Domain-Specific Service Selection for Composite Services

Oliver Moser, *Student Member*, *IEEE*, Florian Rosenberg, and Schahram Dustdar, *Senior Member*, *IEEE*

Abstract—We propose a domain-specific service selection mechanism and system implementation to address the issue of runtime adaptation of composite services that implement mission-critical business processes. To this end, we leverage quality of service (QoS) as a means to specify rigid dependability requirements. QoS does not include only common attributes such as availability or response time but also attributes specific to certain business domains and processes. Therefore, we combine both domain-agnostic and domain-specific QoS attributes in an adaptive QoS model. For specifying the service selection strategy, we propose a domain-specific language called VieDASSL to specify so-called selectors. This language can be used to specify selector implementations based on the available QoS attributes. Both the QoS model implementation and the selectors can be adapted at runtime to deal with changing business and QoS requirements. Our approach is implemented on top of an existing WS-BPEL engine. We demonstrate its feasibility by implementing a case study from the telecommunication domain.

Index Terms—Service composition, quality of service, monitoring, service selection, domain specific languages.

1 Introduction

Enterprise applications are typically constituted of a number of interorganizational and mission critical business processes [1], [2]. Such process-driven applications orchestrate a number of backend services and communicate with external services offered by business partners. These processes represent an integral part of a company's daily business operations. To be competitive in the market, a high degree of agility of those processes and the underlying IT systems is important. It allows us to adapt rapidly and cost efficiently in response to changes in the business environment. Therefore, such enterprises increasingly adopt the Service-Oriented Architecture (SOA) paradigm for their applications [3]. It allows us to expose the core business logic as services, which are then orchestrated to implement the business processes using composite services [4], [5]. This has a number of benefits, such as loose coupling, platform independence, and integration capabilities of existing legacy systems. Web services technology emerged as the de facto standard for the implementation of services [6].

Adopting SOA as a paradigm itself does not necessarily solve the problem of provisioning mission-critical processes in interorganizational settings. However, it can provide some of the required building blocks as outlined above [7]. A key challenge in SOA, in particular for service composition infrastructures, is to develop mechanisms to achieve so-called *self-adaptive* and *dependable* service compositions [8], [9]. A self-adaptive and dependable composite service is capable of reacting to changes in the environment [10] and has the ability to deliver services that can justifiably be trusted [11].

Quality of service (QoS) is an important aspect of modeling and managing self-adaptive and dependable composite services and their individual services [12]. Dependability properties of Web services can be categorized and measured by using QoS attributes from different providers [13], [14]. In the service composition context, the QoS of the overall composite service is particularly important. The QoS of the individual services usually determines the QoS of the overall composition [15]. Thus, a single service with bad QoS may cause deteriorating performance or failure of the composite service.

In our QoS model, we broadly distinguish between two categories according to their business *domain*. The first category includes *domain-agnostic* QoS attributes used in numerous business domains and processes. It encompasses a broad range of attributes, among them performance and dependability ones such as high availability and throughput, fast response time, and a minimized planned downtime of the system [13], [16], [17]. The second category comprises *domain-specific* QoS attributes that are only applicable to certain business domains and processes, such as voice or data traffic rates or the call setup fee in telecommunication processes. Typically, such QoS attributes have no meaning in other domains.

A major problem with existing approaches is that current composition infrastructures only provide limited support for defining and handling even simple QoS-based adaptations at runtime [7], [16], [15], [18], [19], [20], [21], [22]. An example of such a QoS-based adaptation is the replacement or rebinding of a bad performing service that is part of a

- *O. Moser and S. Dustdar are with the Distributed Systems Group, Information Systems Institute, Vienna University of Technology, Argentinierstrasse 8, Vienna A-1040, Austria. E-mail: {Moser, Dustdar}@infosys.tuwien.ac.at.*
- *F. Rosenberg is with the IBM T.J. Watson Research Center, 19 Skyline Dr, Hawthorne, NY 10532. E-mail: Rosenberg@us.ibm.com.*

Manuscript received 17 Nov. 2009; revised 26 Apr. 2010; accepted 17 Mar. 2011; published online 1 Apr. 2011.
Recommended for acceptance by F. Pohl and C. Ghezzi.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSESI-2009-11-0357.
Digital Object Identifier no. 10.1109/TSE.2011.43.

composite service definition with a compatible and better performing alternative service. Another example is a change of a QoS attribute or the introduction of a new QoS attribute during the lifetime of a composite service, especially domain-specific ones. Handling such adaptations requires 1) mechanisms to define which service should be selected based on the desired QoS attributes and 2) appropriate runtime capabilities that define how the newly adapted composite service can be executed. However, most of these adaptations currently require a system engineer to manually reengineer and redeploy the process, which implies a disruption of the provisioning process.

In this paper, we focus on QoS-based adaptations of composite services with regard to service selection and binding. In particular, the proposed approach addresses the following requirements:

R1. The QoS model implementation should be adaptable to capture the need to add, remove, or change QoS attributes at runtime.
R2. The service selection mechanism should support runtime adaptability to address the need to change the logic how a service is selected (e.g., based on changes in the QoS model according to R1 or changing business requirements). A tailored Domain-Specific Language (DSL) should support the specification of the selection logic.
R3. The adaptations as specified in requirements R1 and R2 should be carried out at runtime and prevent any downtime of the composition infrastructure. Additionally, manual intervention by a developer or system engineer should not be necessary, e.g., to undeploy and redeploy the composite service.
R4. The system implementing the selection mechanism needs to be unobtrusive. This means that they do not need to be changed when using different composition engine implementations (e.g., different WS-BPEL engine implementations).

