Electronic Communications of the EASST
Volume 17 (2009)

Workshops der Wissenschaftlichen Konferenz Kommunikation in Verteilten Systemen 2009 (WowKiVS 2009)

Management of Business Processes with the BPRules Language in Service Oriented Computing

Diana Comes, Steffen Bleul and Michael Zapf

12 pages

Guest Editors: M. Wagner, D. Hogrefe, K. Geihs, K. David
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Management of Business Processes with the BPRules Language in Service Oriented Computing

Diana Comes[1], Steffen Bleul[2] and Michael Zapf[3]

[1] comes@vs.uni-kassel.de
[2] bleul@vs.uni-kassel.de
[3] zapf@vs.uni-kassel.de
Distributed Systems
Kassel University, Germany

Abstract: Quality of Service (QoS) concerns are an important topic for the realization of business processes. While BPEL is considered the de facto standard for web service compositions, QoS requirements are not part of its specification.

We present the *BPRules* (Business Process Rules) language for the management of business processes with respect to QoS concerns. BPRules is a rule-based, declarative language which brings novel benefits in the management of business processes, like QoS dependability for sub-orchestrations and corrective actions tailored to the specific needs of the clients. We present the main constructs of the BPRules language and how they support the flexible adaptation of the business process during runtime. Decision making is done according to the behavior of several process executions. An illustrative scenario shows how BPRules is applied to a business process.

Keywords: Business Process Management, quality of service, web service composition, orchestration, sub-orchestration, BPEL

1 Introduction

The constant growth of business processes across organizational boundaries inevitably leads to the need for integrating services from business partners into business processes. By encapsulating applications into web services, platform-independent, distributed and heterogeneous applications may be integrated easily over the Internet. Web Services from different partners can be composed into more complex workflows through web service compositions that implement the business process.

Among those languages which support the definition of Web Service compositions, BPEL is known as the de facto standard. The BPEL process has to fulfill certain functional and non-functional criteria so that the expectations of its clients are satisfied. For business processes to execute properly in real world scenarios, Quality of Service (QoS) is a major issue which needs to be taken into consideration. However, BPEL does not contain any specification for dealing with the QoS of business processes. In this paper we address non-functional requirements like response time, capacity, throughput and availability.

The business process is made up of several building blocks, among which services from other

business partners are triggered to achieve the desired business task. Thus, the global QoS of the entire business process depends on the QoS of the building blocks, as well as the binding of services.

As it is often the case, the business process does not always behave as expected, since a lot of unpredictable problems may occur, such as a service not being available or not responding in the desired time interval, or a network failure. Therefore it is important to specify corrective actions so that the business process will be appropriately managed to behave normally even in unexpected situations. Still, within current research studies, QoS requirements and management actions for the business process can be specified only on a broader level. Flexible and adaptable service management is an important but open issue in service management.

Our proposed language addresses exactly these management shortcomings and provides a solution to specify QoS requirements and corrections in a more refined and flexible manner.

The paper is structured as follows. Section 2 describes our motivation for proposing the new *BPRules* (Business Process Rules) language for the management of business processes. Moreover, the requirements for the BPRules language are stated. Section 3 presents the related work, by comparing the BPRules Language with similar research approaches. Section 4 presents the main constructs of the BPRules Language and provides several examples for their use. Section 5 describes how the prototype of the management system is build.

2 Motivation

We propose the BPRules language that allows to specify flexible management capabilities with respect to QoS concerns over business processes. BPRules is an expressive language, providing all the features that we identified as mandatory for business process management. According to the assessment of the current quality of the business process, a specified set of management actions may be selected and applied on the process. This assessment may, for instance, comprise the number of services which have failed during the execution of the process. Depending on the gravity of process malfunctions, rules with moderate impact on the business process are exchanged by rules with stronger impact at runtime.

As the process behavior may change in time, getting better or worse, the corrective actions need to be adjusted appropriately. Also, modifications that might occur, e.g. in the contract terms, should be reflected in the rules. This requires *changing or updating rules dynamically* at runtime which is another novel aspect of process management.

Another important task is *relating QoS parameters* and QoS constraints *between sub- orchestrations*, as a QoS parameter value from one sub-orchestration could be dependent on the value that it takes in another sub-orchestration of the process.

