Electronic Communications of the EASST
Volume 27 (2010)

Workshop über Selbstorganisierende, adaptive, kontextsensitive verteilte Systeme (SAKS 2010)

QoS-based Self-Management for Business Processes

Diana Comes, Michael Zapf, and Kurt Geihs

12 pages

Guest Editors: Klaus David, Michael Zapf
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

QoS-based Self-Management for Business Processes

Diana Comes, Michael Zapf, and Kurt Geihs

comes@vs.uni-kassel.de, zapf@vs.uni-kassel.de, geihs@vs.uni-kassel.de,
Distributed Systems Group
Universität Kassel, Wilhelmshöher Allee 73,
34121 Kassel, Germany

Abstract: Business processes are commonly implemented as compositions of Web Services, using the Business Process Execution Language (BPEL) as an orchestration specification. Business processes do not only require an appropriate setup but also need to be monitored throughout their runtime, especially when Quality-of-service (QoS) constraints have to be met. Monitoring results may be used for the automated reconfiguration and optimization of business processes.

We show how we achieve self-management based on QoS constraints within our system. The BPRules Language that we set up can be used to improve the QoS behavior of business processes by triggering appropriate management actions on the process. Also we propose a service selection strategy for the dynamic selection and replacement of services within business processes.

Keywords: Business processes, WS-BPEL, Self-Management, MAPE

1 Introduction

Web Services are the standard technology for implementing business tasks in heterogeneous IT environments. They embody the building blocks for cross-organizational business processes. In recent years, development environments for Web Services have appeared that enable domain experts and system designers to realize flexibly configurable business processes.

In order to form a consistent composition, Web Services have to match with each other according to functional and non-functional requirements. Functional criteria are basically covered by input, output, precondition, and effect. Non-functional requirements are commonly referred to as Quality-of-service parameters (QoS). For a client, QoS criteria of a business process may be particular interesting, especially when we consider response time or availability. Typically, these objectives are defined in service-level agreements.

Business processes are typically represented using a language like the Business Process Execution Language (BPEL) [8]. While monolithic applications may be tailored to predefined QoS constraints, and tests may be used to effectively verify the compliance, Web Service-based business process implementations have a high degree of distribution and a loose coupling of components of possibly different sources. Hence, an effective business process management requires a support for calculating and aggregating QoS values, and should also allow for a flexible configuration of rules for autonomous corrective actions.

A successful execution of business processes implies continuous monitoring of QoS at runtime but also immediate intervention in undesired situations, like when a service is down or not

responding in the desired time frame. In that case corrective actions need to operate on the business process to improve its QoS behaviour. One of these actions may be to select another service which provides better QoS and to replace the old service.

In this paper we describe how we manage business processes within our management system. The key feature in our business process management is a self-managing architecture which has been developed in the course of the ADDOaction project[1]. In order to describe corrective operations, we make use of the *BPRules* language which we briefly introduce. BPRules allows us to define rules on the business process level, associating QoS constraints to appropriate actions. Moreover, we propose a service selection strategy for the dynamic selection and replacement of services within business processes. Our selection strategy is customizable to the clients' needs. We will show how the service selection strategy can be specified with BPRules.

The paper is structured as follows: Section 2 provides a short introduction to business processes and their structures. We summarize the main features of BPRules, a language for specifying corrective actions in the self-management of business processes. We also explain the advantages of the service selection strategy and how it is applied. Section 3 describes the architecture of our system, pointing out the MAPE structure of the control loops. Finally we present some links to related work in Section 4 and our conclusions in Section 5.

2 Managing Business Processes by QoS

The management of business processes along constraints on the quality of service (QoS) requires to set up rules or policies which can express two aspects:

- *Evaluation*: What do we consider to be a good or a failing process?
- *Action*: What can we do to restore or improve the service quality?

In order to be able to specify these rules, we have to take into account how business processes are structured. This is usually described by documents formulated in the specification language WS-BPEL.

2.1 Business processes and self-management

Web Services, often provided by different partners, can be composed to one big service which realizes a business process. Over time, several composition languages were developed, like the Web Services Flow Language (WSFL) from IBM, the Web Services Conversation Language (WSCL) from W3C or XLANG from Microsoft. Business Processes can also be described by using the Business Process Model and Notation (BPMN) [13]. The focus of BPMN is on the visualization and notation of a business process. For executing the process, a language like WS-BPEL is required.