To address these requirements, we propose a *domain-specific service selection mechanism* and *system implementation* on top of a VieDAME-enabled WS-BPEL runtime [23]. To this end, we use the MAPE loop [24] (Monitor, Analyze, Plan, and Execute) from the autonomic computing area to achieve the required degree of runtime self-adaptation capabilities for composite services. In particular, we leverage an adaptive QoS model implementation in the sense that it can be extended and changed at runtime. For the specification of the service selection strategy, we use a domain-specific language called VieDASSL to implement so-called *selectors*. The language can be used by domain experts to define such runtime adaptable selectors based on the QoS attributes in the adaptive QoS model. This approach ensures that changes in the QoS model and selectors can be handled at runtime without disruption of the business processes. An evaluation of the proposed system using a case study from the telecommunications domain demonstrates the effectiveness of the system. The evaluation also discusses the main benefits of our approach with regard to different stakeholders. The proposed contribution makes the following explicit assumptions based on our insights and access to expert knowledge in the telecommunication domain:

- We consider composite services where usually only a small number of services are replaceable at runtime.
- We assume that only a small number of alternative services (i.e., max. 1-5) are available for each replaceable service in a service composition.
- For the QoS model, we assume the units of measurement for each QoS attribute are fixed and cannot be dynamically adapted.
- We do not consider autonomic discovery of alternative services as part of our approach.

The remainder of this paper is organized as follows: Section 2 presents an illustrative example from the telecommunication domain. Section 3 introduces the domain-specific service selection approach. A detailed evaluation of our system is discussed in Section 4. Section 5 presents related work with respect to the main contributions of this paper. Finally, Section 6 concludes this paper and outlines some future work.

2 Illustrative Example

This section presents the *ManagedRoaming* (MR) business process as an illustrative example which is referenced throughout the paper. The MR process enables the telecommunications enterprise *Phonyfone* to implement their roaming provisioning using a composite service. Thereby, Phonyfone can offer the best rates to their customers when they use their cell phones in foreign networks.

Prior to the discussion of the MR process, we explain the relevant terms from the telecommunication domain. A *subscriber* represents a customer in Phonyfone's information systems, while a *deal* represents an orderable service offered by Phonyfone (e.g., mobile e-mail access or managed roaming). The term *roaming* describes a network operator's capability to extend their services, both voice and data related, to a network other than the home network. The term MSISDN (*Mobile Subscriber Integrated Services Digital Network Number*) is, simply put, the cellphone number of a SIM card. The *Subscriber Identity Module* (SIM) card is a removable chip that stores subscriber information in cellphones. Finally, the IMSI (*International Mobile Subscriber Identity*) and the ICCID (*Integrated Circuit Card ID*) are identifiers stored on the SIM card and both are globally unique. The IMSI is the key for the network operator to identify a subscriber, while the ICCID identifies the SIM card itself. For a detailed discussion of roaming in wireless networks, please refer to [25].

2.1 The ManagedRoaming Process

Fig. 1 illustrates the process using a slightly relaxed BPMN (Business Process Modeling Notation) diagram [26]. This means that we annotated the diagram with two concrete roaming partners including their values for certain QoS attributes. Please note that fault and compensation handling are not shown for brevity.

Initially, the MR process waits for an incoming *RegisterRoamer* message, which instantiates the process.

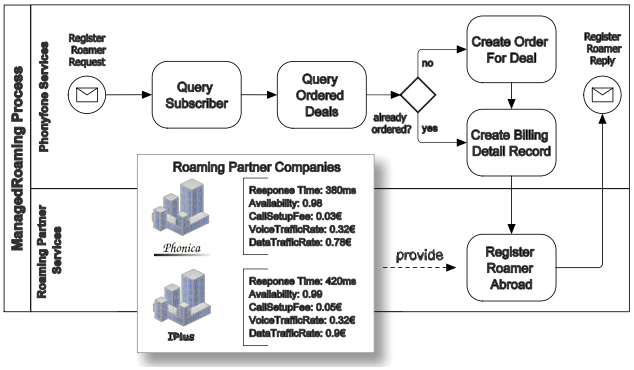

Fig. 1. ManagedRoaming process.

The message includes the MSISDN of the subscriber that wants to order the MR deal.

The first activity maps the subscriber's MSISDN to the IMSI by invoking the *QuerySubscriber* service. This is necessary because some of the services in the MR process require the IMSI of the related SIM card. Provided that the input MSISDN matches a subscriber object in Phonyfone's subscriber database, the service returns the details of the subscriber, including the IMSI.

The next activity checks whether the subscriber already has an order for the MR deal in Phonyfone's subscription platform. The *QueryDeal* service provides the required functionality, accepting the MSISDN as an input parameter. It returns relevant data of all deals that the subscriber has ordered.

If there is no order for the MR deal, the process creates it by invoking the *CreateOrder* service. It takes the MSISDN and the desired deal name, e.g., ManagedRoaming, as input parameters.