In order to tailor the corrective actions to the specific requirements of the clients, we define a set of *instances-subset functions* to select subsets of process instances to which the QoS concerns should apply. For decision making we consider the behavior of the process in its sub-orchestrations. BPRules supports the specification of dependencies between QoS constraints and QoS parameters from several sub-orchestrations.

BPRules is a domain-specific language and intended to be used in conjunction with BPEL processes.

The main design rationales for the BPRules syntax are simplicity, expressivity, reusability, and separation of concerns. *Rules* state under which circumstances certain corrective actions must be triggered on the business process. They are defined in XML and the syntax of a BPRules document may be validated by the BPRules XSD schema.

BPRules allows to reference parts within the same document using *id* attributes and external documents using URI references, emphasizing the reusability aspects of the language. Separation of concerns is achieved in BPRules by defining the rules separately from the business logic specified in BPEL. BPRules does not change or amend BPEL process descriptions.

In BPRules, *process sections* are sub-orchestrations that are defined as single blocks (like a *while* statement together with its activity block) or as a sequence of activities, starting and ending at two specified activities. In the following text, we will use the terms section and sub-orchestration synonymously.

To demonstrate the requirements for a new management language we illustrate a business process for renovating a house. We define two sections for the process. In the first section, *section1* of the process, two services are bound, one for the retrieval of materials and another one for having them installed into the house. In *section2*, services are invoked for painting the house and buying furniture. The two sections are triggered sequentially. For the process we specify two rules, which we group in a rule set. The following two rules (shown in an informal way) illustrate some aspects of the expressiveness of the language that we envisage.

Rule set 1-2

rule 1: **if** *FORALL running* process instances the *responsetime* in *section1*
is greater then 1/3 of the *total responsetime* of the process
then *replace all* services from section1.

rule 2: **if** *minimum 50%* of the instances are *failed*
then *activate* the *red* ruleset

It should be possible to specify corrective actions according to the behavior of an arbitrary set of process instances, thus, this set of instances must be dynamically adjustable. Therefore we define a set of functions, e.g FORALL, EXISTS, MIN, that can be used to specify to how many instances the requirements apply to. For example, it is not the same, if 2 instances failed or over 50% percent of the instances failed. In the latter case, corrective actions with stronger impact on the process (e.g. the red rule set) should be applied. The set of instances may be also created by filtering the set of instances according to certain properties like state (e.g. CANCELLED or RUNNING, see rule 1). It must be possible to trigger changes in the process and instances states.

Another important matter is relating QoS dimensions between sub-orchestrations, as it is the case in rule 1. Rule 1 defines a proportion between the value of a QoS dimension (e.g. response-time) of one section and the value of the QoS dimension of the entire process. Note that this is not limited to proportions; any kind of mathematical functions defined between QoS dimensions of sub-orchestrations is conceivable.

Another type of dependabilities are between QoS constraints for sub-orchestrations, which should be linked through logic operators. Therefore an example would be ”the *throughput* in

section1 is less than 1000 and the *throughput* in *section2* is less than the *throughput* in *section1*".

As all these features are beyond the capabilities of currently available languages, we propose the new BPRules language.

3 Related work

The *Quality of Service Language for Business Processes* (QoSL4BP) [BRL07] is a policy-based language addressing QoS requirements for business processes. The language offers a series of constructs for taking actions like detecting the violation of a SLA or violation of the QoS of a scope, selecting and renegotiating a concrete service, or replanning. Their approach is similar to ours in that it considers QoS requirements for sub-orchestrations. In contrast, our BPRules language is based on rules and rule sets that can be changed at runtime. We address other important issues like relating QoS parameters of different sections and considering the dimension of a subset of process executions, which are not possible with QoSL4BP. In addition, our language is XML-based and we can parameterize expressions for different sub-orchestrations, offering reusability of the QoS constraints. The languages also differ in the provided corrective actions.

[BGP06] present an approach for monitoring WS-BPEL processes, with a focus on security constraints. They describe the *Web Service Constraint Language* which is based on policies and is compliant with the WS-Policy framework. Policies are attached in WS-CoL only on service invocation activities, while our approach allows for defining rules on any activity and sub-orchestration.

