BPRules and the BPR-Framework: Comprehensive Support for Managing QoS in Web Service Compositions

Diana Comes, Harun Baraki, Roland Reichle, Kurt Geihs

BPRules and the BPR-Framework: Comprehensive Support for Managing QoS in Web Service Compositions

Diana Comes, Harun Baraki, Roland Reichle, and Kurt Geihs

University of Kassel
Distributed Systems Group
Wilhelmshoeher Allee 73, 34121 Kassel, Germany
{comes,reichle,geihs}@vs.uni-kassel.de
baraki@student.uni-kassel.de
http://www.vs.uni-kassel.de

Abstract. For a successful collaboration between enterprises, Web services and service compositions need to fulfill certain QoS (Quality of Service) requirements so that they can be trusted by their clients. Thus, the best services have to be chosen for the composition, the performance of the composition needs to be monitored and in case of QoS deviations, appropriate management actions are required. We propose the BPRules language and the BPR-framework that offer novel capabilities and improved flexibility for the management of BPEL processes with regard to QoS concerns. The BPRules language allows to specify the QoS monitoring of BPEL processes and offers a variety of management actions for controlling the process and for the improvement of its QoS behavior. Thereby, the BPR-framework provides the necessary components to perform the QoS monitoring and to execute the management actions. For the selection of high quality services, the BPR-framework comes with efficient selection algorithms, like our $OPTIM\_PRO$ algorithm. We present the features of BPRules that we consider as indispensable for managing the services' QoS behavior.

1 Introduction

Web Services gained much interest through their reliance on XML based standards (e.g. SOAP, WSDL) and their accessibility over the Internet. Web services from different partners may cooperate to form a business process that delivers a higher business value to its clients. In a service oriented architecture (SOA), business processes are commonly realized as Web service compositions. The Web Services Business Process Execution Language, shortly WS-BPEL [7] (or BPEL) has emerged as the standard technology for executing such a process. Services need to offer and maintain adequate quality, also known as Quality of Service (QoS) (e.g. response time, availability, reliability) to accomplish clients' expectations. Usually, the QoS values that were negotiated between the client and the

service provider are kept in a service level agreement (SLA). We assume that in future service markets, there are multiple services offered from different service providers having the same functionality but with different service levels. Thus, choosing the services with appropriate QoS levels is an important issue for the success of the business process. Unfortunately, undesired situations, like a service becoming unavailable or not responding in the desired time frame, might happen in a dynamic environment like SOA. The malfunction of one single service might cause the failure of the entire process and lead to clients' dissatisfaction. In order to avoid such situations, the business process needs to be continuously monitored and immediate management actions have to be undertaken to improve the business process behavior. In this paper, we address exactly these challenges and propose the BPRules language (***B**usiness **P**rocess **Rules** Language*) and the BPR-framework which offer novel management capabilities and flexible monitoring of QoS for BPEL processes. The rules specified with BPRules are executed within our BPR-framework, which is responsible with the monitoring of QoS and triggering appropriate corrective actions on the business process when undesired QoS values are measured. Specifying the rules properly may prevent possible SLA violations and ensures a better control of the BPEL process. We especially designed BPRules with features that we envision as mandatory for the QoS management of business processes. Among the features of BPRules are: *flexible QoS data retrieval*, *section control*, *instance-set handling*, a *corrective actions set* including *flexible service selection* methods and a *dynamic rule set change*. The BPR-framework and BPRules go beyond the state of the art by offering novel features like *instance set handling* and a *dynamic rule set change*, but also by improving the flexibility of already established features as e.g. for *QoS data retrieval*, *section control* and an advanced *decision and control support* for the business analyst. With regard to these features BPRules provides more sophisticated means to specify management rules compared to already existing approaches. Improved flexibility is also achieved with regard to the service selection strategies, where for each section of the process a different service selection algorithm can be employed depending on the section size (number of involved services) and the expected number of service candidates. Since QoS dimensions vary over time, we designed the BPR-framework to be flexible when new QoS dimensions or new corrective actions need to be considered.

The remainder of the paper is structured as follows. In section 2 we present some foundations about BPEL and QoS. Section 3 presents the BPRules language and its features. The architecture of the BPR-framework is described in section 4. Finally, we compare our approach with related works in section 5 and conclude the paper in section 6.