The next process activity represents the creation of a *Billing Detail Record* (BDR for short). The BDR is required internally for Phonyfone's billing and accounting system.

Up to this point in the process execution, only on-premises services are involved in service interaction. This means that Phonyfone has full control over the participating services. The final and most important activity in the MR process covers the registration of the subscriber in the foreign network. For this purpose, Phonyfone contracts several roaming partners. Fig. 1 exemplifies two of these contractors, *IPlus* and *Phonica*. These partner companies have to offer an interface that abstracts the details of subscriber registration in the foreign network. The *RegisterRoamerAbroad* activity invokes this roaming interface and provides the MSISDN, the home network provider, e.g., Phonyfone, and the IMSI and ICCID of the SIM card to the external partner service. The process designer, who links the corresponding partner service with the activity in the MR process, then defines which contracted roaming partner is used for the RegisterRoamerAbroad activity.

2.2 Nonfunctional Requirements

When considering the nonfunctional requirements of the MR process, the overall execution time is an important hard-limit of the process. If it exceeds a certain threshold, usually a few seconds, the overall call setup will fail. This results in the worst-case scenario where customers are unable to use their phones. Thus, process execution time is crucial, implying that both the availability and the response time of the partner services are important to the overall process performance.

Additionally, the external roaming partners have different fees and rates associated with their services. The *CallSetupFee* are costs that incur during the initial setup of the roaming call and have to be paid by Phonyfone. The *VoiceTrafficRate* and the *DataTrafficRate* are costs paid by the customer. They are charged for the actual duration of the call and the transferred data. Fig. 1 shows sample values for the aforementioned nonfunctional properties.

From an operational perspective, Phonyfone uses composition engines from various vendors. This stems from the fact that during Phonyfone's company history, several acquisitions of smaller telcos have been made, which are now operating as subsidiaries of Phonyfone. These subsidiaries also run the MR process, but eventually use a different composition infrastructure for its execution.

Phonyfone's main objectives are to minimize the costs for them and their customers and to maximize the overall performance of the MR process. As a consequence, Phonyfone needs a system to *dynamically adapt* their MR process, independent of which composition engine is executing the MR process (cf., requirement R4 from Section 1). Thereby, subscribers can be registered in the foreign network using a roaming partner service that complies with these objectives.

In this regard, the term dynamic refers to several features of such an adaptation. First, it must remove the hard linked dependencies on partner services (cf., requirement R2). As described above, there are alternative services available for the RegisterRoamerAbroad activity. However, which service is invoked is determined at deployment time or even during the design of the process. Using another service for the RegisterRoamerAbroad activity requires process adaptations, which typically cause system downtimes where the MR process is unavailable. Such unavailability hinders customers using their phones and therefore causes monetary loss and affects Phonyfone's reputation as a telco provider. Selecting the roaming partner service and adapting the MR process *at runtime* prevents such downtimes (cf., requirement R3). Please note that due to contractual obligations, an autonomic discovery of such alternative services is not applicable in most telco enterprise scenarios.

Second, rate changes or outages of services cannot be foreseen, as well as additional nonfunctional requirements that emerge due to new market requirements. New roaming partners with better offerings than the currently contracted partner may enter the market. Therefore, a viable adaptation solution 1) must be capable of reflecting nonfunctional requirements that need constant monitoring, such as response time of partner services, and 2) must provide corresponding extensibility mechanisms to address newly emerging nonfunctional requirements (cf., requirement R1).

3 Domain-Specific Service Selection

This section presents our approach for domain-specific service selection. First, we discuss the fundamental concepts of the proposed system. Second, we integrate these fundamentals into the overall system design. Finally, we

TABLE 1
Supported Nondeterministic QoS Attributes

QoS Attribute	Formula	Unit
Response Time	$\frac{1}{\#requests} \sum_{i=1}^{n} rt_i$	Milliseconds
Throughput	$\frac{\#requests}{second}$	Requests per Second
Availability	1 - $\frac{downtime}{uptime}$	Percent
Accuracy	1 - $\frac{\#failedrequests}{\#totalrequests}$	Percent

present a prototype implementation of the system design to demonstrate its feasibility.

3.1 Theory of Operation

The system design that we will present in Section 3.2 leverages certain conceptual building blocks, which are extensively discussed in the following three sections. First, Section 3.1.1 introduces a model for categorizing and reasoning about service quality. Second, the conceptual approach for dynamically adapting composite services is explained in Section 3.1.2. Finally, both of these concepts are merged into mechanisms that provide declarative management of the dynamic adaptations of a composite service, as presented in Section 3.1.3.

3.1.1 Adaptive QoS Model

The term *quality of service* can refer to different aspects. In general, QoS is used to describe service consumer satisfaction. We encounter QoS in our daily routine, be it customer service in your preferred coffee shop, voice quality in mobile networks, or the page load time of an online trading platform. In the following, we explain the different QoS attributes and how they can be reflected in a QoS model.

Quality aspects differ in many regards, such as measurability and objectiveness. Likewise, diverse domains feature different requirements with respect to service quality. This leads to two distinctive features that we will examine hereafter.

