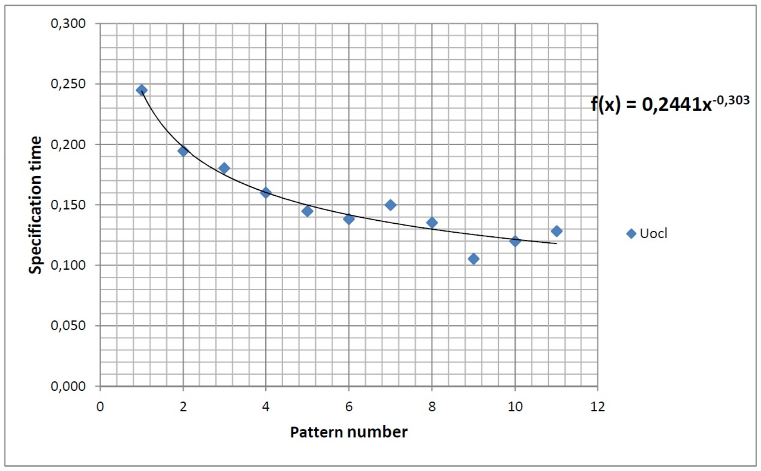


Figure 12: Inverse power regression on Uocl values

number of patterns from the other hand.

OCL constraints: The obtained values for the 11 real patterns are depicted in Table 5. We first normalized these estimated time values. Indeed, the Ph.D students have naturally acquired experience when specifying each time a new constraint. This experience can bias our experiment. We have thus decided to dismiss it, in order to get the most possible objective values. We have measured an approximative *coefficient of difficulty* (CD) for each constraint. CD represents the architect's opinion on the perceived difficulty when specifying a constraint. We applied here AHP pairwise comparisons for prioritizing OCL constraints difficulty (Figures 10 and 11) in the same way as for C_1 values (Figures 6 and 7). The developer expresses her/his opinion (measured on the scale of Table 4) like: "constraint i has an absolutely higher difficulty than j" has a value of "9". Then, we multiplied the previous specification time values by $1 - CD$.

The next step was to calculate the specification time for a lexical unit (token) in a constraint "`Uocl`". Values were obtained as follows:

$$Uocl = \frac{Tocl * (1 - CD)}{NbTokens} \tag{2}$$

Now, having time unit values for each pattern constraint we can apply a regression *f(x)* model to extrapolate values for the other imaginary patterns. We can notice that "Uocl" values (in Table 5) have a decreasing trend, but actually they do not converge to zero. Instead, they converge to a minimal value corresponding to specifying a constraint as a "mechanical task", *i.e.*

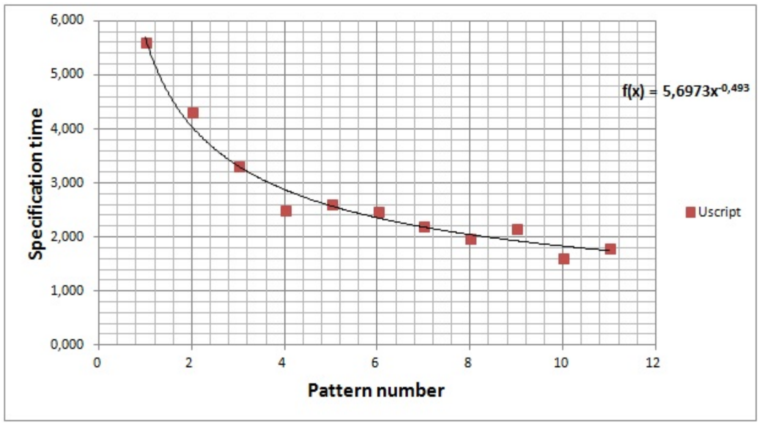


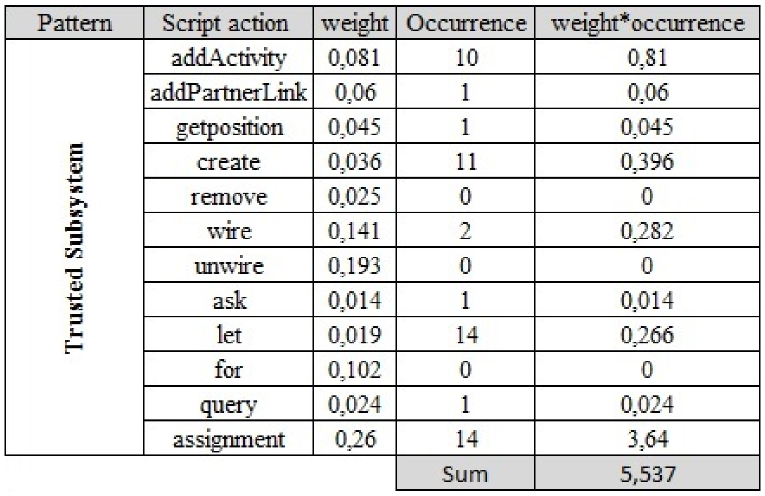
Figure 13: Inverse power regression on Uscript values