2 Service composition and QoS

2.1 Overview

WS-BPEL [7] is a language for the specification of service compositions and their execution. A BPEL process may consist of several activities, like *invoke* for

calling a Web service, activities for defining loops (e.g. *while, repeatUntil*) and conditional activities (e.g. *if*). The activities can either be called sequentially (nested inside a *sequence* activity) or in parallel (nested inside a *flow* activity). A BPEL instance may be started with the arrival of a request message from the client by the *receive* or the *onMessage* activity. After processing the request, the response may be sent back to the client by the *reply* activity. As an example we consider the *bookshop process* for buying books in an online shop, which we implemented using BPEL. The process starts with the arrival of a request from the client, which contains the list of books the client wants to purchase. Then, the *stock service* is invoked and it verifies, whether all of the books from the list are in stock. If this is not the case, 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 prices are selected to be bought. The books are assigned to the client's bill and the *billing service* is invoked for creating the bill. Finally a *bank service* is invoked to make the withdrawal from the clients' credit card to the shop account.

We considered QoS dimensions like availability, reliability, response time and cost, but the framework is not limited to these dimensions as it can be easily extended to other QoS properties as well. In our framework, we interpret the QoS of a Web service as described by Zeng et al. [2]. *Response time* is computed as the duration of the time when the service operation was requested and the time when the result is received by the requestor. It is the sum of the transmission time and the processing time. The *cost* of the service is the price of invoking an operation of the service. *Availability* is the probability that the service is accessible. *Reliability* is the probability that the request is responded correctly in the expected time frame.

2.2 Service Composition representation

In the BPR-framework service compositions are represented as trees. This is achieved by transforming the BPEL description file into a BPEL tree. The BPEL activities are represented as tree nodes and the tree structure is built from the BPEL XML file structure. We consider only the activities, which are relevant for the QoS monitoring (e.g. *partnerlinks* are not included). The BPEL engine creates a new process *instance* for every request that it receives from the client. In the BPEL process, an *abstract service* represents the functionality of the desired service. We assume that there are multiple *concrete services* that provide this functionality but at different QoS levels. Selecting the right concrete services for the composition is known as the service selection problem and is known to be NP-hard [9]. We give a short insight of how we handled this problem in section 3.1 and further details can be found in [4]. In Figure 1 we consider a simple BPEL tree for illustration purposes. We have four abstract services: S_A, S_B, S_C, and S_D. We denote with $S.V$ the service variants being the set of concrete services that can realize the service S. The QoS requirements for the service selection are specified using our BPRules language.

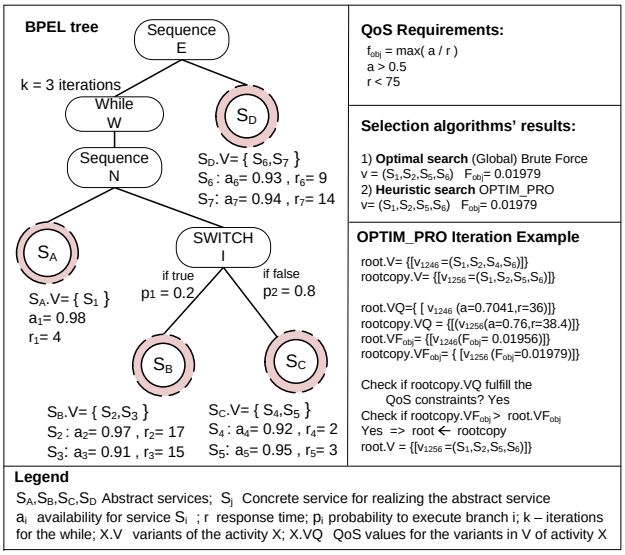

Fig. 1. BPEL Tree example

3 The BPRules Language

Guaranteeing the fulfillment of the QoS requirements needs runtime monitoring and management if there is a risk of QoS violations. BPRules is a rule-based language that offers management capabilities with regard to the QoS behavior of single Web services and Web service compositions. The business analyst may specify rules for the service process by stating what corrective actions should be undertaken if specific QoS requirements are not met. Appropriately chosen rules enable a proper execution of the business process even when unpredictable problems occur (e.g. a service is not accessible). The rules are specified in BPR-documents in the BPRules language and are processed by the BPR-framework. Corrective actions might rank from just notifying the interested parties about certain events, over starting or stopping the process to actions like selecting and replacing some services with others that provide better QoS. The rules are specified in XML and the syntax is validated against the BPRules XSD schema. We developed BPRules with the following design rationales in mind: *simplicity, expressivity, reusability* and *separation of concerns*. BPRules is *simple* to use because rules are specified in XML. The business analyst who specifies the rules is not required to have any programming skills. BPRules is *expressive* because it provides various features for QoS management, as will be shown in the next section. *Reusability* is supported by the possibility of reusing elements specified inside the BPR-document. Elements are identified by *ids* and can be reused throughout the BPR-document by simply referencing the *id*. Also other BPR-documents can be included or external BPR-documents can be referred to

by their *URI*. We achieve *separation of concerns* by specifying rules in BPR-documents which are stored separately from the business logic.