Within WS-CoL and QoSL4BP, extra invoke activities are inserted into the BPEL process for the monitoring purpose. We keep the BPEL description untouched, specifying the sensor placement in external XML files which are interpreted by the Oracle BPEL PM server.

The AO4BPEL framework [CSHM06] also addresses non-functional requirements in BPEL processes. The approach focuses on reliable messaging, security and transactionality requirements. By choosing this area, the authors are concerned with some other important aspects of the non-functional requirements in comparison to our approach. While we took advantage of the Oracle BPEL PM server support for attaching sensors, they developed their own process container where the processes are executed. They define the process requirements for the BPEL activities in deployment descriptor files.

The *Web Service Offerings Language* (WSOL) [TPP02] and *Web Service Level Agreements* (WSLA) [LKD+03] are similar to our language in the sense that they address the management of web services by specifying QoS requirements and actions to be performed. Their focus is on requirements for web service operations and port types. Therefore they do not consider specific needs in orchestrations, like requirements attached to specific activities of the BPEL structure. In case of violations, management actions like monetary penalties (in WSOL) or notifications (in WSLA) can be triggered. Both languages specify the responsible management parties. Within our approach, the monitoring and management of the web service composition is the responsibility of the service provider, as they take place in the execution environment of the business process. We have adopted a similar syntax to WSLA for the specifications of expressions and functions.

4 BPRules - Business Process Rules Language

The BPR rules are at the core of the BPRules language. A BPR rule defines the measures that have to be performed if the specified QoS constraints are met by the business process. The QoS requirements specified within rules are evaluated by the rules engine which is also responsible for triggering the corrective actions. An important concern in the management of processes represents our state model that the process traverses during its lifecycle. Figure 1 represents the possible transitions of the process states. We derive the classification of our management actions from this state model. The most prominent feature is that each state can be changed by an action of our language and that we include a locked management state for an arbitrary amount of management entities. The process starts in the *Start* state where the BPEL process description is available inside the system. Before the process can be invoked we have to deploy the process on an execution engine, represented by the *Stopped* state. In the *Running* state, instances of the process appear for every client invocation. Afterwards there are two ways to stop the process. From the *Paused* state, instances may either be abruptly ended (destroy instances) or, the second option is, that the process can't be invoked, but existing instances remain running. In both cases, the process reaches again the *Stopped* state.

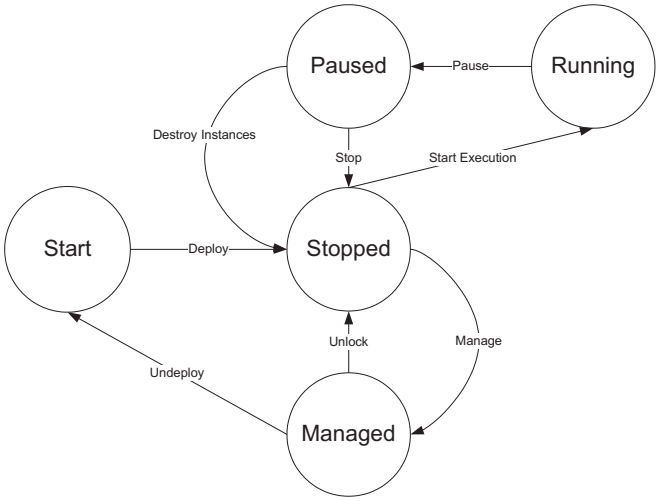

Figure 1: Business Process States

Management actions inside the BPRules language can be classified as follows: First, we offer management actions to change the state of a process with our rule actions, e.g. start, stop, and undeploy. Second, management actions like updates, changes, and the replanning of the process description itself are only available in the *Managed* state. In this state, the process is already stopped and locked which means that only one manager is allowed to submit this class of management actions. The third class of management actions triggers changes in the state of process instances, e.g. start, stop, cancel.

The fourth class of management actions can be executed independently of the process state, e.g. email notifications and logging. The first class represents a static class where new actions

cannot be introduced whereas the other two classes are flexible and can be customized. This also influences the system design of the runtime interpreter in order to easily adapt the language with new elements, e.g. a semantic service registry.