without having to think about complex parts in it. Therefore, we calculated "Umec" and we obtained 0.039 minute/token. So, the function of our regression should be defined as : *f(x)+Umec* . After that, we subtracted "Umec" from "Uocl" values then we applied an inverse power regression on the new values (Uocl-Umec in Table 5). We found that the inverse power model is the one that best fits our data. The result is illustrated in Figure 12. At the end, we used the following inverse power regression function to extrapolate time unit values for OCL constraints: $f(x) = 0,244x^{-0,303} + 0,039$. Finally, using Formula 2 we obtained the specification time *Tspec_ocl*:

$$Tspec\_ocl = \frac{(Uocl - Umec) * NbTokens}{(1 - CD)} \quad (3)$$

Scripts: We followed the same steps as for OCL constraints to determine the specification time for pattern scripts, except for the "*coefficient of difficulty*" (CD) estimation. We started first by listing the different script actions[17] used in the scripts, then using pairwise comparisons we calculated the weight of each action according to its difficulty of use (Figures 15 and 16). The next step was to calculate the occurrences of each action in each script to get its global weight. Figure 14 shows an example of the way "CD" values have been estimated. Column 3 in Figure 14 shows the individual CD values obtained for each action (from Figure 15). The overall value of the different

[17]To distinguish between adding a BPEL "activity" and adding a "PartnerLink" BPEL element we suffixed "add" by the terms "Activity" and "PartnerLink" (Lines 1 and 2 in Figure 14).



Pattern	Script action	weight	Occurrence	weight*occurrence
Trusted Subsystem	addActivity	0,081	10	0,81
	addPartnerLink	0,06	1	0,06
	getposition	0,045	1	0,045
	create	0,036	11	0,396
	remove	0,025	0	0
	wire	0,141	2	0,282
	unwire	0,193	0	0
	ask	0,014	1	0,014
	let	0,019	14	0,266
	for	0,102	0	0
	query	0,024	1	0,024
	assignment	0,26	14	3,64
			Sum	5,537

Figure 14: Scripts CD values estimation

weights constitute the pattern's script CD value (5.537 for the script of the Trusted Subsystem Pattern, see Figure 14). This corresponds to 9.05% of the overall value for all the pattern scripts which represents 0.0905 (See Table 6). We defined the time for adding a single BPEL element by a script as the time unit "`Uscpt`":

$$Uscpt = \frac{Tscpt * (1 - CD)}{NbBpelElem} \tag{4}$$

Figure 13 shows the result of applying an inverse power regression model: $f(x) = 5,6973x^{-0,493} + 1,89$

$$Tspec\_script = \frac{(Uscpt - Umec) * NbBpelElem}{(1 - CD)} \tag{5}$$

$$Tspec\_pattern = Tspec\_ocl + Tspec\_script \tag{6}$$

Using the formulas 2, 3, 4 and 5 we were able to estimate the specification time (formula 6) for the 5 "imaginary" patterns.

4.3. Simulation

The aim of the simulation is to evaluate SAQIM's cost effectiveness. Table 7 shows the measures for the real patterns used in our simulation process. We calculated the necessary time for integrating a quality attribute without using and with using SAQIM (Columns 2 and 3). Column 3 includes the pattern script configuration and automatic application time (Column (a)), the

