Processing the Evolution of Quality Requirements of Web Service Orchestrations: A Pattern-Based Approach

Tarek Zernadji, Chouki Tibermacine, Foudil Cherif

Processing the Evolution of Quality Requirements of Web Service Orchestrations: a Pattern-based Approach

Tarek Zernadji
Computer Science Department
University of Biskra, Algeria
zernadji@yahoo.fr

Chouki Tibermacine
LIRMM, CNRS
and Montpellier II University, France
tibermacin@lirmm.fr

Foudil Cherif
Computer Science Department
University of Biskra, Algeria
foud_cherif@yahoo.fr

***Abstract*—Currently Web services remain one of the leading technologies for implementing components of service-oriented software architectures. One of the most frequent form of compositions of these entities is Web service orchestration. As any other software artifact, such service compositions are subject to an unescapable evolution (Lehman's first law of software evolution). Either for answering new user requirements, for adapting, for correcting or for enhancing the provided functionality or quality, an architect has to conduct some evolutions on the design of these artifacts. In this paper, we present a method which aims at helping software architects of Web service orchestrations in processing an evolution of quality requirements. This method introduces a template for describing quality evolution "intents". It then analyzes these intents and assists the architects in answering them by proposing some patterns. We consider in our work the postulate stating that quality can be implemented through patterns, which are specified with checkable/processable languages. Besides this, the method that we propose simulates the application of these patterns and notifies the architect with its consequences on the other implemented qualities.**

I. Introduction: Context and Motivation

Service-oriented architectures (SOA) provide regardless of particular implementation technologies, an architectural style for building service-based software systems. This style supposes the existence of a collection of reusable services, which deliver some functionality for client applications. One possible and quite frequent form of this kind of software architectures is Web service orchestration. BPEL processes are one of the most largely used technologies for making these orchestrations executable.

As any other software artifact, these software architectures must evolve during the system's life cycle (Lehman's 1st law [1]), and undergo changes that can harm the qualities originally planned by architects (Lehman's 7th law). One of the major causes of this alteration of quality is related to the phenomenon of "knowledge vaporization" [2], which is due to the fact that most decisions made during the construction of software architectures remain implicit (undocumented). The lack of information about previously made decisions can lead architects to accidentally affect them and consequently alter the qualities they implement. In a previous work [3], we have proposed an approach to address the problem of "knowledge vaporization" by documenting major design decisions and their rationale (quality requirements) and use such documentation for supervising the architectural evolution of Web service orchestrations.

In this paper, we propose a process that provides a systematic assistance to architects during the evolution of quality requirements of Web service orchestrations. Before starting the process, the architect should first make an architecture diagnosis and gather some information that feed the first step. We propose a template for describing quality evolution "intents" in order to enable the specification of this information (Section II-A). Then, such intent descriptions are processed (Section II-B) in order to propose to the architect some patterns for helping her/him to take design decisions. We argue in this work that for answering quality evolution intents, an architect can have as a design decision the selection of an SOA pattern. Our process is thus based on a documentation of design decisions as SOA patterns and their rationale as the quality attributes they implement. This process aims more precisely at helping an architect in choosing the well suited pattern to apply on her/his architecture. It uses a set of evaluation criteria and a quality impact analysis for that purpose (Section II-D). The architect is then assisted in a semi-automatic way to apply the selected pattern (Section II-C) thanks to reusable and customizable scripts defined using a scripting language for Web service orchestrations, named WS-BScript, which is introduced in this paper. The process ends by asking the architect to document the new design decisions (Section II-G) made into her/his architecture so that future evolutions can be assisted in the same way. Before concluding and presenting some perspectives to our work, we make an overview of the related work (Section III).

II. Proposed Approach

Through this process the architect is assisted to: i) make concrete changes leading to a new service orchestration, and ii) perform this with minimal negative effects on existing qualities. The process steps are detailed in the following sections.

A. Evolution Intent Specification

The architect should specify the needed information according to a template described in Table I. She/he provides in this template the quality attribute targeted by this evolution activity (*i.e.* the architect wants to implement in the

TABLE I. TEMPLATE FOR QUALITY EVOLUTION INTENT DESCRIPTION

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

orchestration). We adopt at the top level of our specification the ISO 9126[1] quality model. We consider in our work 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. Additionally, the architect should identify the architectural regions which are the main architectural elements (or sets of these elements) in the BPEL process concerned by the changes. Besides this, the architect has to indicate the evolution kind by indicating if she/he wants to add (a new), enhance (an existing), weaken, or withdraw (an existing) quality attribute. 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. 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 evolution kinds (add or enhance), this section is left empty.