First, QoS attributes differ with respect to *determinism*. For certain domains, it might be well suited to observe only QoS attributes whose values are known in advance. These values are not intended to constantly change during the service offering, such as the VoiceTrafficRate from our ManagedRoaming process. It is contracted by the service offering party, e.g., IPlus, and the service consuming party, Phonyfone, prior to the service invocation and will rarely change. On the contrary, values of other QoS attributes might change over time. The response time of the roaming partner services in our ManagedRoaming process is a dynamic QoS attribute whose values are subject to constant change. Therefore, we follow related work [16] and distinguish between *deterministic* and *nondeterministic* QoS attributes. Deterministic QoS attributes are attributes whose values are known prior to service invocation. Values of nondeterministic QoS attributes are not known in advance and require constant monitoring. Table 1 list nondeterministic QoS attributes supported by our approach. A detailed discussion of the listed QoS attributes can be found in [17]. Please note that Response Time and Throughput are

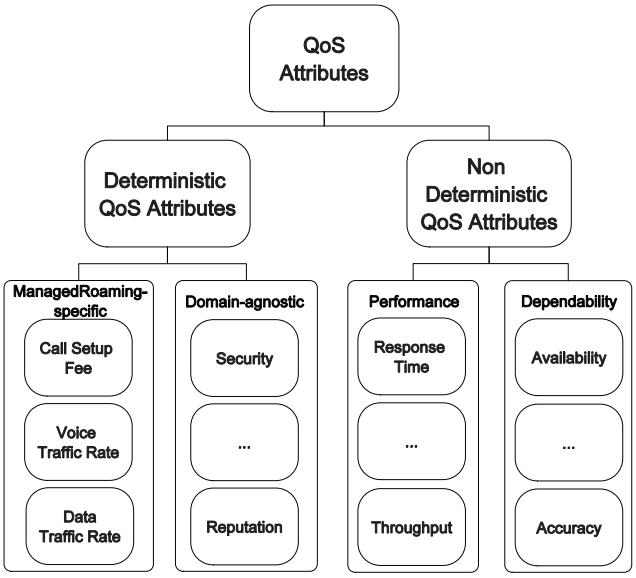

Fig. 2. QoS model for ManagedRoaming.

Performance-related QoS attributes, whereas Availability and Accuracy are *Dependability*-related QoS attributes. The value rt_i, where $0 < i < n$, denotes one observation of the response time of a service in a series of n response times of this service.

Second, QoS attributes can be distinguished regarding their *applicability*. In this regard, some nonfunctional properties, such as reputation or availability, are common to many service implementations. Such properties are *domain-agnostic* in the sense that their applicability is not restricted to a certain business domain. *Domain-specific* QoS attributes, on the other hand, are viable to their specific business domain only. As an example, consider the CallSetupFee from our ManagedRoaming process. Although essential to our process, this attribute is of no relevance for other processes, such as purchase order or loan approval scenarios.

To reason about QoS attributes, a *QoS model* is needed [16]. Such a QoS model organizes the different QoS attributes for a particular use case, e.g., a business process such as the ManagedRoaming process. This model can be arbitrarily extended to support additional attributes. Fig. 2 exemplifies a QoS model using the concepts explained above.

It is important to note that Phonyfone, and all other mobile operators, are subject to certain market situations. Both regulators as well as marketing requirements can impose constraints on operators. A regulator could define the maximum price for text messages originating from a foreign network to protect consumers from high roaming fees. On the other hand, marketing departments are often obliged to react on short-term offers by the competition. In many cases, being the first operator to launch a new product or a promotional offer is the key factor that decides success or failure. As an example, consider Phonyfone launching a product package that includes the free usage of a value added service, such as mobile TV. Such mobile TV solutions are provided by third party media gateways. Keeping the interenterprise fees between Phonyfone and the various media gateway partners low is required to make

TABLE 2
Summary for Selector Implementation Approaches

Feature	Static Selector	Parameterizable Selector	Scripted Selector	Domain-specific Selector
Design-time Deployable	✓	✓	✓	✓
Runtime Deployable	✗	✗	✓	✓
Design-time QoS Weighting Adaptation	✓	✓	✓	✓
Runtime QoS Weighting Adaptation	✗	✓	✓	✓
Runtime Selection Strategy Adaptation	✗	✗	✓	✓
Implementation Agnostic Authoring	✗	✗	✗	✓

the product package profitable. However, a media partner could decide to charge different rates for different content, e.g., a free news channel in contrast to expensive adult content, or simply increase the rates on existing content.

All too often, such market situations cannot be foreseen and additional domain-specific QoS attributes may need to be added to the QoS model. Ideally, the implementation of such a QoS model supports runtime adaptability. This allows us to dynamically tailor the QoS model to specific needs and domains (e.g., the telco domain). Therefore, we propose an *adaptive QoS model* implementation to eliminate the need for a system downtime when adding, changing, or removing QoS attributes. Static QoS model implementations or implementations that allow only design time modifications do not support the adaptability required by Phonyfone.