The WS-BPEL language has emerged as the standard technology for implementing business processes in a SOA. WS-BPEL permits specifying Web Service compositions by defining how interactions between given Web Services take place and is therefore an example of the *Programming-in-the-large* paradigm. A WS-BPEL process may consist of several activities, like

invocations of Web Services, control structures for defining loops (*while*, *for*, *foreach*, *repeatUntil*), or conditional activities (*if*, *switch*). A *sequence* is an activity block in which activities are sequentially processed, while a *flow* defines several parallel activites.

WS-BPEL processes may be utilized to represent and implement business processes. Compared with single services, where we have some freedom in choosing an appropriate service, adapting our application to it where required, services which are orchestrated in a business process need to be mutually compatible. Specifically, outputs of previously executed services must be processable by following services. More precisely, services must fulfill certain functional requirements, which requires them to comply by formal contracts like interfaces, describing the classic Input/Output/Precondition/Effect (IOPE) matching. The challenge for setting up business processes in that aspect is to select those services that are compatible with each other and which form a process with the desired inputs and outputs.

As an example, we can conceive a *bookshop process* for buying books in an online shop which we implement using WS-BPEL. In the process there are four atomic services envolved: a *stock service*, a *distributor service*, a *bookshop service*, and a *bank service*. The *bookshop process* starts with the arrival of a request from the client, which contains the list of books the client wants to purchase at the online bookshop. The list of books is checked by the *stock service*, whether all of the books are in stock. If the books are not in stock, the *distributor service* is invoked to purchase the missing books from a book wholesaler. If the *distributor service* returns several books with different prices, the books with the minimum price are selected to be bought. These books are put on the clients' bill and the *bookshop service* is invoked for creating the bill. Finally the *bank service* is called to perform the withdrawal from the client's credit card to the online shop account.

However, for practical usage, not only the functional requirements but also non-functional requirements are important, which determine the *quality* of parts or of the overall service. This becomes even more salient for distributed business processes which contain Web Services from different locations, all subject to local conditions and management. Network failures or services becoming unavailable or not responding in time are possible issues which undermine the user's trust in the complete process implementation; the malfunction of one single service may cause the failure of the entire process.

One promising concept for handling these issues is to keep available a set of functionally equivalent services and to replace failing services or services which cannot ensure the promised QoS anymore. However, in most cases, the user will just notice the *end-to-end* quality failing the expectations, without an indication which service actually causes the problems. As in most cases, the reason for some failing service quality cannot be immediately tracked down to one service, finding the troubled service may require prohibitively high efforts, which may again impair the trust of the client into the whole business process implementation.

2.2 BPRules

One of the basic concepts of the BPRules language are the handling of *sections* of a business process. We can easily identify sections within the process where a set of Web Services are composed to a larger component, like services under the authority of a single provider. Thus, defining sections within the process can reduce the complexity and so help to faster identify

process parts that cause problems. In essence, sections help us to reduce the complexity of the overall business process management and so make an automatic monitoring and management of complex processes actually feasible.

As *self-management* with respect to business processes we understand the application of management activities on business processes, like the exchange of component services, on request of some internal part of the execution environment. This request is based on the observation of current properties of the business process and appropriate evaluation and deduction of corrective actions. Ideally, such a system should be able to maintain certain properties (like QoS) without human intervention. Hence, a *self-managing business process* is a system comprising the business process definition, sensors for detecting the current state, a managing subsystem accessing management interfaces, and the execution engine.

Apart from detecting undesired situations, management actions need to be taken to improve the business process behaviour. For the specification of reactive management actions in self-managing business processes we propose a special language called *BPRules* [4]. We now briefly describe the concepts of the BPRules language and its processing within our system to demonstrate the contribution of the language to the self-management capabilities.

The management of our business processes consists of checking conditions and executing associated corrective actions which are formulated using BPRules. For each business process we can associate a *BPR document*. Mainly, a valid document defines the following information:

- *sections* of the business process;
- *rulesets*, which contain collections of *rules*;
- *periods* of time which determine the set of process instances for which a rule(set) applies to.