A regular BPR rule consists of a QoS *condition* which is monitored and the corresponding *action* which is triggered when undesired QoS values are measured. The *action* part in turn might enclose several BPR corrective actions available in the actions set of BPRules. A BPR-document contains several elements like *sections* for the specification of sub-orchestrations and *rule sets* for grouping rules together. We designed BPRules with several features that we envision as mandatory for the QoS monitoring and management. In the following we give an overview of these features.

3.1 BPRules features

Flexible QoS data retrieval: Interpreting and processing the QoS data may be dependent on the period of time when the execution of the process took place. For example, past QoS behavior may be retrieved for a report or analysis while current QoS behavior malfunction may be remediated by updating the process at runtime. With BPRules we can specify rule sets that consider process instances from a specific period of time. For instance, the period may be a time interval in the past or might range from a moment in the past till the actual moment. It can be specified as a concrete time interval (with a start and end date/time) or as a relative period in the form: *last x time-unit* (e.g.: last 10 hours).

Section Control: For a better control and detection of QoS deviations we can divide the process into several parts, which we call *sections*. We may define a section by referring to a structured activity with its nested sub-activities (e.g. all activities inside a *flow*). Another way for specifying a section is to consider all activities between a *start* and an *end* activity inside a *sequence*. As an example, in our bookshop process we define a section which consists of several activities, involving the invocation of the distributor service for checking if the book is available, then choosing the book with the minimal price and buying it.

```
1  <ruleset id="distributor">
2   <period> <!-- time interval --> </period>
3   <rule>
4    <condition> <constraints>
5      <!-- min 10 % of the instances -->
6      <expression applysection="distributorsection">
7        <or> <!-- responsetime > 3 or cost > 0.25 -->
8        </or> </expression> </constraints>
9    </condition>
10   <action> <!--from the BPR-corrective actions set-->
11     <replace-ws> <service name="bookshop/DistributorService">
12       <wsdl-url> <!--url to the WSDL of the new service-->
13       </wsdl-url> </service>
14     </replace-ws>
15   </action>
16  </rule>
17 </ruleset>
```

Listing 1.1. A BPR-rule example for a section

The Listing 1.1 contains a rule example defined for the *distributor section*, where a low *response time* and *cost* is required. The example also shows the

general structure of a rule set and a BPR-rule. Please note here that at some places in the example listings, commentaries are used instead of the lengthy XML syntax for brevity reasons. When the QoS of the section reaches some risky values (*response time* $>$ 3 s or *cost* $>$ 0.25) then the distributor service will be replaced with another one that provides better QoS and whose WSDL description is available at the specified URL. The *expression* element contains the QoS constraints which can be linked by the logical operators *AND*, *OR* and *NOT* to form more complex conditions. We can specify different QoS requirements in different sections. With BPRules it is also possible to establish relations between the QoS of different sections and the entire process. For example a query like this is possible: the *response time* from the *distributor section* is less than 1/2 of the *response time* of the *bookshop process*. Thus, the business analyst may be informed if the distributor section consumes too much time in comparison to the response time of the entire process, which can be a good indication for a malfunction in the distributor section. Furthermore, this kind of QoS conditions may ensure keeping an appropriate proportion between the QoS parameters between process sections and the process.