B. Evolution Intent Analysis

The evolution intent specification is analyzed, and depending on the evolution kind indicated in this specification two cases are distinguished. These are detailed in the following subsections. The proposed process is based on an "SOA Patterns Catalog", where each pattern is specified according to a specific structure shown in Table II.

1) Processing the Evolution 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. The patterns catalog is automatically analyzed and a collection of patterns related to the targeted quality is identified[2] and proposed to the architect.

As depicted in Table II, the pattern's specification includes a "name" with a simple description of its role, the "quality attribute" the pattern implements, an "architectural script" which describes the way it should be applied in the orchestration, and finally "architectural constraints" which are a formal specification of the pattern and allow the checking of its presence or absence in the orchestration.

[1]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

[2]A quality attribute may be implemented by applying several patterns in different ways.

TABLE II. PATTERN STRUCTURE SPECIFICATION

Pattern Name	The identifier of the pattern and a simple textual description of its role.
Quality Attribute	The ISO 9126 quality characteristic or sub- characteristic that is implemented by the pattern (or concrete quality attributes).
Architectural Script	The set of parameterized actions that indicate the way the pattern can be applied on the architecture. Actions are formalized using a scripting language for Web service orchestration reconfiguration.
Architectural Constraints	The list of parameterized constraints that enable to check if the orchestration is compliant with the pattern.

2) Processing the Evolution by Removing a Pattern: No patterns are proposed here, rather a cancellation of the pattern implementing the quality attribute is performed. This cancellation is automatically obtained from the scripts for a pattern application.

C. 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 called "*WS-BScript*". WS-BScript is a lightweight DSL that enables the patterns catalog administrator, whose responsibility is to feed the patterns catalog, 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[3]. A script declares a set of parameters (BPEL orchestration elements), which represent the scope of the architectural actions.

In this step of the process, the architect will apply one or several predefined[4] 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).

D. 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 Multi-Criteria Decision Making (MCDM) method, named "WSM" [4] (Weighted Sum Model), to evaluate a number of SOA pattern alternatives and determine the one that best satisfies the architect in a quality requirement evolution step, and ii) the solicitation of a quality-oriented assistance service that helps in diagnosing the consequences of any applied pattern on the other implemented qualities.

For the first 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, having the best degree of satisfaction for the targeted quality attribute, and is the most suitable to the architect preferences (context

[3]The complete specification can be found here: https://sites.google.com/site/wsbscript/ws-bscript-specification

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

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) Satisfaction degree of a pattern for a quality attribute (C_2);
  3) Context-Suitability of the pattern (C_3).

For our evaluation purpose using the "WSM" method, we chose to normalize the aforementioned criteria according to the scale proposed in [5]. The later gives eleven scores ranging from 0.045..0.665, 0.745, to 0.955 and their corresponding linguistic terms from *"Exceptionally low"*..*"High"*, *"Very high"*, to *"Exceptionally high"*. This normalization allows us to deal with a single-dimensional case (all the units are the same) of the MCDM problem which fits well the use of "WSM" method. 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 [4]:

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

Weights wj represent the importance of each criterion according to the architect's preferences in the evolution process (also normalized according to the scale). a_{ij} is the value of an alternative "i" (pattern) in terms of a decision criterion "j". We note here that the patterns in the catalog are previously documented by the architect according to the model proposed in [3]. This model introduces some fine-grained information namely, the criticality degree (a_{iC1}) of a quality attribute, the formalization degree, and the satisfaction degree (a_{iC2}). The documentation is enriched with a context-suitability degree (a_{iC3}), which is specified and documented at evolution time because it depends on the pattern's suitability to a given situation and to the orchestration. This degree cannot be reused in different orchestrations. It can however be reused in the future evolutions of the same orchestration.

The second 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-like constraints that we used to specify parameterized architecture constraints [6] for Web service orchestrations. These constraints are defined using OCL and navigate in a metamodel of BPEL. They serve to verify if an architecture conforms to the pattern or not.

E. New Patterns Definition

It is on the responsibility of the architect to validate its choice of a specific pattern or to reject it. If the architect is not satisfied with any of the proposed patterns, then she/he can define new patterns (specialization of existing patterns, for example), which she/he is asked to document according to the proposed structure (Table II). 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 "Patterns Application" step to simulate the effect of the new catalogued pattern.

F. Pattern Cancellation

The architect may want to enhance a quality attribute not by adding a new pattern that implements the quality, or by replacing an existing pattern by another one which implements better the quality, but by eliminating or weakening a given quality attribute. In this case, the process execution takes another path. Thus, if the specified kind in the evolution intent is to withdraw or weaken a quality attribute, 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*" interpreter) 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[5].

G. 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 patterns) that was made to build the architecture. In addition, the architect has to complete a part of this documentation, namely the criticality degree of the quality attribute the pattern implements, the satisfaction degree of the pattern for the quality attribute, the formalization degree of the pattern, and also the related qualities of the quality attribute. This information is necessary for the evolution assistance especially in the patterns selection process (quality impact analysis step).