Apart from the possibility to formulate conditions and actions as rules for the management of business processes, the simple XML syntax of BPRules allows a business analyst to specify rules without requiring additional programming skills. The business analyst himself may take care to specify proper rules and to make sure that the application of the rules will improve the QoS of the process.

Management actions may range from just notifying the interested parties of certain events, over starting, stopping, or updating the process, to actions like selecting and replacing services with other versions that promise a better QoS. The BPRules language allows to change or update rules dynamically at runtime, to use references for QoS parameters and QoS constraints between sections, or to set rules on a subset of instances (like, for example, running instances).

For each business process we can associate a BPR document as shown in Figure 1. Here it becomes clear that dividing the process into several parts (*sections*) provides us with a better control on the business process by a tunable granularity. We can group multiple activities (e.g. all activities inside a *flow*, or a *sequence*) into a single manageable part. Sections are defined by declaring start and end activities by their names, or by referring to a structured activity with nested subactivities.

For our *bookshop* example we define a section and the associated QoS requirements which are desired to be met. Informally, we define:

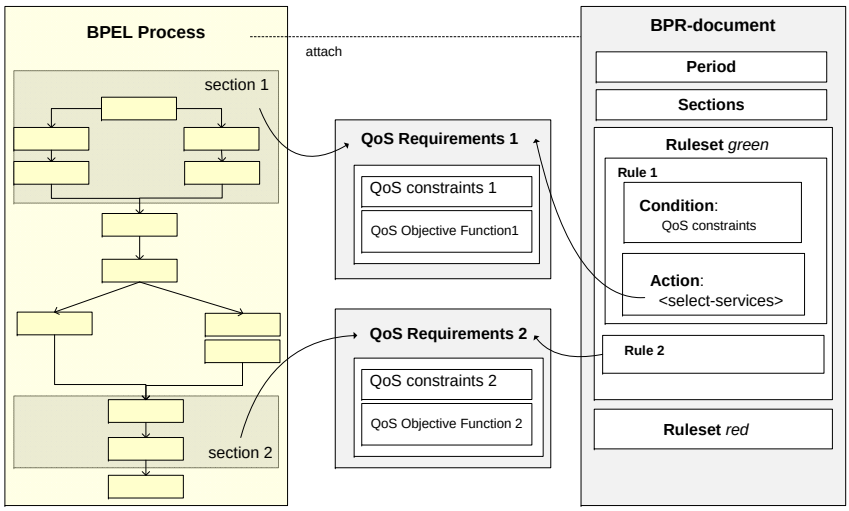

Figure 1: Business process and BPRules document

***section**: the distributor section*
targets multiple activities:
request the books to the distributor service, select the books with minimal price, buy these books from the distributor service
QoS requirements: availability > 0.98, cost < 20, *minimal* responsetime

In general, rules define a selection of instances which a rule applies to, the condition expression, and associated actions. If the condition constraints are met, the associated actions are triggered. Time periods may be defined to constrain the application of a rule to certain process instances (e.g. running at that moment).

Rulesets are bundles of rules which may be addressed as one element. For instance, when a certain alert level is reached, another collection of rules has to apply. Rulesets can be modified at runtime and they can be activated or deactivated as a result of an action. Any element like sections, periods, rulesets, rules, actions, or expressions can be referenced by its *id* within the document contributing to the reusability.

2.3 Service Selection strategy

One possible action as an instance of business process management is to replace a failing service; this entails the selection of an appropriate replacement. Thus, BPRules allows for the specification of a *service selection strategy* for the dynamic selection and replacement of services within business processes. The strategy permits selecting services for different sections or the entire process having different QoS requirements. The service selection strategy is customizable with respect to the selection method (selection algorithm) and the origin of the QoS values.

A service process is a composition of multiple services that are required in order to execute the service process. An *abstract service* represents the functionality of the desired service, and we assume that there are several *concrete services* that provide this functionality but which have

different QoS levels. In order to evaluate whether the entire service composition complies with the promised QoS level, we need to aggregate the QoS of the services that build up the service composition, using the algorithm from [5].