Instance-set handling: With BPRules, we can specify a certain set of instances to which the QoS constraints apply. This is an important task since, for example, situations when 2% of the instances failed or over 20% of the instances failed need to be treated differently. While the first case could be tolerable, the second case needs to be addressed adequately. In Listing 1.2 it is stated that if *minimum* 20% of the instances failed then a *select services action* should be undertaken to replace the services with others that provide better QoS.

```
1  <rule id="selectAll">
2   <condition>
3     <constraints>
4       <instances-subset function="MIN">20%</instances-subset>
5       <expression>
6         <property-check select="state">FAULTED</property-check>
7       </expression>
8     </constraints>
9   </condition>
10  <action>
11    <select-services methodClass="ALG.OptimPRO">
12      <service-registries> <ws-url><!-- url to registry --></ws-url>
13      </service-registries>
14      <qos-requirements>
15        <!-- resp < 3; avail > 0.95 ; cost < 0.3 -->
16        <!-- obj f = max (5 * avail) / (resp + cost) -->
17      </qos-requirements>
18    </select-services>
19  </action>
20 </rule>
```

Listing 1.2. A service selection example

As described in Listing 1.2, the state of the instances can be queried with the *property-check* element (line 6). We distinguish between states like FAULTED for instances with activities that have thrown an exception, RUNNING for instances with activities that are still executed, and COMPLETED for instances where all of their activities are completed. For querying the size of the instances set that fulfill or violate the QoS constraints, BPRules offers a set of functions:

FORALL targeting all the instances in the set, EXISTS for at least one instance, MIN nr(%), MAX nr(%), EQUALS nr(%) to refer to a percentage of the total number of instances. With these functions, BPRules makes it possible to trigger appropriate actions according to the runtime behavior of the instances.

Flexible Service selection: BPRules provides extra flexibility for the *selection of services*. The *select-services action* from BPRules may be employed for the entire process, for an abstract service or only for some of its sections. It triggers a selection algorithm to search for services in specified service registries and to replace the old services in the process with new ones that provide better QoS. The selection algorithms receive as input the QoS requirements of the process, which consists of the QoS constraints and an objective function to be optimized. In contrast to other works [2], our selection strategies are also able to deal with non-linear objective functions, aggregation functions and constraints.

Our selection action is *customizable with regard to the selection method (algorithm)*. For example, when searching a few services, like within a section, a trivial brute-force search is sufficient, while in a search that involves many services (e.g. for the entire process) during runtime, a more advanced and rapid search is needed. For this purpose, the BPR-framework provides three algorithms, $OPTIM\_S$, $OPTIM\_PRO$ and $OPTIM\_HWeight$, that can be employed for the selection of services. For brevity reasons we only present a rough sketch of $OPTIM\_PRO$ in this paper. Further details about the selection algorithms can be found in [4]. $OPTIM\_PRO$ is an iterative algorithm and improves the variant found on the root node (the objective value of the variant) with each iteration step. Figure 1 represents a simple iteration example. The QoS of the root node variant is computed by performing a QoS aggregation from the bottom of the tree to the top. The objective value of the root node is computed by applying the objective function to the found variant on the root node. All nodes of the tree are assigned with a priority factor, which means that nodes that are executed more often receive a higher priority. In the following steps, new services are selected for the nodes in the order of their priorities. For each service candidate we make a copy of the root node, where the currently selected service candidate replaces the old service candidate, aggregate the QoS of the root copy variant, check it against the QoS constraints and compute the objective value. If this objective value represents an improvement in comparison to the old root variant, then the root is going to receive the value and the services of its root copy variant, otherwise the root variant remains the same. The variants that no longer can be improved are saved into the list *vlist*. The same process starts again with a randomly selected service variant. Finally the variants from the *vlist* are sorted by the objective function and the best found variant is returned.

Listing 1.2 (lines 11-18) represents an example of a *select-services action* defined with BPRules. New services are searched in the service registries. We assume that a service registry is exposed as a web service and accessible via a URL. The *methodClass* attribute (line 11) is used to specify which of the selection algorithms is employed. In the listing example, the $OPTIM\_PRO$ algorithm is called. There may be situations when certain services are preferred and it is not

desired to replace them during service selection. In this case, we may declare these services in BPRules as *fix*, which means that they will not be replaced during the selection procedure.

