A Declarative Approach for QoS-Aware Web Service Compositions*

Fabien Baligand[1,2], Nicolas Rivierre[1], and Thomas Ledoux[2]

[1] France Telecom - R&D / MAPS / AMS,
38-40 rue du general Leclerc, 92794 Issy les Moulineaux, France
{fabien.baligand,nicolas.rivierre}@orange-ftgroup.com
[2] OBASCO Group, EMN / INRIA, Lina
Ecole des Mines de Nantes,
4, rue Alfred Kastler, F - 44307 Nantes cedex 3, France
thomas.ledoux@emn.fr

Abstract. While BPEL language has emerged to allow the specification of Web Service compositions from a functional point of view, it is still left to the architects to find proper means to handle the Quality of Service (QoS) concerns of their compositions. Typically, they use ad-hoc technical solutions, at the message level, that significantly reduce flexibility and require costly developments. In this paper, we propose a policy-based language aiming to provide expressivity for QoS behavioural logic specification in Web Service orchestrations, as well as a non-intrusive platform in charge of its execution both at pre-deployment time and at runtime.

1 Introduction

BPEL language provides abstractions and guarantees to easily specify safe service compositions, but its expressivity is limited to functional concerns of a composition, implying that architects have to handle other concerns, such as QoS management, by other means. QoS management, in the context of Web Services, relates to a wide scope of properties such as performance, availability, price or security. To guaranty the QoS of a relationship between a customer and a service provider, a Service Level Agreement (SLA) that contains guarantees and assumptions is negotiated.

Dealing with QoS in service compositions faces numerous challenges both at pre-deployment time and at runtime. At pre-deployment time, architects have to guaranty the QoS properties of the composite services and have to find local services whose QoS satisfies to the global QoS. As discussed in [5], dealing with the QoS properties combinatory is a complex task. Current works [2,4] focus either on a bottom-up approach, that deduces the QoS of the composite service out of local services QoS and the composition structure, either on a top-down

* This work was partially supported by the FAROS research project funded by the French RNTL.

B. Krämer, K.-J. Lin, and P. Narasimhan (Eds.): ICSOC 2007, LNCS 4749, pp. 422–428, 2007.

approach, that aims to find a set of local services satisfying to the QoS of the composite. However, both ways do not take into account architects advanced requirements. For instance, the architects may want to specify QoS of some parts of their orchestrations and may require that some local services are discovered to match the global QoS. At runtime, QoS of local services is likely to vary, and the orchestration client may use various paths in the BPEL flow execution. Such variations lead to QoS variations of the composite service that need to be dynamically counterbalanced. Also, QoS mechanisms such as security, reliable messaging or transaction, which rely on WS-* protocols, are major features that must be addressed.

Because BPEL language does not provide expressivity for QoS management, architects cannot easily specify QoS requirements and behavioural logic in their orchestrations. Instead, they handle QoS management at the message level, using different frameworks and languages: some specific platforms take care of SLA documents, while SOAP filters contain QoS mechanisms implementation and that BPEL engines may offer basic QoS features. Making all these frameworks work together leads to code that lacks flexibility and portability, that decreases loose coupling nature of the composition, and which is error-prone. To provide the required expressivity for QoS management at the composition level, we propose a language accurately targeting parts of the BPEL orchestrations.

In this paper, we present our approach that aims to be non-intrusive with already existing infrastructures and languages. This approach offers a policy-based language, called "QoSL4BP" (Quality of Service Language for Business Processes), and a platform, namely "ORQOS" (ORchestration Quality Of Service). The latest version of ORQOS platform has not been fully implemented yet, but already existing components of previous versions have been used for proof of concept purposes. The remainder of the paper is organized as follows: Section 2 describes QoSL4BP language structure and primitives, Section 3 details the three steps of ORQOS platform process, Section 4 illustrates our approach with a scenario and Section 5 discusses the related works.

2 QoSL4BP Language

Design. To allow a seamless integration with BPEL language, and to increase reusability and portability of our language, QoSL4BP language was designed as a policy-based language. A policy consists in a declarative unit containing the adaptation logic of a base process. It is commonly agreed that a policy contains objectives to reach and actions to perform on a system. In our context, the BPEL orchestration is divided into sub-orchestrations (called scopes), each scope being addressed by a specific QoSL4BP policy, in order to allow the architect to address well-delimited systems of the orchestration, to decompose the QoS aggregation computation problem (described in section 3), and also to increase policies reusability. Thus, QoSL4BP policies contain both static and dynamic QoS behavioural logic, hence allowing architect to specify QoS constraints and adaptation logic over scopes of the orchestration.