3.1.2 Runtime Service Selection with Selectors

When dealing with composite services, the QoS of each atomic service is a highly influential factor for the QoS of the whole service composition. In many cases, the unavailability of a single service results in unavailability of the service composition. For the ManagedRoaming process, this means that an unresponsive, faulty, or unavailable partner service would affect the execution time or accuracy of the whole process. Therefore, we proposed an approach to facilitate runtime exchange of services during process execution in our previous work [23]. In particular, it allows us to substitute an underperforming partner service with a suitable and reasonably performing alternative partner service. We presented the concept of *selector*s to dynamically replace partner service instances by implementing a specific service selection strategy. Its primary task is to determine which service, from a set of available alternative services, matches the service selection strategy best. If none of the alternative services outperforms the originally defined service in terms of the service selection strategy, the selector returns the originally defined service.

The selection strategies reflected by a selector implementation can range from trivial randomized or single QoS attribute optimizing strategies to complex strategies where several QoS attributes need to be included in the selection decision. When handling multiple QoS attributes in a selection strategy, it might further be desired to express conditional selection and weight the different QoS attributes according to their relative importance for the selection strategy. Besides the actual selection semantics, another important feature of such a selector component is *how* and *when* the selection strategy needs to be defined. In the following, we discuss several approaches for selector implementations. Table 2 summarizes the features of the selector implementation approaches.

Static Selectors implement a hard-coded selection strategy and certainly represent the simplest solution. They have the inherent drawback that any change to the selection strategy implies a system downtime. A modify-recompile-redeploy cycle and a restart of the process execution environment are required.

Parameterizable Selectors include parameters that can be adapted during runtime. As an example, consider a selection strategy where the average response time of a service should be minimized and the availability should be maximized. Assuming that the response time is more important than the availability, the QoS attributes need to be weighted. This weighting can be adjusted using weighting *parameters* in the selection strategy implementation. If these parameters are exposed, e.g., through a user interface, the weighting can be adapted at runtime. This might be necessary when the requirements for the selection strategy change. However, there are two major drawbacks of Parameterizable Selectors. First, implementation specific know-how is needed to adapt the underlying selection strategy. Second, if modifications of the selection strategy cannot be covered via parameter changes, system downtime for the deployment of the adapted selector implementation is inevitable.

Scripted Selectors can be "hot-plugged" into the execution environment at runtime. The source code of the selection strategy implementation is interpreted during runtime rather than compiled at design time. Thus, Scripted Selectors can be leveraged to provide true runtime adaptation of the service selection strategy. However, the implementation of the selection strategy does not abstract from programming language details. This makes Scripted Selectors practically unusable for domain experts that are not capable of the required programming language skills.

*Domain-Specific Selector*s inherit the runtime deployability features of Scripted Selectors. However, instead of relying on a general purpose programming language, Domain-Specific Selectors leverage a DSL to describe the underlying service selection semantics at the abstraction level of the problem domain. Additionally, it overcomes the drawbacks of the previously listed selector approaches (cf., Table 2) with regard to ease of strategy adaptability.

3.1.3 A Domain-Specific Service Selection Language

The *Vienna Domain-Specific Service Selection Language* (VieDASSL) is a language that implements a domain-specific selector approach as described in the previous

TABLE 3
QoS Attribute to VieDASSL QoSSymbol Mapping

QoS Attribute	QoSSymbol	Unit
Response Time	responseTimeMs	milliseconds
Throughput	throughputReqPerSec	requests per second
Availability	availabilityPercent	percentage
Accuracy	accuracyPercent	percentage

section. In particular, VieDASSL addresses requirement [R2] from Section 1 to provide a flexible, yet reasonably manageable specification language for service selection strategies. It should allow domain experts to create and adapt these selection strategies without programming language know-how. Additionally, it should leverage the QoS model described in Section 3.1.1 to maximize flexibility with respect to QoS attributes.

Supported QoS Attributes. For each QoS attribute from Fig. 2, VieDASSL provides a *QoSSymbol* that represents the QoS attribute in the language. Table 3 lists the mapping of QoS attributes to QoSSymbols supported by VieDASSL and their related unit of measurement.

For brevity, only nondeterministic QoSSymbols are shown in Table 3. The full set of QoSSymbols can be accessed on the project website [27]. However, it is important to note that the set of supported deterministic QoSSymbols can be arbitrarily extended during runtime, in compliance with requirement R1 (cf., Section 1). Furthermore, the QoSSymbols from Table 3 include a hint about their unit of measurement in their name. When adding new QoSSymbols to the set of deterministic QoSSymbols, their name should also include the related unit of measurement, both to improve traceability and make QoSSymbols self-explanatory.

The values of the QoS attributes in Table 3 are based on *all* available monitoring data for a particular service. In addition, our approach also provides *time-windowed* versions of these QoS attributes. By default, the 5, 10, and 15 minute averages are available. The QoSSymbols of these time-windowed attributes are equal to the QoSSymbols from Table 3, with the suffix *5Min*, *10min*, or *15min* added at the end of the QoSSymbol. As an example, the response time over the last 10 minutes can be referred to by the QoSSymbol *responseTimeMs10Min*.

Language Description. VieDASSL supports three types of rules that represent the building blocks for defining service selection strategies. Please note that the service defined in a business process definition, which is subject to replacement by alternative services, is hereinafter referred to as the *original service*.