III. Related Work

Many works have been proposed in the literature to address quality requirements integration in software architectures. Alnaeem et al [7] proposed *"ArchDesigner"*, which use optimization techniques to determine optimal combination of design alternatives. We use a simulation and feedback technique at the evolution stage 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).

In [8], [9] the authors use reusable design decisions namely attribute primitives and architectural tactics, we use SOA patterns. However, they focus on the design stage, while we focus on the evolution stage. 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

[5]A complete list of these rules can be found in: https://sites.google.com/site/wsbscript/ws-bscript-cancellation-rules

quality goal. Besides this, we help the architect in applying the selected design decision in a semi-automatic fashion, and we give her/him assistance to make impact analysis.

In [10], [11] the authors use a Patterns catalog to document patterns as identified design decisions. However, their work differs in the way pattern selection and validation is performed. Indeed, in [10] they use questions to help architects in choosing and validating patterns, whereas, we use an MCDM method in a complementary way with a quality-related impact analysis to select and validate patterns. Additionally, our process offers a support to integrate patterns in a semi-automatic way.

In [12], similarly to our work they mapped some quality attributes addressed by SOA patterns [13] (that could not be related to any quality attribute in the S-Cube Quality Reference Model (QRM) [14]) 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 [15] investigated as in [12] a quantitative evaluation of the impact of some architectural patterns (Layers, Pipes and Filters, Blackboard...) from [16] 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 [17], the authors 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 which are used to generate designs encompassing quality requirements, and not as extra information which are exploited at a post-deployment time.

In [18], the quality attributes and the high level architectural design decisions achieving them are identified manually. In our work, design decisions are identified and proposed to the architects in a catalog as patterns (for SOA). They used a decision graph transformation strategy to analyze a design decision impact, whereas, we simulate the application of a selected collection of patterns and assist the selection (MCDM method) of the most appropriate pattern (semi-automatically), then report its impact (automatically) to the architect.

IV. Conclusion and Future Work

We argue in this paper that catalogs such as [19], [16], or [13] of design patterns can be documented in a (more or less) structured, automatically checkable and semi-automatically processable way. Such documentation is operated by a process that we specified in this paper, and whose main goal is to assist architects in processing the evolution of quality requirements by suggesting to them the "most" appropriate patterns: i) that respects the more the evolved 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 deal in our work with a particular specialization of service-oriented software architectures, which are Web service orchestrations concretely defined as BPEL processes.

As perspectives to our work, we would like to enhance the organization of the catalog of patterns. Instead of a flat organization, we want to define a hierarchical one, built using some classification techniques like FCA (Formal Concept Analysis [20]). In this way, we can easily look for substitutable patterns which can be proposed together to the architect in the process. Besides this, we plan to integrate in the proposed process an impact analysis activity on the business logic aspect, thus evaluate also the impact on the existing functionality implemented in the software architecture.

References

[1] M. M. Lehman and L. A. Belady, Eds., *Program evolution: processes of software change*. Academic Press Professional, Inc., 1985.

[2] A. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions," in *Proc. of WICSA'05*. IEEE CS, 2005, pp. 109–120.

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

[4] P. C. Fishburn, "Additive utilities with incomplete product sets: Application to priorities and assignments," *Operations Research*, 1967.

[5] S.-J. J. Chen and C. L. Hwang, *Fuzzy Multiple Attribute Decision Making: Methods and Applications*. Springer-Verlag, 1992.

[6] C. Tibermacine, R. Fleurquin, and S. Sadou, "A family of languages for architecture constraint specification," *In the Journal of Systems and Software (JSS), Elsevier*, vol. 83, no. 5, 2010.

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

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

[9] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice, 2nd Ed.* Addison-Wesley, 2003.

[10] Z. Durdik and R. Reussner, "Position paper: approach for architectural design and modelling with documented design decisions (admd3)," in *Proc. of QoSA '12*, 2012.

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

[12] M. Galster and P. Avgeriou, "Qualitative analysis of the impact of soa patterns on quality attributes," in *Proc of QSIC'12*, 2012.

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

[14] A. Gehlert and A. Metzger, "Quality reference model for sba," S-Cube Consortium, Tech. Rep., 2009.

[15] N. B. Harrison and P. Avgeriou, "Leveraging architecture patterns to satisfy quality attributes," in *Proc. of ECSA'07*, 2007.

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

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

[18] H. Choi, Y. Choi, and K. Yeom, "An integrated approach to quality achievement with architectural design decisions," *JSW*, vol. 1, 2006.

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

[20] B. Ganter and R. Wille, *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag, Inc., 1999.