Table 1 provides an overview of the main corrective actions. Corrective actions on the business process may be distinguished between *management actions* and *general actions*. The *management actions* trigger changes in the state of the business process or inform the interested parties about changes occurred in the lifecycle of the process. The *general actions* target the rule sets. For the sake of brevity, we will not describe all possible actions.

Table 1: Corrective Actions

1. Management actions	
Class 1 - trigger changes in the state of the process	
Deploy/Undeploy	Deploys/ Undeploys the process.
Start/Stop	Starts / stops the process and changes its state.
Class 2 - the process is in the managed state	
Update	Updates the BPEL process description by a URI-referenced description.
Replace-ws	Replaces *web service1* with an alternative service, *web service2*.
Replan	Replaces all the web services in a specified section with alternative web services.
Class 3 - trigger changes in the state of an instance	
Start/Stop-Instance	Starts/ stops a process instance.
Cancel -Instance	Cancels a process instance.
Class 4- can be executed independently of the process state	
Notify-client	Notifies the client.
Throw-event	Generates an event.
Throw-exception	Generates an exception.
2. General Actions	
2.1 Rule Set	
SetActive-ruleset	Activates or Deactivates the rule set identified by an ID.
Reload-ruleset	Reloads a new rule set at runtime.

Here is an example of a *BPR document* containing several rule sets.

```
<bprules id="BPR–document1" processid="DrivingLicence1">
        <ruleset id="green" importurl="url1" active="true"/>
        ...
        <ruleset id="red" active="false">
                <rule>...</rule>
        </ruleset>
</bprules>
```

Figure 2 represents the main elements within a rule set.

Multiple rules can be grouped into a rule set. A rule set can be active, being evaluated at

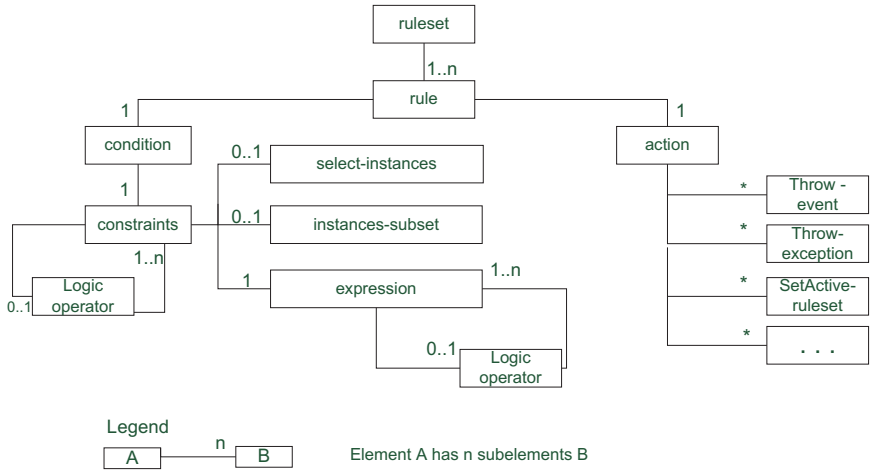

Figure 2: Rule elements

runtime, or inactive, being ignored temporarily. The various rule sets may be used for different alarm states, analogously to a *traffic light system*. Also, our language allows for importing rules from other documents, using the *importurl* element. URLs can include parameters which are utilized to adapt a rule document. In this case the name of the consumer or the name of the service may be required to generate specific management actions, e.g. notifications. We envision the dynamic rule set changing at runtime as a very important task. According to the grade of process behavior we are able to adapt the rules dynamically. For example, if the BPEL process runs as expected, we only want to notify the interested parties. As opposite, if the process violates the requirements, we wish to trigger more severe actions, like replacing some services inside the process. By combining rules into rule sets and change them at runtime, we are able to adapt the rules specifically to the process behavior. Thus we avoid the increasing of the complexity for the evaluating rules process by removing rules that are no longer needed from the memory. This is done simply by *activating* or *deactivating* rule sets.

As flexibility and changes play the central role in a SOA, like: contract modifications between partners, changes of partners, changing of endpoint URLs for services, or the service registry, rules have to be adapted accordingly. The *reload* action in BPRules permits reloading rules, by adding new rules to a rule set or overwriting existing rules at runtime. Rules may be updated from an URL.