The BPR corrective actions set: BPRules offers several corrective actions which we divided into 4 categories: (1) actions for *controlling the BPEL process*, (2) actions that are meant to *improve the QoS behavior of the process*, (3) actions which offer *information about the QoS behavior* and (4) actions for *controlling the rule sets*. Table 1 gives an overview of the actions offered by BPRules. The actions from the first category offer support for controlling the process and its instances, like deploying and undeploying the process, or stopping a set of process instances. The actions from the categories (1) and (2) trigger changes in the state of the process and the process goes into the *managed state*. In the managed state, the actions from the categories (1) or (2) may be triggered only sequentially in order to avoid process inconsistencies. The actions from the category (2) are meant to improve the QoS behavior of the process, by replacing one or more services with other services that provide better QoS. If errors were detected inside the process, these usually have to be repaired by updating the BPEL file. This kind of correction is supported by our *update* action, which overwrites the process description file with another file from a given path or registry. The third category is meant to inform the interested parties about the behavior of the process. BPRules can provide information during process execution (e.g.: throw-event, notify-client) but also reports for longer periods of time. For the business analyst BPRules offers different kinds of reports: a *regular report*, a *rules-report*, and an *error-report*. All these reports deliver a good picture of the process behavior to the business analyst. In the *rules report* the business analyst can see how the rules were executed, which helps him with future rules specifications. The actions described in Table 1 are atomic actions. Usually, for managing the process properly, several actions need to be triggered. For this purpose, the atomic actions can be composed into so called complex actions. BPRules has some predefined complex actions. Also, the business analyst is able to specify its own complex actions which he may reuse. The different kinds of reports and the possibilities of composing actions, defining custom actions or applying manual actions (see the $< replace - ws >$, $< fix >$ declarations) deliver an advanced *control and decision support* for the business analyst. This feature was included as not always a fully automatic management is desired.

Dynamic rule set change: We may *activate* or *deactivate* rule sets at runtime. Active rule sets are those rule sets which are executed, while inactive rule sets are temporarily ignored. We may use the various rule sets for different alarm states analogously to a traffic light system. For example, if the process behaves well, then the active rules could only inform the interested parties about the behavior. In contrast, if the QoS of the process gets worse another rule set could be activated with rules that have more impact on the process, e.g: replacing one or several services. In this way we may adapt the rule sets dynamically at runtime, according to the behavior of the process. This mechanism reduces complexity by removing the rules that are no longer needed from the memory.

1. Control the process	
Deploy/Undeploy	Deploys/ Undeploys the process from the specified *path* or registry identified by a URI.
Stop	Stops the process identified by the *processID*. All the process instances of this process are stopped. All the requests that are received while the process is stopped, are stored into a *request-queue*.
Start	Starts the process identified by the *processID* so that the process is able to receive requests. New process instances are started for the requests from the *request-queue* if the waiting time in the queue didn't exceed the given threshold (*timeout*).
Stop-instances	Stops a set of process instances. (E.g.: instances that started within a given time interval)
Resume-inst.	Resumes a set of process instances, that were previously stopped.
Cancel -instances	Cancels a set of process instances.
2. Improve the QoS process behavior	
Update	Updates the BPEL process description (or section) from the specified *path* or registry identified by a URI.
Replace-ws	Replaces the Web service that realizes a given *abstract service* with a new *concrete service* (or replaces an entire list of services). The *URL* of the WSDL of the new concrete service has to be specified.
Select-services	Selects services with better QoS from the repository and replaces the Web services in the specified section/process/abstract service.
3. Information about the process behavior	
Report	Makes a report about all the monitored artifacts: the measured QoS values, including exceptions and events of a process during a given time period.
Report-rules	Makes a report with the rules that were triggered for a process during a given time period. The report can be created for all rule sets or only for the specified rules.
Report-error	Makes a report with the errors that were encountered during process execution during a given time period.
Notify-client	Sends a message to the client announcing him about e.g. QoS constraints fulfillments/violations or details about the execution.
Throw-event	Generates an event and informs the subscribers.
Custom action	An interested party may implement a customized action for its own specific needs. Therefore the *path* to the *class* file that implements the action interface from the BPR-framework has to be provided.
4. Control the rule sets	
SetActive-ruleset	Activates or deactivates the rule set identified by an ID.
Reload-ruleset	Reloads a new rule set at runtime.

Table 1. Corrective Actions Set from BPRules

BPRules provides also a *reload-ruleset* action for updating the rules at runtime. This is necessary in a dynamic SOA where partners or contracts may change. The *reload-ruleset* action permits overwriting, adding new rules into the rule set or removing rules. We even may retrieve rules from a URI.