Structure. The structure of a QoSL4BP policy is composed of three sections, as shown on Figure 1: The "SCOPE" section specifies the BPEL activity (basic or structured) targeted by the policy, "INIT" section contains the initial QoS settings of the scope, used at pre-deployment time, and "RULE" section embodies Event-Condition-Action (ECA) rules. This section is performed at runtime while the composition performs within the scope targeted by the policy.

```
POLICY policy_name = {
    SCOPE = { BPEL activity targeted by the policy }
    INIT = { scope initial QoS settings }
    RULE = {
        (Condition)? -¿ (Action)+
    }
}
```

Fig. 1. QoSL4BP Policy Template

Primitives. QoSL4BP language offers a limited set of context access and action primitives, as illustrated on Figure 4. Conditions of rules are formed by testing the context access primitives and can be composed with the usual boolean operators. Context access primitives returns QoS data collected both at the service and at the composition levels: **REQUIRE** and **PROVIDE** primitives give information about the QoS mechanisms required and provided by a service; **SLAVIOLATION** and **SCOPEVIOLATION** primitives respectively detect if a SLA is violated and if the scope QoS initial settings are violated; **USER**, **EXCEPTION**, **RATE** and **LOOP** primitives respectively returns information about the user, QoS exceptions, branch rate of use in a switch activity, and number of loops in a while activity. Action primitives allow the architect to specify QoS behavioural logic of the orchestration: **PERFORM** and **PROCESS** primitives enforce QoS mechanisms for outbound and inbound SOAP messages; **SELECT**, **RENEGOTIATE** and **REPLANNING** primitives respectively enable to select a concrete service to use for an abstract service, to renegotiate a concrete service to match an abstract service, and to perform QoS replanning to satisfy to the scope QoS initial settings; **FAIL** and **THROW** primitives allow to throw QoS exceptions to the customer and inside the orchestration.

3 ORQOS Platform Process

ORQOS platform process includes three steps. First, ORQOS platform statically singles out a set of concrete services to match the abstract services of the orchestration whose QoS aggregation satisfies to the SLA of the composite service, then it modifies the BPEL document to introduce monitoring activities at pre-deployment time, and finally ORQOS performs QoS adaptation at runtime.

QoS Planning. Let k be the number of services of the orchestration, and let n be the number of potential concrete services that can implement each of services of the orchestration, then the number of potential configurations to evaluate is n^k, making the problem NP-hard [5]. To bring answers to these issues, ORQOS decomposes, using policies scopes, the computation of the composite service into multiple computations at some "sub-composite" levels, and recomposes the solutions afterwards. For decomposition, QoS initial settings of QoSL4BP policies are considered both as expectations (for the local services contained in policies scopes) and as guarantees (when evaluating the global QoS of the orchestration). Thus, as shown in Figure 2, smaller aggregations are tested against the QoS initial settings of QoSL4BP policies, then the QoS aggregations of sub-composite services are tested against the SLA of the composite service. Therefore, let p be the number of policies, let $c_i (i \in [1; p])$ be the number of services included in the scope of policy i, and let c_0 be the number of services which are not included in any scope of policies, then the number of potential configurations ORQOS has to evaluate is $\sum_{i=0}^{p} n^{c_i}$, which is in $\Theta(n^{max(c_i)})$, meaning that, with a set of appropriate scopes, testing each configuration with aggregation techniques, such as presented in [2], becomes affordable.

Monitoring Sensors Insertion into BPEL. The second step of ORQOS platform processing consists in inserting sensor activities at relevant places into the BPEL document, to monitor performance of orchestration scopes and to inform ORQOS at runtime. Such sensors are standard "invoke" activities that monitor scope QoS, BPEL execution paths, and exceptions. They call an ORQOS sensor manager interface, hence allowing ORQOS to collect data at runtime. As shown on Figure 2, sensors are inserted into the BPEL document according to the instructions specified in the "RULE" section of QoSL4BP policies. After this transformation step, the BPEL document can be deployed on any BPEL engine.