A BPR rule consists of two parts: a *condition* and an *action* part. The condition specifies the constraints regarding the QoS requirements for the process that have to be evaluated. QoS constraints can be specified for process instances and sections of the process. The action part specifies what corrective actions are going to be undertaken in case that the condition was previously evaluated to true. The general form of a BPR rule may be seen below:

```
1 <rule id="rulename1">
2   <condition id="cond1">
3     <!--the QoS constraints for the business process -->
4   </condition>
```

```
5   <action id ="act1">
6     <!--the corrective measures to be undertaken-->
7   </action>
8 </rule>
```

4.1 A Business Process Scenario

In order to illustrate the BPRules language, we will consider a business process for requesting a driving license at the police office, also represented in Figure 3. Several web services are involved in the process: the *Police Service (PS)*, the *Medical Service (MS)*, two *Bank Services (BS1 and BS2)* and the *Photo Service (PHS1)*. For the process we define three sections. In the

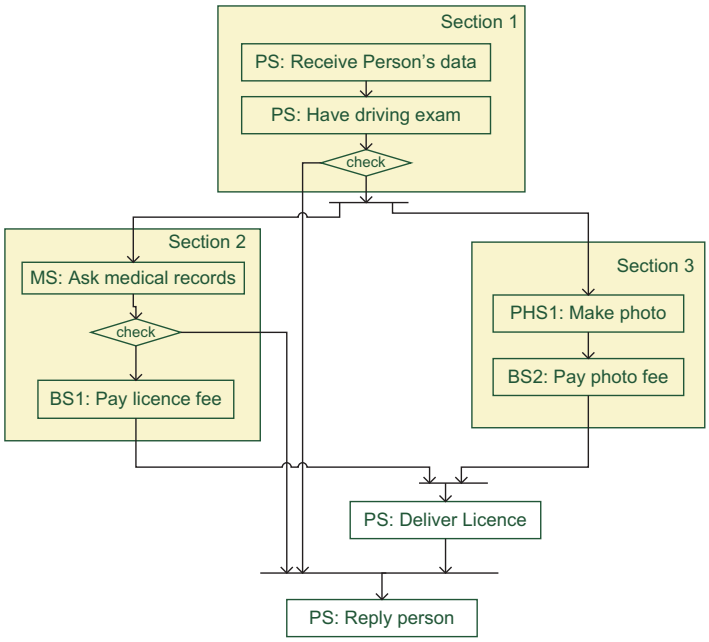

Figure 3: Driving Licence Business Process

first section, the *Police Service* cares about receiving the person's data and this person has to take his driving exam. If the driving exam was passed, the process continues with the sections *section2* and *section3* which are triggered in parallel. Within *section 2* the *Medical Service* is asked for the medical records of the person which are then checked for the person's health. *Bank Service 1* is called to pay the license fee. *Section 3* is responsible for the photo session. Finally, if everything went well, the driving license is delivered to the person by the *Police Service*. If the person hasn't passed the driving exam or isn't healthy, he will receive a message with this information.

As an example we define two rules for the process. The first rule illustrates how the behavior of a set of process instances influences the future behavior of the process, by triggering appropriate corrective actions. The rule states that if the process behaves extremely badly, that means that at least 30% of the process instances failed, the *red* rule set should be loaded at runtime. The *red*

rule set is loaded only when some serious corrective actions need to be done, for example, the replacement of services with alternative ones.

```
1 <rule id="scenario-rule1">
2   <condition id="failedinst">
3     <constraints>
4       <instances-subset function="MIN">30%</instances-subset>
5       <expression>
6         <property-check select="state">FAILED</property-check>
7       </expression>
8     </constraints>
9   </condition>
10  <action id="act1">
11    <setactive-ruleset id="red" setactive="true" />
12  </action>
13 </rule>
```

4.2 Constraints

The *BPR constraints* is the main construct to specify a condition of the process. Here we can select between process instances with certain properties (e.g. RUNNING instances) and also define the size of the set of process instances (see Table 2, e.g. MAX 40 instances) to which the QoS constraints apply. We can select, for example, process instances with a certain id or state (e.g. CANCELLED) to which the QoS constraints should apply to.