We illustrate the usage of the selection strategy by a simple example in listing 1, focusing on the rule definition. For saving space we simplify the structure and content of the example and use comments to informally describe the semantics. The rule states that

> **if** the running process instances (line 4-6) meet the undesired QoS constraints (availability < 0.7 or cost > 30) in section1 (line 7-9)
>
> **then** perform an action by selecting services from the service registry with the desired QoS requirements (line 16-27): availability > 0.95 and cost < 20 using the objective function: $f_{obj} = max(\frac{availability}{cost})$.

A service instance in a business process may be replaced if the service registry knows about other services which promise a better QoS. Within the *condition* part, constraints appear as criteria for service selection. The `<select-services>` tag (line 13-28) represents in BPRules the corrective action for service search and replacement.

What still needs to be specified are the criteria to select a service from a set of possible alternatives. This is specified within the QoS requirements (the `<qos-requirements>` element: line 16-27): Those services are selected from the service registry which fulfill the *QoS constraints* (the `<expression>` element: line 17-19) and optimize the given *objective function* (the `<objective-function>` element: line 21-26). In this example, the optimization criterion is an expression defined by the quotient of availability and cost. A preferred choice should yield a high value of this ratio; this can be set by the objective function (lines 21-26) which states that the QoS dimensions are subject to maximization.

In this way, every section may specify different QoS requirements. For example, when triggering the section of the bookshop process which involves the bank service, a maximum security is required, while for the distributor section we desire a minimal responsetime.

Furthermore, we want to be able to set the selection method. BPRules allows to specify the *selection algorithm* to be used. For instance, when we search n abstract services and each abstract service may have m concrete service realizations there are in total m^n combinations possible. Combining all concrete services that can realise the service composition leads to an combinatorial explosion; generally, the selection problem is NP-hard.

Selection algorithms may have very different complexity. In some situations, an optimal search may cause too high efforts (and thus delays) and is not desired, while other situations highly depend on a sophisticated algorithm. In BPRules we can specify the appropriate algorithm depending on the number of services to be searched. For example, when searching one service, a trivial search is sufficient, while in a search that envolves many services or the entire process, a more advanced search is needed, like an optimization algorithm using heuristics. Based on previous experience, the business analyst should be able to choose the appropriate selection method most suited for the particular case.

Selection strategies may also differ between set-up time and runtime. At runtime, a quick and effective solution is usually preferred to an optimal but slow strategy.

Various methods are proposed in the literature to solve the selection optimization problem,

like the *Integer Programming* approach [3], a *Genetic Algorithm* [2] or other heuristic algorithms [12],[6]. We can define which kind of strategy should be used for set-up or runtime. The appropriate selection method for a particular process is provided as an attribute (`method`, line 14) of the `<select-services>` action element. In this case we choose a heuristic selection (line 14: `method = "ALG.OPTIM_S"`). The `OPTIM_S` algorithm which we developed avoids the combinatorial explosian by setting an upper limit to the possible service variants that are chosen. The service variants are selected by a heuristic function.

```
1  <rule id="rule1">
2     <condition>
3        <constraints>
4           <select-instances>
5              <!-- the RUNNING instances -->
6           </select-instances>
7           <expression id="expsect1" applysection="section1">
8              <!-- undesired QoS constraints: availability < 0.7 or cost > 30 -->
9           </expression>
10       </constraints>
11    </condition>
12    <action>
13       <select-services serviceRegistry="http://registry:8097/services"
14           method = "ALG.OPTIM_S" methodClass="selection.OptimServiceImpl"
15           qosValues = "QoS.MONITOR" qosClass ="qos.QoSClass">
16         <qos-requirements>
17           <expression applysection="section1">
18              <!-- desired QoS constraints: availability > 0.95 and cost < 20 -->
19           </expression>
20           <!-- fobj = max(avail/cost) -->
21           <objective-function type="MAX" resultType="double">
22               <function type="Divide" resultType="double">
23                  <operand><QoSParameter>availability</QosParameter></operand>
24                  <operand><QoSParameter>cost</QosParameter></operand>
25               </function>
26           </objective-function>
27         </qos-requirements>
28       </select-services>
29    </action>
30 </rule>
```

Listing 1: Rule formulated in BPRules

Our selection strategy is customizable with regard to the origin of the QoS values. The QoS values of the services may be retrieved from several sources. When searching for services in the service registry, some clients could be interested in the measured QoS values while others would consider the values promised in the service level agreements (SLA). In our example we chose the QoS values that were monitored within our system (attribute `qosValues="QoS.MONITOR"` line 15).