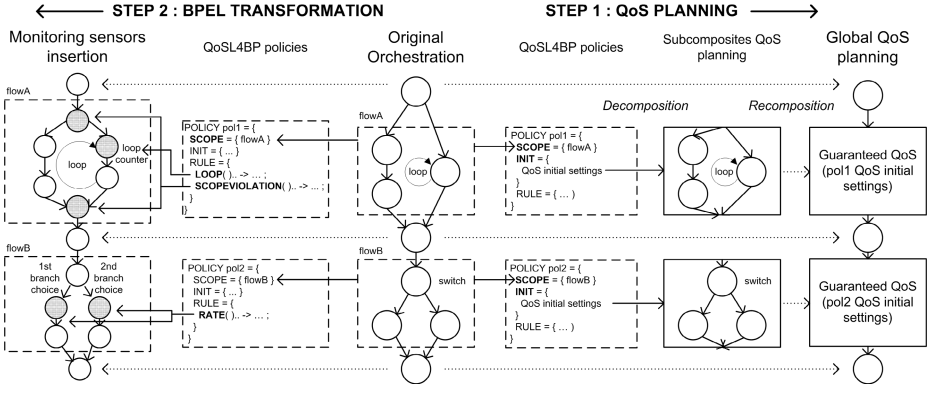

Fig. 2. Pre-deployment Process Steps

QoS Adaptation at Runtime. Once the orchestration is deployed, the BPEL engine exposes both a WSDL interface and an SLA offer for customers to invoke the composite service. As can be seen on Figure 3, a proxy layer has been added for SLA monitoring, for WS-* mechanisms enactment, and for flexible dynamic service binding. Thus, the proxy acts both as a sensor and an actuator in partnership with ORQOS platform. Meanwhile, ORQOS platform is in charge of processing the rules contained in QoSL4BP policies. It receives information both from the proxy (SLA violation, usage of orchestration customers) and from the BPEL engine via the sensors inserted at pre-deployment time . Upon satisfaction of any of the rules conditions, the corresponding actions are performed, hence allowing QoS to be readjusted at runtime.

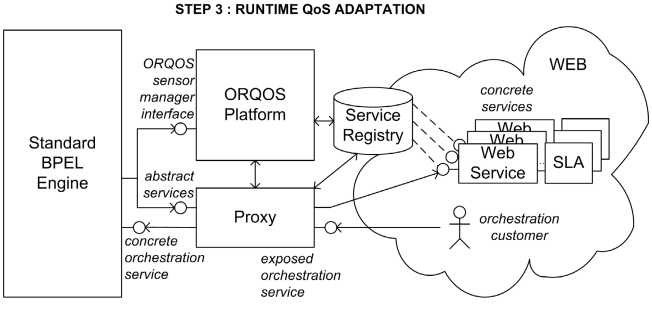

Fig. 3. Runtime QoS Adaptation Step

4 Illustrative Scenario

Depicted in Figure 4, the "Personal Medical Records" scenario illustrates a Web Service orchestration called by a doctor to get medical records of a patient. Upon reception of the request, some registry services are called in parallel. Next, a records management service that stores the medical records is called. Then, a "while" activity calls a "fetcher" service to collect the corresponding medical items. Finally, a folder containing the list of items is assembled by an "archiver" service, and is sent by an FTP delivery or a mailing service.

Policy "guarantyFlow" targets the flow("registry") activity, describes the QoS settings of the scope (response time below three seconds per request, throughput exceeding one hundred of requests per second) and specifies message encryption (using WS-Security) as well as a rule specifying scope QoS replanning if any service SLA is violated. **Policy "adapt2loop"** specifies a number of loops (five) for static computation. Depending on the number of loops performed at runtime, it renegotiates with the "fetcher" service or throws an exception in the orchestration. The "archiver" service can be implemented by two services ("ZIPService" and "RARService") that do not come with SLA. **Policy "noSLA"** specifies the expected QoS for static computation, and implements the service selection logic ("ZIPService" can hold a forty requests per second throughput while "RARService" shows better performance) depending on the