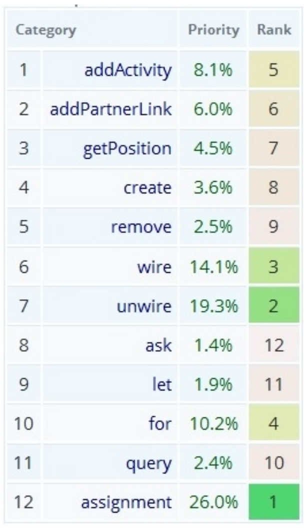
Table 6: Scripts specification results

Pattern	Tscpt	Var	CD	1-CD	Tscpt*(1-CD)	Uscpt	Uscpt-Umec	NbBpel Elem
(1)	39	1,54	0,0378	0,9622	37,52	7,50	5,615	5
(2)	75	2,54	0,0905	0,9095	68,21	6,20	4,311	11
(3)	90	3,5	0,1297	0,8703	78,33	5,22	3,332	15
(4)	81,5	4,66	0,1353	0,8647	70,47	4,40	2,514	16
(5)	50	2,16	0,0967	0,9033	45,17	4,52	2,627	10
(6)	38	1,16	0,0771	0,9229	35,07	4,38	2,494	8
(7)	40,25	0,87	0,0819	0,9181	36,95	4,11	2,216	9
(8)	38,5	1,04	0,0968	0,9032	34,77	3,86	1,974	9
(9)	36	0,29	0,1011	0,8989	32,36	4,04	2,155	8
(10)	50,75	0,79	0,1016	0,8984	45,59	3,51	1,617	13
(11)	19,50	0,29	0,0514	0,9486	18,50	3,70	1,809	5

OCL constraint configuration time (Column (b)), and the pattern documentation time (Column (c)), when using SAQIM. Column 2 includes the time spent in understanding each pattern as well as the time spent in manually applying a pattern using the Netbeans BPEL editor (without using SAQIM). The last column shows the specification time (OCL constraints and scripts) for each pattern which is used in the simulation process when using SAQIM with measures of Column 3 (Columns (a), (b), and (c)). The simulation has been run in two different situations based on an assumption stating that without SAQIM, the architect has at her/his disposal the same patterns used by the architect that uses SAQIM and which she/he should apply manually to integrate quality attributes. This simplification assumption does not bias the results of the experiment. Rather, it ignores the time spent by architects for searching appropriate patterns, which favors the situation of “not using SAQIM”.

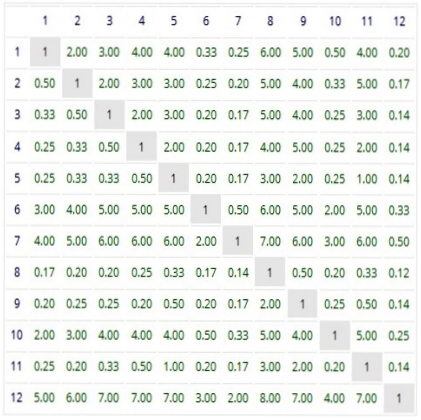
We conducted our simulation process on the TRS system according to three different scenarios. The simulation has been iterated around 50 times to maximize randomness.

Worst case: We simulated the application of SAQIM, using a random generated order of 16 patterns, then we re-applied SAQIM using the same order of patterns but without counting their specification time. We simulated also the quality integration without using SAQIM. The aim in this case is to observe the situation where SAQIM is less beneficial. Figure 17 shows the



Category		Priority	Rank
1	addActivity	8.1%	5
2	addPartnerLink	6.0%	6
3	getPosition	4.5%	7
4	create	3.6%	8
5	remove	2.5%	9
6	wire	14.1%	3
7	unwire	19.3%	2
8	ask	1.4%	12
9	let	1.9%	11
10	for	10.2%	4
11	query	2.4%	10
12	assignment	26.0%	1

Figure 15: Weights for script actions.



	1	2	3	4	5	6	7	8	9	10	11	12
1	1	2.00	3.00	4.00	4.00	0.33	0.25	6.00	5.00	0.50	4.00	0.20
2	0.50	1	2.00	3.00	3.00	0.25	0.20	5.00	4.00	0.33	5.00	0.17
3	0.33	0.50	1	2.00	3.00	0.20	0.17	5.00	4.00	0.25	3.00	0.14
4	0.25	0.33	0.50	1	2.00	0.20	0.17	4.00	5.00	0.25	2.00	0.14
5	0.25	0.33	0.33	0.50	1	0.20	0.17	3.00	2.00	0.25	1.00	0.14
6	3.00	4.00	5.00	5.00	5.00	1	0.50	6.00	5.00	2.00	5.00	0.33
7	4.00	5.00	6.00	6.00	6.00	2.00	1	7.00	6.00	3.00	6.00	0.50
8	0.17	0.20	0.20	0.25	0.33	0.17	0.14	1	0.50	0.20	0.33	0.12
9	0.20	0.25	0.25	0.20	0.50	0.20	0.17	2.00	1	0.25	0.50	0.14
10	2.00	3.00	4.00	4.00	4.00	0.50	0.33	5.00	4.00	1	5.00	0.25
11	0.25	0.20	0.33	0.50	1.00	0.20	0.17	3.00	2.00	0.20	1	0.14
12	5.00	6.00	7.00	7.00	7.00	3.00	2.00	8.00	7.00	4.00	7.00	1

Figure 16: Decision Matrix.

experiment result.