3 Architecture for self-managing business processes

A crucial challenge of realizing complex business processes by orchestrating Web Services is to manage the complete process during its lifetime, especially when the process contains many different Web Services from various sources, and when it is more than just a sequential processing. To enhance the manageability, we argue that self-management capabilities of the system are

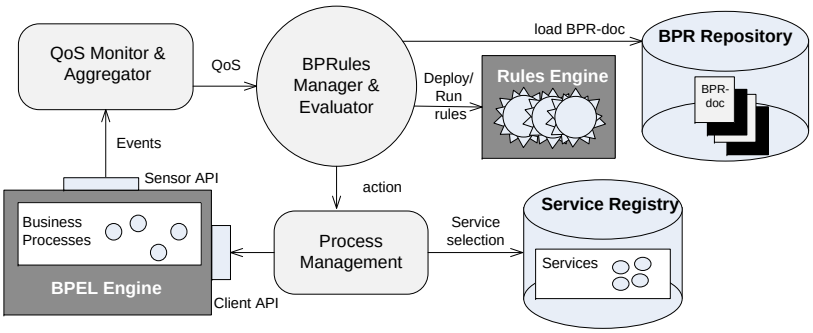

Figure 2: The management system

indispensible.

Usually, systems with self-stabilizing properties, which are the result of internal self-organizing, self-managing, or other "self" processes, include a component which supervises the *system under control*, which is, in our case, the complete business process, but also parts of it. This component requires sensors which allow to retrieve values that lead to appropriate actions, exerted by the actuator parts.

The controlling entity may be decomposed according to the subtasks required for the overall control task. One popular decomposition is known as the *MAPE* cycle, commonly cited in conjunction with IBM's Autonomic Computing initiative [10]. The MAPE cycle consists of four significant conceptual parts. If these parts can be automated, they form an *intelligent control loop*:

- **Monitor**: Queries the *sensors* and processes their inputs to some appropriate format. This allows the system to collect the details it needs.
- **Analyse**: Gets the data from the monitor part and evaluates them according to predefined or evolved functions. The analysis part is required to detect the current state of the system to determine if there is a need for action.
- **Plan**: From the current state, suitable actions are selected in the case that the state is not the target state. These actions which specify the necessary changes should bring the system into a new, "better" state.
- **Execute**: Perform the actions which have been collected in the planning component by means of certain *effectors* or *actuators*.

The execution of business processes is controlled by an execution engine. In our environment we employ the *Oracle BPEL Process Manager* (PM) [9] for executing business processes. The Web Services themselves are executed on the *Oracle Application Server OC4J*. By using the *Sensor API* and *Client API* of the Oracle BPEL PM, we can provide suitable sensors and effectors for attaching our system to a BPEL execution environment, implementing the control loop. The main constituents of our new self-management system are the following components:

- *QoS Monitor and Aggregator component*: monitors the QoS values for each one of the services from the process and computes the QoS values for the entire process;
- *BPRules Manager and Evaluator*: evaluates the current state of the process and determines corrective actions;
- *Process Management*: performs actions on the process.

Two repositories assist the management process: The *BPR repository* stores the rulesets formulated in our BPRules language [4], and the *service registry* contains descriptions of services to be integrated into the process.

Our system also shows a *self-configuration* capability during set-up time: At deployment time, the system parses the BPEL process description file and creates and attaches a sensor for each of the relevant BPEL activities (as for *sequence*, *invoke*, *assign*, and *for*). Each time an activity state changes (represented by *activated*, *faulted*, or *completed*), an event is published via the Sensor API and received by the QoS Monitor. This starts the loop as shown in Figure 2.