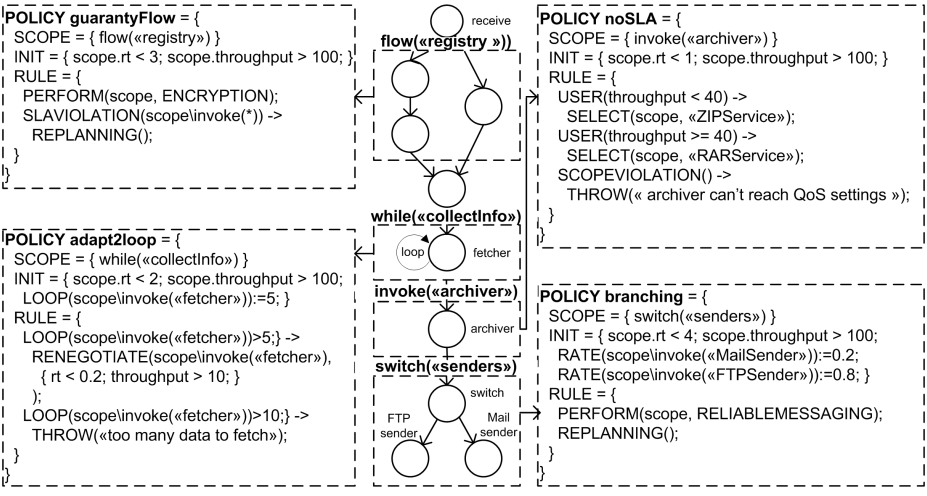

Fig. 4. "Personal Medical Records" Orchestration and QoSL4BP Policies

usage of orchestration customers. Also, it specifies that an exception should be thrown if the static QoS properties are violated. **Policy "branching"** contains the initial rates of use of the "switch" branches for static computation ("FTPSender" service is initially called four times more than the "MailSender" service). At runtime, WS-ReliableMessaging protocol is specified for each service of the scope and the QoS of services has to readjust according to the rates of use of the branches.

5 Related Works

In [3], the authors have elaborated a language named "Aspect Oriented for Business Process Execution Language" (AO4BPEL) that allows BPEL aspects specification calling non functional mechanisms, such as security (using infrastructural services that modify SOAP messages). This work is different from ours because the framework requires a purposely built BPEL engine, it uses an imperative language to specify extra-functional requirements, and it does not address performance requirements. In [1] the authors propose a policy assertion language, "WS-CoL" (Web Services Constraint Language), based on WS-Policy and specifically designed to be used in monitoring BPEL processes. The approach is similar to ours in that a non intrusive manager, in charge of the evaluation of policies, is called by standard BPEL invoke activities. However, the authors focus on monitoring and only consider security assertions (WS-SecurityPolicy).

In [2] the authors propose a QoS prediction algorithm consisting of a set of graph reduction rules applied to the composite constructs of a workflow. Upon only one atomic service remains, QoS properties corresponding to the process are exposed. We use a similar reduction approach as it enables fast workflow QoS estimation, but we adapt it to take into account dynamic replanning and

objective functions, such as architects constraints. In [4] the authors propose a method allowing to select concrete services so that QoS aggregation is optimized and that the global constraints are satisfied, either by a local optimal selection for each individual abstract service, either by global planning optimization of the composite service as a whole but at the price of higher computational cost. As discussed in [6,5], such solution cannot be considered when the number of abstract or candidate services is large. Our approach is similar but takes advantage of the orchestration decomposition process to apply global planning to limited parts of the workflow. Although such decomposition might lead to suboptimal solutions, it significantly improves the performances.

6 Conclusion

Although QoS management in service compositions is crucial, architects still lack means to address this concern in a flexible and reusable manner. Our solution aims to provide expressivity to address the QoS management concerns of service compositions. Our first contribution is the QoSL4BP language, allowing to specify QoS policies over scopes of BPEL orchestrations. Execution of the QoSL4BP language is performed by the ORQOS platform, designed to be non intrusive with already existing infrastructures. At pre-deployment time, ORQOS guarantees the static QoS properties of the orchestration and singles out a relevant set of concrete services. At runtime, ORQOS platform monitors the execution environment and processes policies rules enabling QoS mechanisms management, SLA renegotiation and QoS exception handling.

References

1. Baresi, L., Guinea, S., Plebani, P.: Ws-policy for service monitoring. In: Bussler, C., Shan, M.-C. (eds.) TES 2005. LNCS, vol. 3811, pp. 72–83. Springer, Heidelberg (2006)
2. Cardoso, J., Sheth, A.P., Miller, J.A., Arnold, J., Kochut, K.: Quality of service for workflows and web service processes. J. Web Sem. (2004)
3. Charfi, A., Schmeling, B., Heizenreder, A., Mezini, M.: Reliable, secure, and transacted web service compositions with ao4bpel. In: ECOWS. Proceedings of the 4th IEEE European Conference on Web Services, December 2006, IEEE Computer Society Press, Los Alamitos (2006)
4. Zeng, L., et al.: Qos-aware middleware for web services composition. IEEE Trans. Softw. Eng. 30(5), 311–327 (2004)
5. Jaeger, M.: Optimising Quality-of-Service for the Composition of Electronic Services. PhD thesis, Berlin University of Technology (January 2007)
6. Yu, T., Lin, K.-J.: Service selection algorithms for web services with end-to-end qos constraints. In: CEC'04. CEC '04: Proceedings of the IEEE International Conference on E-Commerce Technology, Washington, DC, USA, pp. 129–136. IEEE Computer Society Press, Los Alamitos (2004)