4 The BPR-Framework

4.1 Architecture and implementation

We have designed and implemented the BPR-framework for evaluating how our BPR-rules impact the QoS behavior of BPEL processes. The processes are executed on the *Oracle BPEL Process Manager* engine [8] and the Web services on the Apache Axis2 engine. We have implemented a service registry using a MySQL database where services can be searched or published. Besides the WSDL files of the services, we also store the QoS values promised by the service providers in the registry. The BPR-framework (see Fig. 2) is implemented in Java and it contains several modules: the *BPRules Manager* (shortly Manager), which is the core module, the *QoS Monitor & Aggregator* module for QoS monitoring, and the *Process Management* module for performing the corrective actions. The BPR-documents are stored into the BPR-repository. We distinguish between two execution phases: the *initial* phase, when all the necessary monitoring artifacts are deployed, and the *monitoring phase*, when the actual QoS monitoring and management takes place. In the *initial phase* the *Manager* loads the BPR-documents (see Fig. 2, *step Ini 1*) from the BPR-repository. The *Manager* reads from the BPR-documents which BPEL processes, sections and QoS Parameters are going to be monitored. The service selection algorithm is triggered by the *Manager* to select appropriate concrete services. The *Manager* creates a proxy for each of the abstract services, which contains a reference to the URL of the currently selected concrete service. It intercepts all the messages that are transmitted to the concrete service. The *Manager* may update the endpoint references from the BPEL file with the URL of the proxy. When a service replacement is triggered, the proxy is updated, referencing to another concrete service URL. Currently, the BPR-framework supports synchronous, stateless web services.

For the monitoring we use a feature of the Oracle BPEL engine which offers the possibility to attach sensors to the BPEL activities. Such a sensor may inform when a BPEL activity is started/ended or when a failure occurred. The *Manager* dynamically attaches sensors to all the activities of the BPEL process. By this, all the monitoring artifacts were created and the BPEL process can be deployed (step *Ini 2*). In the next step (*Ini 3*) the BPR-rules need to be deployed on the rules engine. We employed the *Drools* rules engine from JBoss for executing the rules. Before deployment, the rules from the BPR-documents are dynamically transformed into Drools files (having the Drools syntax), which can be processed by the Drools engine. Since the BPRules and the Drools rules contain common rules constructions (e.g. condition/action, logic operators) the transformations between the two syntaxes can be done dynamically. We also used the possibility offered by Drools to implement customized functions for percentage, MIN, MAX that are applied to the QoS objects. Finally, the Drools files are deployed to the Drools engine and the initial phase is terminated. During process execution, the sensor messages (from each activity) are delivered to the Manager (step 1). The sensor message contains the instance ID of the process, the sensor ID, the timestamp, the evaluation time (activation or completion of the activity) and

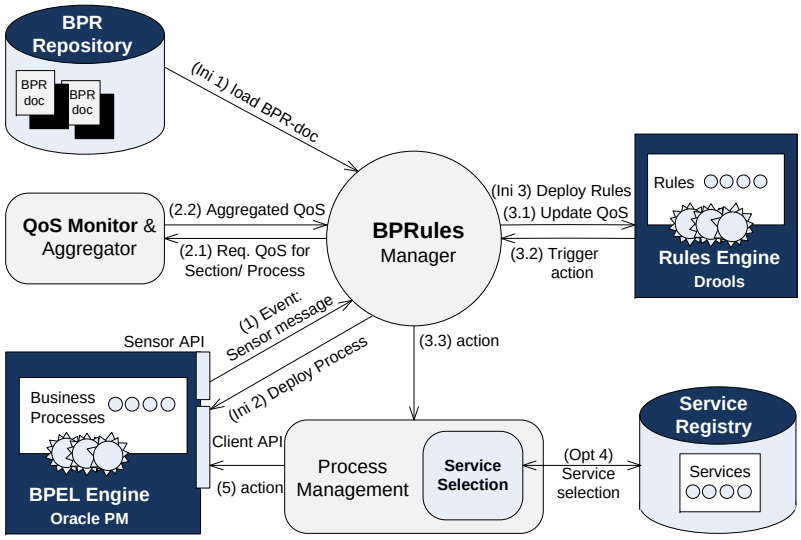

Fig. 2. The BPR Framework

whether an error occurred. If the sensor represents the end of a section or of the process, the *Manager* calls the *QoS Monitor and Aggregator* to perform the QoS computation of the section or the process instance (steps 2.1, 2.2). The QoS of the section or process are computed out of the QoS of the atomic services within the section or process. Further details about our aggregation algorithm can be found in [3]. With these new QoS values, the *Manager* updates the QoS objects from the Drools memory (step 3.1). The Drools engine permanently evaluates the QoS conditions and in case they are met it delegates the corrective actions to the *Process Management (PM)* module. Finally, the *PM* module is able to execute the actions on the process. The Oracle BPEL engine offers a Client API for querying and controlling the BPEL instances (e.g. stopping instances, deploying the process). Our PM module makes use of this Oracle API and additionally adds other necessary actions (e.g. select-services, replace-ws, etc.).