The *QoS Monitor and Aggregator component* is responsible for monitoring and/or aggregating the QoS dimensions of the business process and mainly plays the role of the *Monitor* component in the MAPE cycle. In order to obtain reasonable QoS values for the process, we have to consider the QoS values of all Web Services of the business process. Moreover, QoS values can be delivered by other monitoring services as well, e.g. from the service provider. The main task of the QoS Monitor/Aggregator is to compute the QoS values for the activities of the process from all of these inputs. Some examples of measured QoS dimensions are response time, cost, throughput, or availability. More details about the monitoring can be found in [5].

The *Analysis* part is realized by the *BPRules Manager and Evaluator Component* which is the core of the system. It runs a repository to store and retrieve BPR documents and evaluates them according to the monitored business process. For the evaluation, the BPR documents are automatically transformed into Drools Files which are then executed on the *Drools Rules Engine*. The BPRules Manager/Evaluator gets the QoS values of the business process from the QoS Monitor/Aggregator and matches them against the rule conditions. On a match, the corresponding action is triggered in order to improve the process behaviour. This may be understood as the *Planning* part of the cycle.

The actions from the BPRules manager are delegated to the *Process Management Component*, i.e. the *Execute* part. It utilizes the *Oracle Client API* as effectors to apply changes on the process like starting, stopping, deploying, or un-deploying. It is also capable of replacing services by consulting the service registry and dynamically changing the parts of the process. Hence, the process is managed to be conformant to the QoS expectations.

Replacing services at runtime is not as simple as it seems at first sight. After candidates have been found in the service registry, these services need to be bound dynamically into the process, but without affecting the course of business. This is handled in our system by the use of proxies; more specifically, we employ Java Servlets as intermediary components which receive the messages from the process and further delegate them to the concrete service. Accordingly, the WSDL service references which occur throughout the BPEL description file are replaced with the proxy references. When a service must be replaced, the proxy endpoint reference is

changed, pointing to the reference of the concrete service which has been discovered from the service registry.

4 Related work

Rule-based approaches targeting QoS in Web Service Compositions have been proposed by [11] and [15]. Similar to our work, the web service compositions were also implemented with BPEL. Baligand et al. describe in [11] their *QoSL4BP*, a policy-based language. Repp et al. propose their *WS-Re2Policy* language [15], based on WS-Policy. Notable differences between the two languages and BPRules are found in the provided features, whereas the semantics and syntax of the languages are different.

BPRules supports some features which distinguish it from the other approaches. We allow for relations between QoS parameters from different sections, rules defined on a number of instances (like, for instance, a constraint applying to 40% of the running instances), rules applied to instances executed in a certain period of time, adaptive service selection (permitting to select an appropriate selection method) or grouping of rules into rulesets. Moreover, rulesets may be changed and updated at runtime.

Features like the selection of services are generally required in business process management, and are thus shared between the approaches. Beyond that, our selection approach allows for different QoS requirements in different sections and the possibility to specify the selection algorithms for the sections and the process. We can also define where the QoS values are retrieved from, whether they are measured or defined in the SLA.

Zeng et al. describe a "QoS-Aware Middleware for Web Service Composition" [3]. In their work, the service compositions are represented as statecharts which consist of states and transitions. In our approach we use the BPEL constructs and the BPEL description file which we map to a tree. We have addressed similar QoS parameters as Zeng et al., like cost, responsetime, and availability. The authors present two approaches for the service selection, one via local optimization and another one via global planning using integer programming. While in their approach the objective function needs to be linear, in our approach we also consider non-linear functions.

M. Jäger also addresses QoS in service compositions [7],[6]. He concentrates on the aggregation of QoS and on different approaches for service selection, such as integer programming and heuristic algorithms. Compared to [6],[3] we proposed a rule-based approach, for easier detection of process malfunctions and behavior improvement. We can also integrate the selection algorithms described in [6],[3] as selection methods (like in the above example) within our BPRules language. By the specification of management rules the QoS can be better tailored to individual management requirements.