Best case: This case represents the parallel use of patterns. So, we used the same pattern order of the worst case, then we simulated the application of SAQIM by considering the parallel design of three (3) BPEL orchestrations. That means, in the first time we took into account the pattern's specification time. In the remaining two applications we have not taken it into account. We simulated the quality integration without using SAQIM with the same pattern order. The aim in this case is to observe the situation where SAQIM is the most beneficial. The result is given in Figure 18.

Random case: we simulated the application of SAQIM by adding each time we want to integrate a quality, the pattern specification time (last column in Table 7) only if it is the first use of the applied pattern. In the second step, we simulated the quality integration without using SAQIM. The experiment result is shown in Figure 19.

4.4. Discussion

The worst case (Figure 17) shows that SAQIM begins to be cost effective starting from the 29-th iteration, which corresponds to (more or less) approximatively 25.23 hours (three full-time working days of 8 hours). It is worth noting that, according to our estimations, the pattern catalog specification (with 16 patterns) takes 24.51 hours. The difference between the two measures, which equals 43.2 minutes, is the estimated time taken in this

Table 7: Measures used in the simulation process

Pattern	Time (minutes)				
	without using SAQIM	using SAQIM			pattern specification time
		(a)	(b)	(c)	
(1)	43,37	0,59	0,72	1,57	76,77
(2)	52,42	0,65	0,39	3,20	143,10
(3)	44,47	0,93	0,66	1,40	141,73
(4)	56,28	0,86	1,00	1,10	155,02
(5)	56,27	0,71	0,65	1,03	106,16
(6)	59,12	0,67	1,18	2,17	70,53
(7)	49,42	1,17	1,23	2,05	95,31
(8)	49,23	0,63	1,03	1,58	87,53
(9)	56,16	0,57	1,18	2,16	85,33
(10)	48,09	0,56	0,98	1,25	105,54
(11)	57,48	0,71	1,09	1,54	47,96

simulation for making profitable the approximate three working days of the (16-pattern) catalog specification time, in the worst case.

The random case (Figure 19) shows that SAQIM begins to be cost-effective starting from the 19-th iteration, which corresponds to (more or less) approximatively 22.78 hours (2.84 working days of 8 hours). This period of time corresponds to the time of learning and familiarizing the architect with the method. Indeed, as in all engineering methods learning involves an additional cost. We can also say that the use of the pattern catalog is capitalized after the aforementioned period of time.

Note that in the TRS system which is a medium size project, we embodied using SAQIM ten patterns (including pattern instances like for the "Exception Shielding pattern"). Therefore, we can deduce that SAQIM becomes beneficial after the construction of the second BPEL process (when referring to the random case). Furthermore, if the patterns catalog is used with several BPEL processes in parallel (the best case in Figure 18), the catalog specification time will be distributed over all these processes, and hence, the use of SAQIM becomes more beneficial. If we consider that the pattern catalog will be developed by three architects, the specification time (24.51 hours) will be divided by three, i.e. approximately 8.17 hours (one working day).