Every selection strategy modeled with VieDASSL has at least one *FactorRule*. FactorRules are used to calculate a *score* for each of the services by considering 1) a QoS attribute, reflected in a rule by its corresponding QoSSymbol, and 2) a weight $W_j \in [0,1]$ and $\sum_{j=1}^{k} W_j = 1$ for k QoS attributes used in the rules. The weighting factor expresses user preferences with regard to the considered QoS attributes. As with all other rules which we will discuss shortly, FactorRules have to be enclosed in a `selection{}` block. Listing 2 shows a VieDASSL selection specification having a single FactorRule.

Listing 1. EBNF of VieDASSL

```
 1 <SelectionRule> ::= selection
   '{' <RuleBody> '}'
 2 <RuleBody> ::= <FactorRule> |
   <SumSelectionRule> |
   <ConditionalSelectionRule>
 3 <SumSelectionRule> ::= <FactorRule>
   {<FactorRule> }
 4 <ConditionalSelectionRule> ::= 'inCase('
   <QoSSymbol> ',' <Comparator> ','
   <QoSAttributeValue>')'
 5                       {' <RuleBody> '}
 6                       {' <RuleBody> '}'
 7 <FactorRule> ::= <MinimizingRule> |
   <MaximizingRule>
 8 <MinimizingRule> ::= 'min(' <QoSSymbol> ','
   <QoSAttributeWeighting> ')'
 9 <MaximizingRule> ::= 'max(' <QoSSymbol> ','
   <QoSAttributeWeighting> ')'
10 <Comparator> ::= 'less' | 'greater'
   | 'equals'
11 <QoSAttributeWeighting> ::= '0' | '1' | '0'
   '.'<Digit>
12 <QoSAttributeValue> ::= <Float>
13 <Float> ::= <Digit> '.' <Digit> {<Digit>}
14 <Digit> ::= '0' | '1' | '2' | '3' | '4' | '5' |
   '6' | '7' |> '8' | '9'
15 <Letter> ::= 'a' |···| 'z' | 'A' |···| 'Z'
16 <Character> ::= <Letter> |'_'
17 <QoSSymbol> ::= <Letter> { <Character> |
   <Digit> }
```

Listing 2. Simple Factor Rule

```
1 selection{
2   max(availabilityPercent, 1)
3 }
```

A selector configured with the specification from above would try to maximize the `availability` QoS attribute with a weighting factor of `1`. Thus, it would pick the service that has the highest availability of all available alternative services. Of course, with only a single FactorRule, the weighting is of no relevance to the total score.

To enable the use of multiple QoS attributes in a VieDASSL definition, a *SumSelectionRule* can contain several FactorRules. Similar to [16], we use Simple Additive Weighting (SAW) [28] to determine the score of each alternative service. The algorithm is applied in two phases. First, a *scaling phase* transforms each considered QoS attribute value into a scaled attribute value $v \in [0,1]$. This process allows for 1) *positive* and *negative* QoS attributes and 2) *uniform measurement*. For positive QoS attributes, higher values mean higher quality, e.g., service availability. To the contrary, for negative QoS attributes, lower values mean higher quality, e.g., service response times. Uniform measurement enables the range and unit independent comparison of different QoS attributes, e.g., to compare

availability, measured as a probability, and response times, measured in milliseconds. Please note that support for various manifestations of different units of measurement is outside of the scope of this work (e.g., a roaming partner might have listed its CallSetupFee in EUR while another roaming partner might list the same attribute in USD). Second, the *weighting phase* computes the overall quality score using the values provided by the scaling phase. Thus, the overall quality score for a service is determined by the formula $\sum_{j=1}^{k} sv_j * W_j$, where sv_j denotes the scaled value for a QoS attribute for k different QoS attributes. A detailed discussion of both the scaling and the weighting phase of SAW can be obtained from [16].

As an example, the specification shown in Listing 3 defines a VieDASSL selection strategy that focuses on the service availability QoS attribute, but additionally takes the average response time into consideration. The higher importance of the service availability over the average response time is expressed by the higher weighting value for the availability attribute.

Listing 3. SumSelectionRule

```
1 selection {
2   max(availabilityPercent, 0.75)
3   min(responseTimeMs, 0.25)
4 }
```

In simple scenarios, the SumSelectionRule might be well suited to model selection strategies that incorporate several QoS attributes. In scenarios where a threshold for a particular QoS attribute must be enforced, a *ConditionalSelectionRule*, as shown in Listing 4, is applicable. It consists of 1) a condition, 2) a positive, and 3) a negative rule. The condition (line 2) includes a QoSSymbol, a comparator, and a threshold value. If it evaluates to true, the score of the positive rule (lines 3-5) is returned; otherwise, the score of the negative rule (lines 7-8) is returned. Moreover, conditional rules can be nested, which allows for modeling complex selection strategies. Listing 4 quotes a ConditionalSelectionRule that picks the most cost effective alternative service, provided that the average response time does not exceed 500 milliseconds. If the threshold is exceeded, we fall back to the SumSelectionRule described in Listing 3. Finally, Listing 1 shows the VieDASSL language specification in EBNF.