4.2 Evaluation

For evaluation purposes we used a Lenovo R60 notebook with Intel Core 2 Duo processor T5600 (2x1,83GHz) with 2 GB memory and Windows XP SP3. As example we used the bookshop process and several other processes and tested the BPRules features. We defined several rules and simulated QoS constraints violations, like services being not available, not responding in the desired time frame or services causing errors and being not reliable. Our experiments revealed that the conditions of several rules might be met simultaneously, which results in the situation that a number of management actions on the process are performed at the same time. To overcome this undesired situation we enhanced the BPR-

framework to block a process in the *managed* state until the actions that are impacting the process are finished. We also added an adjustable mandatory time interval between triggering two consecutive actions that impact the process. We observed that grouping rules into rule sets and activating or deactivating them, makes it much easier for the developers to trace rules. In this respect, the dynamic rule set change offered by BPRules provides an important mechanism to relax the problem of dealing with simultaneously applicable and potentially contradictory rules. So far there are no mechanisms for avoiding contradictory rules. This issue is still left to the business analyst to be solved. In our future work, however, we aim to provide more development support to the business analyst. Another possibility to deal with contradictory rules is to automatically resolve the conflicts. The authors of [11] propose in their architecture a Policy Conflict Resolution module based on business metrics. We plan to analyze if this approach can be adopted for our BPR-Framework. For our bookshop process that has 40 activities we measured the average QoS aggregation time for a process instance as being 0.48 ms. We observed that the aggregation time grows linearly with the number of instances. The time value represents the pure computation time for the QoS aggregation and does not include the time for the database access of QoS data retrieval. For the evaluation of the service selection algorithms we generated multiple BPEL trees with different structures and we varied the number of abstract and concrete services. We performed several experiments for comparing our $OPTIM\_PRO$ with the genetic algorithm from [1] with regard to computation time and optimality of the solution. Our experiments have shown that $OPTIM\_PRO$ was faster than the genetic algorithm, in average it needed about 22% of the time of the genetic algorithm. Concerning optimality our algorithm achieved up to 7% better values for the objective function in comparison to the genetic algorithm.

5 Related Work

By addressing QoS requirements for services, our BPRules language has similar goals as the two languages *Quality of Service Language for Business Processes (QoSL4BP)* [9, 10] and the *Web Service Requirements and Reactions Policy (WS-Re2Policy)* [5] language. All three languages have a similar structure by means of specifying actions to be undertaken upon QoS violations. Even though, BPRules, QoSL4BP and WS-Re2Policy differ in the provided features and syntax. BPRules offers various additional features like: *instance-set handling* with the possibility to query the state of the instances and the instances' set, *dynamic rule set change* and the specification of *rule sets applied on instances from different time periods*, which are not supported by the other languages. Also BPRules provides increased flexibility for the *QoS data retrieval* for past and/or running process executions, and an advanced *control and decision support* for the business analyst. Similar to the *section control* feature from BPRules, the authors from [9] are able to query structured activities for QoS. However, they cannot relate QoS parameters from different sections like in BPRules (e.g: the

response time from the *distributor section* is less than 1/2 of the *response time* of the bookshop process). A crucial action for managing QoS is the service selection action. The *service selection* is supported by all of the three languages but the used selection algorithms are different. In [10] it is mentioned that a constraint programming and a backtracking algorithm are used. BPRules may employ our $OPTIM\_S$, $OPTIM\_PRO$ or $OPTIM\_HWeight$ algorithms which can be triggered depending on the number of the service candidates.

Canfora et. al [1] describe a genetic approach for the service selection. We implemented their algorithm because it can be applied also to non-linear objective and aggregation functions. We used the same aggregation functions as [1] and compared our $OPTIM\_PRO$ algorithm to the genetic algorithm of Canfora. Our evaluations revealed that our algorithm needs less computation time and provide results which are at least as good as the genetic algorithm.