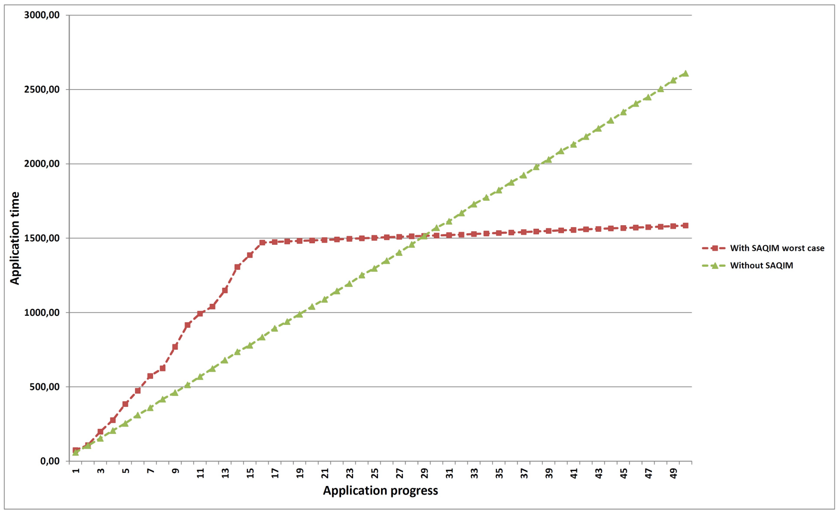


Figure 17: Worst case time variation

We have tested SAQIM with a relatively small set of patterns (16 patterns). We repeated the experience by increasing the number of patterns to thirty (30). We kept the 11 real patterns and we created 19 imaginary ones. Then, we generated different random orders of patterns (6 random orders) and used them in the simulation process. The aim of the experiment is to observe the behavior of SAQIM with a larger number of patterns in the catalog. We found that SAQIM's cost effectiveness (for the random case) varies between the 8-th and 23-rd iteration. This variance is related to the patterns order. Note that the result found with 16 patterns is in the range found when using 30 patterns. We can deduce that the number of patterns in the catalog does not affect SAQIM cost effectiveness. Moreover, to estimate the overhead of the quality impact analysis step (Section 3.4.1 and Section 3.4.2) we have measured its execution average time. We found that, it takes approximately one minute (including the context-suitability criterion configuration time). This constitutes 25.08% (according to the Columns (a), (b), (c) of Table 7) of the overall time for the "Passive Replication" Pattern. This means that, in the worst case, it makes SAQIM beneficial after 25.72 hours instead of 25.23 hours, which represents a difference of a small period of time of 29.4 minutes (0.49 hours).

4.5. Threats to validity

Wohlin et al. describe four areas where the validity of the results may be threatened [19], we discuss threats in each of these areas.

Internal validity: We used a combination of real data and simulated

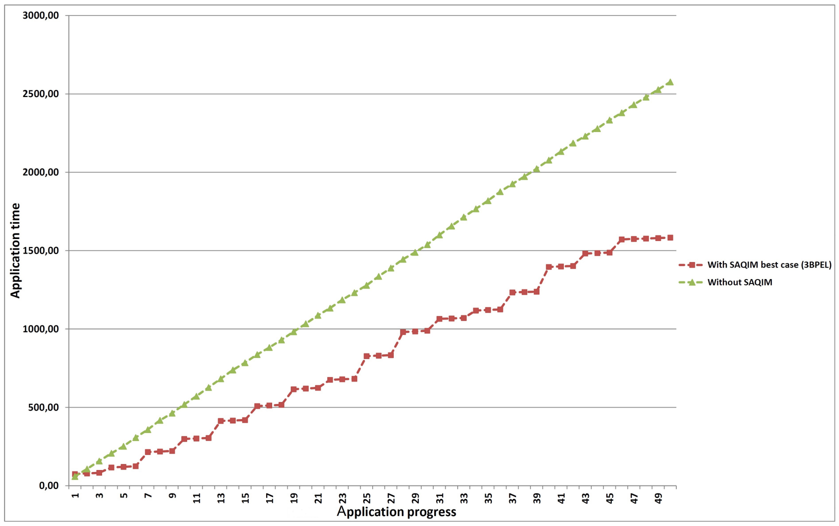


Figure 18: Best case time variation

data, which was generated from the real data to construct our dataset. The assessment of SAQIM cost effectiveness may be biased by the person's level of expertise and experience participating in the patterns catalog specification. The specification time assessment of the pattern scripts as well as the OCL architectural constraints may differ from one person to another, which may yield to different results. In our experiment to deal with the selection threat [19] which concerns the effect of natural variation in human performance, we selected a random group of Ph.D. students.

Construct validity: To increase construct validity, we avoided evaluation apprehension by separating students and by ensuring the confidentiality of the recorded results. Furthermore, the students were not told about the final goal of the experiment to avoid the experimenter expectancies threat, for example when measuring the time for document reading and comprehension of patterns.

Conclusion validity: To increase conclusion validity, we used regression analysis to get reliable estimation of the imaginary patterns specification time. Additionally, we selected students with relatively similar level of knowledge and background to limit the threat of random heterogeneity of subjects. Furthermore, we used an acceptable number of patterns in the simulation and we compared the results of SAQIM with those obtained using a well-known easy-to-use software tool (NetBeans BPEL designer).