With the *instances-subset* functions we can specify the subset of process instances it takes for the management actions to be executed. An advantage in comparison to previous languages is that we can specify what should happen due to the behavior of a subset of the process instances. A set of functions gives the possibility to select a subset out of the entire set of process instances. Table 2 gives an overview of the functions that may be applied.

BPRules utilizes an expression and function syntax similar to WSLA [LKD+03]. Within BPR expressions, QoS constraints are specified for the entire business process or for the desired sub-orchestrations. An expression is a Boolean statement and contains a specific requirement or other nested expressions. BPR expressions can be applied to multiple sub-orchestrations by referencing an expression definition via the *select* attribute, and using the attribute *applysection* to specifying the section to which the expression applies.

The next rule defined for our *drivinglicence* process illustrates the QoS parameter dependability between sections. As *section2* and *section3* are triggered in parallel, it makes sense that the response time measured in *section2* is less or equal to the response time in *section3* (lines 3-8). Also the cost in *section3* (line 11) should be less than 1/3 of the total cost of the entire process (line 14).

```
1   <expression>
2     <and>
3       <expression id="express1">
4         <predicate type="Greater">
5           <QoSParameter applysection="section2">responsetime</QoSParameter>
6           <QoSParameter applysection="section3">responsetime</QoSParameter>
7         </predicate>
8       </expression>
```

```
9       <expression id="express2" applysection="section3">
10        <predicate type="Greater">
11          <QoSParameter>cost</QoSParameter>
12          <Function type="Divide" resultType="double">
13            <operand>
14              <QosParameter applysection="global">cost</QosParameter>
15            </operand>
16            <operand>
17              <Value>3</Value>
18            </operand>
19          </Function>
20        </predicate>
21      </expression>
22    </and>
23  </expression>
```

By using the *select* attribute in the expression, we may put one expression in correspondence to a previously defined expression identified by the *id* attribute, providing reusability. The next listing illustrates how the same expression *express2*, which was previously defined, is reused for several sub-orchestrations. The expression *express2* is applied once for the section *section1* and once for the section *section2*. An expression may contain other expressions linked through logic operators (e.g. AND, OR, NOT etc.). This is another possibility for creating relationships between QoS constraints of sub-orchestrations.

```
1 <or>
2       <!--reuse of the expression express2 -->
3       <expression select="express2" applysection="section1"/>
4       <expression select="express2" applysection="section2"/>
5 </or>
```

Table 2: Instances subset functions for the set of process instances

Functions	**Semantics**
MIN number (%)	The minimum number of instances or a percent number from the set of instances.
MAX number (%)	The maximum number of instances or a percent number from the set of instances.
EQUALS number(%)	Equals to a certain number or a percent number from the set of instances.
EXISTS	There exists at least one instance.
FORALL	All the instances from the set.

5 Monitoring and Management Prototype

This section gives an overview of the general workflow in our system. At development time, the business process is defined using the BPEL language. The architect proceeds with specifying the sections over the business process. Additionally, for each of the sections and the entire business

process, he states the QoS requirements using declarative rules, written in the BPRules language. These are evaluated at runtime by the Drools Rules engine from JBoss.

For the monitoring purpose, sensors are attached to activities of the business process. The monitoring system takes advantage of the BPEL engine *Oracle BPEL Process Manager* [OPM08] which supports attaching sensors without any changes to the business process. We use several types of sensors: activation sensors, which fire just before the activity is executed, completion sensors, which fire just after the activity is executed, and fault sensors, which fire when faults occur during the execution of the activity. Sensors are attached to BPEL activities apart from the process implementation, in separate XML files. The data received from the sensors is used for QoS computation of the process and its sections. The QoS measurement values are evaluated against the rules by the rules engine and the corresponding corrective actions are triggered.

6 Conclusion

We proposed a rule-based language for the management of BPEL processes. Our main concerns while designing the language were expressivity, reusability and separation of concerns. We keep the monitoring and QoS artifacts separate from the business logic. The BPRules language provides a flexible way to perform corrective actions on a business process, dependent on process executions.