Baresi et. al describe in [12] an approach for service monitoring. The authors define monitoring rules in their Web Service Constraint Language (WS-Col) for WS-BPEL processes. In comparison to our language, WS-Col is limited to monitoring and doesn't allow to specify any corrective actions. In their work [6], the authors describe an approach for preventing SLA violations by a dynamic substitution of fragments (equivalent to our sections) at runtime. We may perform a similar kind of substitution with our *update* action, but in our approach the business analyst has to specify in the rules the exact replacement (e.g. the path) for the section or process. Thus, the work presented in [6] can be considered as an improvement for our update action for a more dynamic substitution of a section, which we plan to adopt in our future work. However, our focus was not on the dynamic substitution of sections. The authors have addressed only this particular substitution aspect in [6]. With our framework we aim to provide a comprehensive support for managing QoS of service compositions, that includes monitoring but also a rich set of corrective actions as well as efficient service selection strategies. The authors of [11] also propose a language and a framework for adaptation of Web-Service compositions, which is able to select the appropriate adaptation strategies for different classes of instances. The strategy selection is not only considering QoS dimensions but also business metrics. With BPRules and the BPR-framework we intend to improve the long-term QoS behavior by selecting and replacing services. Thus, in comparison to [11] our focus is much more on service selection algorithms and on specifying rules that define when and how to replace services.

6 Conclusion

Monitoring and managing QoS are crucial tasks that are decisive for the success of the business process. Our BPR-framework addresses exactly these matters and by means of the BPRules language, novel features are provided to overcome possible QoS deviations. BPRules and the BPR-framework offer improved QoS monitoring features, like monitoring QoS over sections, querying the QoS behavior of running instances (instance-set handling, states querying) but also of

instances which are already terminated. For managing the services, we provide several corrective actions like starting and stopping instances, service replacement, flexible service selection or dynamic rule set changes. The service selection action from BPRules offers the possibility of choosing the right selection algorithm depending on the number of abstract services and the number of available service candidates. $OPTIM\_PRO$, for example, is a very efficient heuristic algorithm suitable for processes with many abstract services. By providing sophisticated support for QoS monitoring, a rich set of management actions and efficient service selection algorithms, BPRules and the BPR-framework constitute a comprehensive solution for the QoS management of Web service compositions.

References

1. 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. ACM, Washington DC (2005)
2. 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)
3. Comes, D., Bleul, S., Weise, T., Geihs, K.: A Flexible Approach for Business Processes Monitoring. In: Proceedings Distributed Applications and Interoperable Systems, p.116–128. Springer, Lisbon (2009)
4. Comes, D., Baraki, H., Reichle, R., Zapf,M., Geihs, K.: Heuristic Approaches for QoS-based Service Selection. In: 8th International Conference on Service Oriented Computing (ICSOC) 2010. Springer, San Francisco (2010)
5. Repp, N., Eckert, J., Schulte, St., Niemann, M., Berbner, R., Steinmetz, R.: Towards Automated Monitoring and Alignment of Service-based Workflows. In: IEEE Int. Conference on Digital Ecosystems and Technologies. IEEE Xplore, Australia (2008)
6. Leitner, P., Wetzstein, B., Karastoyanova, D., Hummer, W., Dustdar, S., Leymann, F.: Preventing SLA Violations in Service Compositions Using Aspect-Based Fragment Substitution. In: 8th Int. Conf. ICSOC. Springer, San Francisco (2010)
7. Web Services Business Process Execution Language Version 2.0, OASIS standard, *http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html* (2007)
8. Oracle BPEL Process Manager, Oracle, *http://www.oracle.com/technology/products/ias/bpel/index.html*, [25.01.12]
9. Baligand, F., Rivierre, N., Ledoux, T.: A Declarative Approach for QoS-Aware Web Service Compositions. In: Proceedings of the 5th international conference on Service-Oriented Computing ICSOC. Springer, Vienna (2007)
10. Baligand, F., Rivierre, N., Ledoux, T.: QoS Policies for Business Processes in Service Oriented Architectures.In: Proceedings of the 6th international conference on Service-Oriented Computing ICSOC. Springer, Sydney (2008)
11. Lu, Q., Tosic, V.: Support for Concurrent Adaptation of Multiple Web Service Compositions to Maximize Business Metrics. In: Proceedings of the 12th IFIP/IEEE International Symposium Integrated Network Management (IM), Ireland (2011)
12. Baresi, L., Guinea, S.: Towards Dynamic Monitoring of WS-BPEL Processes, In: Proceedings of the Third International Conference of Service-Oriented Computing ICSOC. Springer, Amsterdam (2005)