Listing 4. ConditionalSelectionRule

```
 1 selection {
 2   inCase (responseTimeMs, less, 500.0) {
 3     min(callSetupFeeEur, 0.2)
 4     min(voiceTrafficRateEur, 0.6)
 5     min(dataTrafficRateEur, 0.2)
 6   } {
 7     max(availabilityPercent, 0.75)
 8     min(responseTimeMs, 0.25)
 9   }
10 }
```

Selection Postprocessors. Under certain conditions, a Selector that naively applies the score-based ranking mechanism described above can face one considerable problem. Choosing the top scored alternative service for each process invocation can overload the selected alternative service in very high load scenarios. This renders the runtime service selection useless or even making the situation worse. To address this and similar issues, our Selector approach supports the concept of *Selection Postprocessors*. A Selection Postprocessor is provided with the map of alternative services and their related scores. It performs an additional selection among these top scored services. One approach for such a Selection Postprocessor would be a round-robin selection among the top ranked services. Another approach, the *weighted randomized selection*, is described next.

TABLE 4
Service Scores for Weighted Randomized Selection

Service Provider	Score	Selection Probability
Phonica	187.5	0.55
IPlus	151.3	0.44
Roamwave	98.7	0.0

A prerequisite for a reasonable weighted randomized selection is that the score of the alternative services that are subject to the selection does not fall below a certain *relevance threshold*. This prevents bad performing services from being considered for the selection. To explain our approach for this problem, we assume an additional roaming service provider, *Roamwave*. The corresponding scores are listed in Table 4.

The table also features the related selection probabilities for each alternative service provider. For the example, the relevance threshold has been set to 0.20, meaning that only services that have a score above or equal to $minimumScore = score_{max} * (1 - rt)$ where $score_{max}$ is the highest score of all alternative services and rt is the relevance threshold are taken into account for the weighted random selection. As the score of the Roamwave partner service is below $minimumScore = 150$, it is not considered further and its selection probability is 0. The probability for a remaining service S' is calculated by $\frac{s_j}{\sum_{i=0}^{n} s_i}$ for n remaining alternative services and s_j being the score of S'.

Algorithm 1 shows the simplified algorithm in pseudocode. The algorithm is provided with a map structure that holds service references and their related scores ($SCRS$) as well as the relevance threshold parameter (RT). Lines 2-5 initialize the required data structures. Line 6 calculates the minimum score for a service to be eligible for consideration based on the relevance threshold parameter RT. The first loop (lines 7-9) calculates the cumulative score of the available services. The second loop (lines 10-13) calculates the selection probability for each service and stores this information in the `probabilities` map. Finally, the third loop (lines 14-24) determines a candidate service. First, Lines 16-19 check if a service is eligible for selection using the relevance threshold as described above. Second, by evaluating if the difference between a random number $random$ (where $0 \leq random \leq 1$) and the service probability $probability$ (where $0 \leq probability \leq 1$) is positive (line 21), a

service is finally chosen. By default, the algorithm returns the service with the highest score (Line 25).

Algorithm 1. Weighted Random Selection

```
 1: function WEIGHTEDRANDSELECTION (SCRS, RT)
 2:     probabilities ← ∅            ▷ probabilites is a map structures
 3:     totalScore ← 0
 4:     random ← RANDOM(1)    ▷ random value between 0 and 1
 5:     defaultService ← FINDMAXSCORESERVICE(SCRS)
 6:     minimumScore ← max(SCRS.values)*(1 − RT)
 7:     for all service_i ∈ SCRS do
 8:         totalScore ← SCRS(service_i) + totalScore
 9:     end for
10:     for all service_i ∈ SCRS do
11:         probability ← SCRS(service_i)/totalScore
12:         probabilites(service_i) ← probability
13:     end for
14:     for all service_i ∈ probabilities do
15:         probability ← probabilities(service_i)
16:         score_i ← SCRS(service_i)
17:         if score_i < minimumScore then
18:             continue
19:         end if
20:         random ← random − probability
21:         if random < 0 then
22:             return service_i
23:         end if
24:     end for
25:     return defaultService
26: end function
```

The following section integrates the adaptive QoS model and the domain-specific service selection language we presented above and examines the features of the resulting service selection runtime.

3.2 A Domain-Specific Service Selection Runtime

For the design of such a domain-specific service selection runtime, a major goal was that the resulting system should be as *unobtrusive* as possible (cf., requirement R4 from Section 1). When referring to unobtrusiveness, we wanted to achieve two important system features. First, the service selection runtime should be implementation agnostic. Although the prototype implementation discussed in Section 3.3 refers to BPEL as the underlying service composition system, the concepts and patterns incorporated into our system design are generic and, to a large extent, applicable to any message-based service composition technology. Second, the orchestrated composition runtime should be unaware of the service selection runtime. No modifications in the composition runtime should be necessary for the service selection runtime to cooperate. These two requirements imply that a viable approach has to operate on a level that is common to all service composition runtimes, the message level.

3.2.1 Architectural Approach

As outlined in Section 1, we leverage the MAPE loop to achieve runtime self-adaptation capabilities. The following architectural description references system components using terminology from the Autonomic Computing (AC) context. Due to space restrictions, only the most important terms are defined in the following. For detailed coverage of architectural aspects of AC related systems, the please refer to [24], [29].