By introducing QoS dependability between sections and loading rule sets at runtime, a more refined specification of QoS requirements is possible with BPRules. We propose our new language that offers features to react dynamically to changes in a flexible SOA.

In future we plan to extend our BPRules language with more constructs. We want to introduce, for example, time intervals to be able to manage the business process and its services differently during time. On the basis of the triggered management actions we will make evaluations and rankings of the web services and process.

Bibliography

[BBLC07] F. Baligand, D. L. Botlan, T. Ledoux, P. Combes. A Language for Quality of Service Requirements Specification in Web Services Orchestrations. In *4th International Conference, Workshops Proceedings, Service-Oriented Computing ICSOC 2006*. Pp. 38–49. Springer Berlin / Heidelberg, 2007.

[BCKZ08] S. Bleul, D. Comes, M. Kirchhoff, M. Zapf. Self-Integration of Web Services in BPEL Processes. In *Proceedings of the Workshop Selbstorganisierende, Adaptive, Kontextsensitive verteilte Systeme (SAKS)*. 2008.

[BG05] L. Baresi, S. Guinea. Towards Dynamic Monitoring of WS-BPEL Processes. In *Proceedings of the Third International Conference, Service-Oriented Computing - ICSOC 2005*. Pp. 269–282. Springer Berlin / Heidelberg, 2005.

[BGCK08] S. Bleul, K. Geihs, D. Comes, M. Kirchhoff. Automated Management of Dynamic Web Service Integration. In *Proceedings of the 15th Annual Workshop of HP Soft-*

ware University Association (HP-SUA). Infocomics-Consulting, Stuttgart, Germany, Kassel, Germany, 2008.

[BGG04] L. Baresi, C. Ghezzi, S. Guinea. Smart monitors for composed services. In *Proceedings of the 2nd international conference on Service oriented computing*. Pp. 193–202. ACM, 2004.

[BGP06] L. Baresi, S. Guinea, P. Plebani. WS-Policy for Service Monitoring. In *6th International Workshop, Technologies for E-Services 2005*. Pp. 72–83. Springer Berlin / Heidelberg, Trondheim, Norway, 2006.

[BRL07] F. Baligand, N. Rivierre, T. Ledoux. A Declarative Approach for QoS-Aware Web Service Compositions. In Krmer et al. (eds.), *Proceedings of the Fifth International Conference, Service-Oriented Computing ICSOC 2007*. Pp. 422–428. Springer Berlin/ Heidelberg, Vienna, Austria, 2007.

[CPEV04] G. Canfora, M. D. Penta, R. Esposito, M. L. Villani. A Lightweight Approach for QoS-Aware Service Composition. Technical report, Research Centre on Software Technology University of Sannio, New York, USA, 2004.

[CSHM06] A. Charfi, B. Schmeling, A. Heizenreder, M. Mezini. Reliable, Secure, and Transacted Web Service Compositions with AO4BPEL. In *Proceedings of the European Conference on Web Services (ECOWS'06)*. Pp. 23–34. IEEE Computer Society, 2006.

[Dro08] Drools. JBoss. 2008.
http://www.jboss.org/drools/

[JEA+07] D. Jordan, J. Evdemon, A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guízar, N. Kartha, C. Liu, R. Khalaf, D. König, M. Marin, V. Mehta, S. Thatte, D. van der Rijn, P. Yendluri, A. Yiu. Web Services Business Process Execution Language Version 2.0. OASIS, Apr. 11, 2007. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf.

[LKD+03] H. Ludwig, A. Keller, A. Dan, R. P. King, R. Franck. Web Service Level Agreement (WSLA) Language Specification. IBM, 2003. http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf.

[OPM08] Oracle BPEL Process Manager. 2008.
http://www.oracle.com/lang/de/appserver/bpel_home.html

[TPP02] V. Tosic, B. Pagurek, K. Patel. WSOL A Language for the Formal Specification of Various Constraints and Classes of Service for Web Services. Technical report, Carleton University, Canada, 2002.
http://citeseer.ist.psu.edu/674231.html

[ZBN+04] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, H. Chang. QoS-Aware Middleware for Web Services Composition. In *IEEE Transactions on Software Engineering*. Pp. 311–327. IEEE Press, 2004.