External validity: An important threat to the external validity is the use of students as subjects. However, to increase the validity we involved

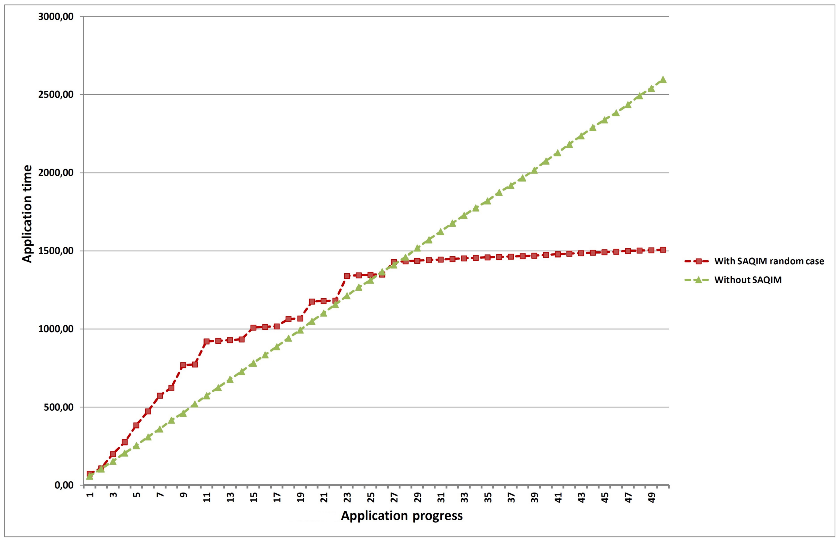


Figure 19: Random case time variation

Ph.D. students which have some development experience, and can easily play the role of architects in industry. Furthermore, despite the fact that the context in which the patterns catalog was developed is not part of a software development project it resembles to that of a real service-oriented software development situation.

5. Related Work

Many works have been proposed in the literature to address quality requirements integration in software architectures. Al-naeem et al [20] proposed *"ArchDesigner"*, which uses optimization techniques to determine optimal combination of design alternatives. In our work we use a simulation and feedback technique to help architects in the decision selection process to meet their quality goals. Architectural design decisions in our work are SOA Patterns which are applied in semi-automatic way, while in their work they are high level architecture design decisions (the choice of Java EE, for example).

Bass et al. [21] proposed the ADD method (Attribute-Driven Design) that follows an architectural design process guided by quality requirements. It uses the concept of attribute primitives, which are collections of components and connectors collaborating to satisfy some quality attributes. These attributes are documented as general scenarios. In [22], the authors proposed architectural tactics which are reusable building blocks, in the same spirit as the primitive attributes to guarantee quality characteristics in software architectural design. These works are quite similar to our work in the sense that they use reusable design decisions (attribute primitives and architectural tactics, we use SOA patterns) to address issues pertaining to quality attributes. However, they differ from our work in that they focus on the design stage, while we focus on the design and evolution stages. In addition, we give support to the architect to choose among several possible alternatives of a design decision the one that satisfies the best a given quality goal. Besides this, we help the architect in applying the selected design decision (the choice of SOA Patterns) in a semi-automatic fashion, and we give her/him assistance to make impact analysis.

In [23, 24, 8], the authors proposed a process which similarly to our work is based on the use of a Pattern catalog to document patterns as identified design decisions. However, their work differs in the way pattern selection and validation is performed. Indeed, in [23, 24] the authors use questions to help architects in choosing and validating the most appropriate patterns, whereas, we use an MCDM method in a complementary way with a quality-related impact analysis to select and validate patterns that best satisfy quality goals in a quality integration step. Additionally, our method offers a support to integrate patterns in a semi-automatic way.

In [25], the authors present an evaluation of the SOA patterns (except for the enterprise architecture patterns) introduced in [7] and their impact (positive or negative) on quality attributes defined in the S-Cube Quality Reference Model (QRM) for service-based applications [26]. They focused on architecture patterns from the catalog and distinguish between patterns with and without impact on quality attributes. Similarly to our work, they mapped some quality attributes addressed by SOA patterns (that could not be related to any quality attribute in the S-Cube QRM) to quality attributes from the ISO 9126 quality model. Their work is complementary to our work and could be helpful to the architect especially while building the patterns catalog. It could be used to deal with mapping between patterns and the quality attributes they impact as well as filtering only patterns having impact from those without impact on quality attributes.

Harrison et al [27] recommended the use of patterns and the integration of their impact on quality attributes in their specification as valuable information in order to increase their usefulness and help the architect satisfy quality attributes. As in [25] the authors investigated a quantitative evaluation of the impact of some architectural patterns (Layers, Pipes and Filters, Blackboard, etc.) from [6] on quality attributes. In our work, we identify automatically the impact through the solicitation of a quality-oriented assistance service that helps in diagnosing the consequences of any applied pattern on the other implemented qualities.