Similar to [14] we are adressing non-functional requirements for BPEL processes. In [14] the authors consider non-functional requirements, which are different from ours, like reliable messaging, security and transaction. Their framework enforces the non-functional requirements by the invocation of middleware web services. In our system the QoS parameters are monitored during the runtime and only if the behavior of the process is not adequate, corrective actions are triggered. Thus we intervene in the process execution only if required. Another similarity of both approaches is the specification of the non-functional requirements in extra files, thus

keeping them separate from the process functionality. Charfi et al. make use of the deployment descriptor while we are using the BPR documents, both specified in XML. In addition, we define rules with corrective actions for behavior improvement within our BPR-documents.

5 Conclusion

The management of business processes with respect to QoS remains a major challenge for the execution of business processes over the Internet. In this paper we presented our management system for business process self-management. Our concept has been implemented within the ADDOaction project [1] and is still work in progress. As we described above, we utilized the Oracle BPEL Process Manager and associated APIs to connect our system. Clients have different expectations in regard to the QoS of the process. By making use of our BPRules language we are able to formulate rules which apply to a whole or to sections of business processes, providing novel management capabilities which we consider as crucial in the process management. We proposed a service selection strategy that can be customized to the client's requirements and we presented how the service selection and replacement is performed within our system. As future work, other service selection algorithms will be analysed to be employed with BPRules.

Bibliography

[1] Bleul, S., Comes, D., Geihs, K.: Automatic Service Brokering in Service oriented Architectures, Project Homepage. http://www.vs.uni-kassel.de/research/addo/.

[2] Canfora, G., Penta, M., Esposito, R., Villani, M.L.: An approach for QoS-aware service composition based on genetic algorithms. In: Proceedings of the 2005 conference on Genetic and evolutionary computation, pp. 1069–1075. ACM, Washington DC (2005)

[3] Zeng, L., Benatallah, B., Ngu, A.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-Aware Middleware for Web Services Composition. In: IEEE Transactions on Software Engineering, pp. 311–327. IEEE Press, (2004)

[4] Comes, D., Bleul, S., Zapf, M.: Management of Business Processes with the BPRules Language in Service Oriented Computing. In: 16th Workshops der Wissenschaftlichen Konferenz Kommunikation in Verteilten Systemen 2009, WowKiVS, Electronic Communications of the EASST, Kassel (2009)

[5] Comes, D., Bleul, S., Weise, T., Geihs, K.: A Flexible Approach for Business Processes Monitoring, In: Proceedings Distributed Applications and Interoperable Systems, DAIS 2009, p.116–128. Springer, Lisbon (2009)

[6] Jäger, M.: Optimising Quality of Service for the Composition of Electronic Services, PhD thesis, University of Berlin, Berlin (2007).

[7] Jäger, M., Rojec-Goldmann, G., Mühl, G.: QoS Aggregation for Web Service Composition using Workflow Patterns. In: 8th International Enterprise Distributed Object Computing Conference (EDOC 2004), IEEE Computer Society, California (2004)

[8] Web Services Business Process Execution Language Version 2.0, OASIS standard, 2007, http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[9] Oracle BPEL Process Manager, 2008,http://www.oracle.com/technology/products/ias/bpel/index.html

[10] Miller, B., The autonomic computing edge: The role of knowledge in autonomic systems, 2009, http://www.ibm.com/developerworks/autonomic/library/ac-edge6/#N10100

[11] Baligand, F., Rivierre, N., Ledoux, T.: A Declarative Approach for QoS-Aware Web Service Compositions. Proceedings of the 5th international conference on Service-Oriented Computing ICSOC '07, Springer (2007)

[12] Berbner, R., Spahn, M., Repp, N., Heckmann, O., Steinmetz R., Heuristics for QoS-aware Web Service Composition, In: Proceedings of the IEEE International Conference on Web Services, IEEE Computer Society (2006)

[13] Business Process Model and Notation (BPMN), OMG Document, 2009, http://www.omg.org/spec/BPMN/2.0/

[14] Charfi, A., Schmeling, B., Heizenreder, A., Mezini, M., Reliable, Secure, and Transacted Web Service Compositions with AO4BPEL, In: Fourth IEEE European Conference on Web Services (ECOWS'06), IEEE computer society (2006)

[15] Repp, N., Eckert, J., Schulte, S., Berbner, R., Steinmetz, R.: Towards Automated Monitoring and Alignment of Service-based Workflows. In: IEEE International Conference on Digital Ecosystems and Technologies (2008)