In [28], the authors presented a language named *"QoSL4BP"* and a tool named *"ORQOS"* that enable the architect to specify QoS constraints and some QoS injection mechanisms in Web Service orchestrations. This approach offers a way to integrate quality requirements as usable information at a functional and runtime level, while our work is positioned at the architectural level and incorporates quality requirements as reusable solutions (SOA Patterns). This approach is complementary to our work, since our work deals with quality requirements as architectural design decisions providing well known recurrent solutions which are used to generate designs encompassing quality requirements, and not as extra information which are exploited at a post-deployment time.

In [29] the authors presented an approach to Web service (WS) modeling, discovery and selection. They use an Intentional Service Model (ISM) which they enhance with quality aspects to configure the WS discovery and selection process. The selected services satisfy some quality requirements. In our work quality requirements are goals to be achieved in the service orchestration and contribute in its construction. Their work is then complementary to our work.

In [30], the authors present an approach, called *"AQUA"*, to quality achievement at architectural level based on design-decision making. They used an evaluation contract (between users and software architects) for quality attributes identification, then a process to manually find high level architectural design decisions achieving these quality attributes. In our work, design decisions are identified and proposed to the architects in a catalog as patterns (for SOA) which include in their specification the quality attributes they concretize, the way they should be applied, and the way they can be checked (constraints). The authors of this paper used a decision graph transformation strategy to analyze the impact of applying a design decision alternative, whereas, we simulate the application of a selected collection of patterns (design decision alternatives) and assist the selection (MCDM method) of the most appropriate pattern (semi-automatically), then report its impact (automatically) to the architect.

In [31] the authors proposed an approach to WS composition that satisfy quality requirements. The result of the composition in their work is a sequence of invocations to services that satisfy dynamic quality attributes achieved at runtime (e.g., response time). In our work we produce service orchestrations which embody more complex BPEL modeling elements (compared to sequences). In addition, in our work we are interested more particulary in static quality attributes (e.g., portability) integrated at design time.

6. Conclusion and Future Work

In a software development project, quality requirements are important software artifacts that are mainly satisfied at software architecture design time [22]. Architects are thus the software developers who are responsible for taking design decisions in order to satisfy this kind of requirements. One of the most common design decisions at this stage of a development process is the choice of an architectural style or design pattern. As catalogs of these well-known recurrent design decisions have been proposed in the literature and practice of software engineering (provided mainly with informal descriptions), we argue in this work that such catalogs can be documented in a (more or less) structured, automatically checkable and semi-automatically processable way. Such documentation is then operated in order to assist architects in satisfying quality requirements by suggesting to them the "most" appropriate patterns. By "most" appropriate pattern, we mean a pattern: i) that satisfies the more the tackled quality attribute (the pattern that gives the best scores for the evaluation criteria), and ii) that affects the less the other quality requirements, already satisfied and documented in the software architecture (through the use of the quality impact analysis). We consider in our work a specific kind of software architectures, which are service-oriented ones, and we deal with a particular specialization of this kind of architectures which are Web service orchestrations concretely defined as BPEL processes. In summary, the main contribution of this work is a method for quality integration in Web service orchestrations using a catalog of SOA patterns.

As perspectives to our work, we would like to define a simulation (decision) system to study the effect that yields the application of all possible initial combinations of selected patterns (that implement the required quality attributes specified in the NFRs). The aim is to generate all possible patterns application sequences, simulate their application, and then record their impact on the embodied qualities. Then, we look for the best application sequence which gives a service orchestration with the minimum effect on the qualities. We believe that the chosen sequence to embody the desired quality attributes is important and yields to different results (*i.e.* a service orchestration with different qualities or differently affected qualities and with different design costs). Another future work we are considering is to evaluate SAQIM as a quality integration micro-process by addressing each of its steps. More particularly, we are investigating the validation of the pattern selection step as well as the quality impact analysis step. For example, to evaluate the weighted sum model for pattern ranking, a sensitivity analysis [**?**] on the decision criteria weights and the criteria values could be introduced for studying and increasing the trustworthiness about the provided decision on patterns ranking.

7. References

[1] Web services business process execution language specification, version 2.0, OASIS Website: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html (April 2007).

[2] O. M. Groupe, Business process model and notation (bpmn) specification, version 2.0, OMG Website: http://www.omg.org/spec/BPMN/2.0/PDF (January 2011).

[3] T. Zernadji, C. Tibermacine, F. Cherif, Processing the evolution of quality requirements of web service orchestrations: a pattern-based approach, in: Proc. of WICSA'14, IEEE CS, Sydney, Australia, 2014.

[4] T. Zernadji, C. Tibermacine, F. Cherif, Quality-driven design of web service business processes, in: Proc. of WETICE/AROSA'14, IEEE CS, Parme, Italie, 2014.

[5] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Sofware, Addison-Wesley Professional Computing Series, 1995.

[6] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, Pattern Oriented Software Architecture: A System of Patterns, John Wiley & Sons, 1996.

[7] T. Erl, SOA Design Patterns, Prentice Hall, 2009.

[8] T. M. Ton That, S. Sadou, F. Oquendo, Using Architectural Patterns to Define Architectural Decisions, in: Proc. of WICSA/ECSA'12, Helsinki, Finland, 2012, pp. 196–200.

[9] OMG, Object constraint language specification, version 2.2, document formal/2010-02-01, Object Management Group Web Site: http://www.omg.org/spec/OCL/2.2/PDF (2010).

[10] L. Briand, Y. Labiche, M. D. Penta, H. Yan-Bondoc, An experimental investigation of formality in uml-based development., IEEE Transactions on Software Engineering 31 (2005) 833–849.

[11] T. U. Dresden., Ocl compiler web site., http://dresden-ocl.sourceforge.net/ (2009).

[12] E. Foundation, Model Development Tools website., http://www.eclipse.org/modeling/mdt/ (2009).

[13] P. C. Fishburn, Additive Utilities with Incomplete Product Sets: Application to Priorities and Assignments, Vol. 15, INFORMS, 1967.

[14] C. Tibermacine, T. Zernadji, Supervising the evolution of web service orchestrations using quality requirements, in: Proc. of ECSA'11, Springer-Verlag, Essen, Germany, 2011, pp. 1–16.

[15] C. Tibermacine, R. Fleurquin, S. Sadou, A family of languages for architecture constraint specification, In the Journal of Systems and Software (JSS), Elsevier 83 (5) (2010) 815–831.

[16] E. Triantaphyllou, B. Shu, S. Nieto Sanchez, T. Ray, Multi-Criteria Decision Making: An Operations Research Approach, Vol. 15, J. Wiley, New York, 1999, pp. 175–186.

[17] T. Saaty, The Analytic Hierarchy Process, McGraw-Hill, New york, 1980.

[18] N. Mead, White paper: Requirements prioritization case study using ahp, Tech. rep., Software Engineering Institute, Carneige Mellon University (2006).
URL `http://resources.sei.cmu.edu/asset_files/WhitePaper/2013_019_001_297260.pdf`

[19] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, Experimentation in Software Engineering, Springer, 2012.

[20] T. Al-naeem, I. Gorton, M. A. Babar, F. Rabhi, B. Benatallah, A quality-driven systematic approach for architecting distributed software applications, in: Proc. of ICSE'05, ACM Press, 2005, pp. 244–253.

[21] L. Bass, F. Bachmann, M. Klein, Quality attribute design primitives and the attribute driven design method, in: Proc. of PFE-4 2001, Springer-Verlag, Bilbao, Spain, 2001.

[22] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, 3rd Ed., Addison-Wesley, 2012.

[23] Z. Durdik, Towards a process for architectural modelling in agile software development, in: Proc. of QoSA'11, ACM, 2011, pp. 183–192.

[24] Z. Durdik, R. Reussner, Position paper: approach for architectural design and modelling with documented design decisions (admd3), in: Proc. of QoSA '12, New York, NY, USA, 2012, pp. 49–54.

[25] M. Galster, P. Avgeriou, Qualitative analysis of the impact of soa patterns on quality attributes, in: Proc of QSIC'12, IEEE, 2012, pp. 167–170.

[26] A. Gehlert, A. Metzger, Quality reference model for sba, Tech. rep., S-Cube Consortium (2009).

[27] N. B. Harrison, P. Avgeriou, Leveraging architecture patterns to satisfy quality attributes, in: Proc. of ECSA'07, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 263–270.

[28] F. Baligand, D. Le Botlan, T. Ledoux, P. Combes, A language for quality of service requirements specification in web services orchestrations, in: Proc. of ICSOC'06, Springer-Verlag, 2006.

[29] M. Driss, N. Moha, Y. Jamoussi, J.-M. Jzquel, H. H. B. Ghzala, A requirement-centric approach to web service modeling, discovery, and selection, in: Proc. of ICSOC'10, Springer-Verlag, 2010, pp. 258–272.

[30] H. Choi, Y. Choi, K. Yeom, An integrated approach to quality achievement with architectural design decisions, JSW 1 (3) (2006) 40–49.

[31] Z. Azmeh, M. Driss, F. Hamoui, M. Huchard, N. Moha, C. Tibermacine, Selection of composable web services driven by user requirements, in: Proc. of ICWS'11, IEEE CS, Washington DC, 2011.