For the discussion of the system architecture, the most important AC related terms are the *MAPE* loop, *Sensors*, and *Effectors*. The Monitor part of the MAPE loop is responsible for collection and aggregation of details about a service composition, whereas the Analyze part is responsible for correlating and analyzing the symptoms provided by the Monitor part (i.e., *when* does a service composition need to be adapted). The Plan and Execute parts are then responsible for planning actions to achieve goals (i.e., *what* needs to be adapted in the service composition) and applying these actions (i.e., *how* a service composition needs to be adapted). Finally, Sensors and Effectors are used to obtain data about the service composition, and to perform adaptive operations on a service composition.

The system we propose introduces an *Interception and Adaptation Layer* (IAL), as displayed in Fig. 3. With regard to AC, it resembles the responsibilities of Sensors and Effectors. The IAL is a generic tool enabling the inspection and adaptation of message exchange. This message exchange is triggered by an arbitrary *Composition Runtime*, which relies on a *Composition Processor* to instantiate business processes defined in a composition language. The *Messaging Layer* is responsible for the creation and processing of inbound and outbound messages. A *Composition Runtime Adapter* is used to encapsulate all implementation specific details of the orchestrated Composition Runtime. It leverages the IAL and provides a generic message context to other components, such as the Selector, Monitor, and Transformer components.

Collecting and processing QoS information for the services under consideration, as well as persisting the information in a *Datastore*, lies in the responsibility of the *Monitor* component. On each service invocation triggered by the underlying business process, the Monitor creates an *operation invocation event*. Such events hold information about the execution time, a timestamp, and a success indicator. The computation of the supported nondeterministic QoS attributes (refer to Table 1) is based upon this data. The Monitor also stores service metadata extracted from the message context, such as endpoint addresses, to reflect all services involved in process execution in the Datastore. Therefore, it effectively handles the Monitor and Analyze parts of the MAPE loop.

As previously described, the *Selector* component is responsible for determining an appropriate alternative service for an original service. The decision finding is based on certain QoS attributes that serve as input for a selection strategy defined in VieDASSL. This implies that alternatives for an original service need to be stored, together with the related QoS data. In combination with the Transformer component (described below), the Selector covers the Plan and Execute parts of the MAPE loop.

In case the original service and the related alternative services do not adhere to the same interface, this interface

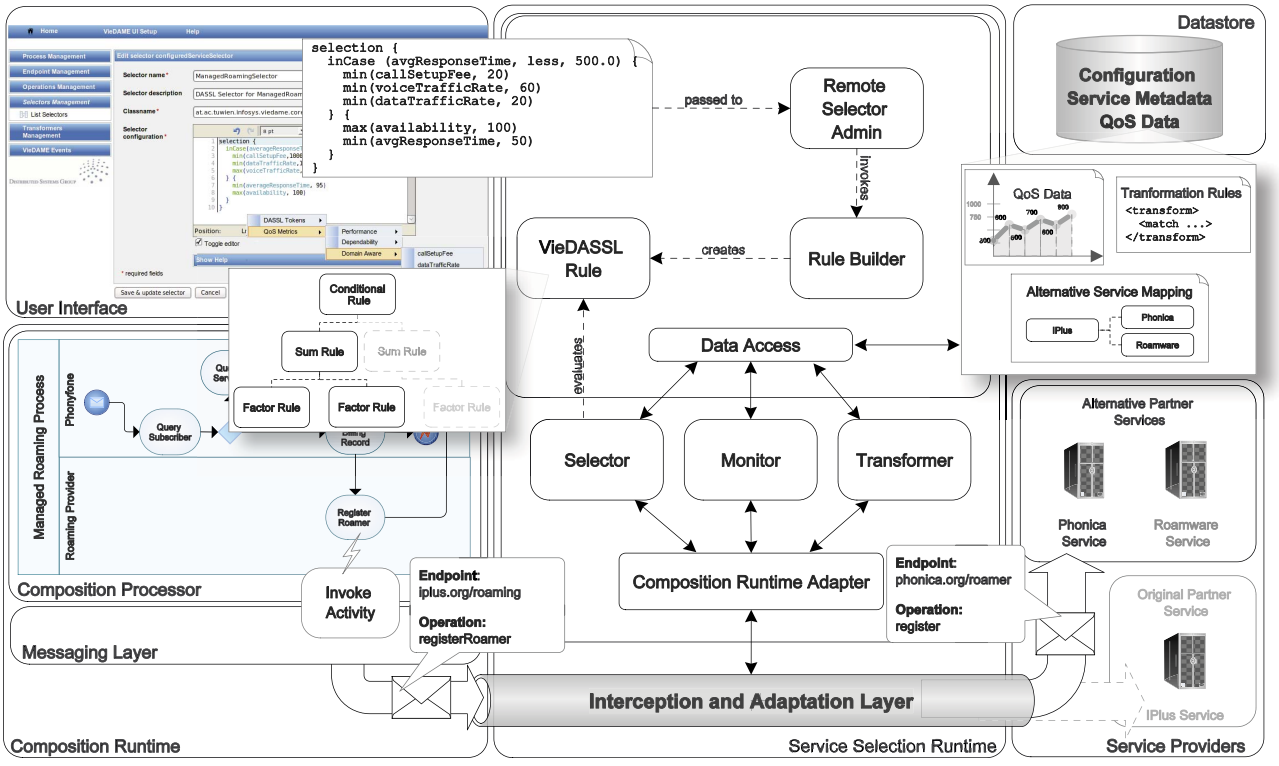

Fig. 3. Architectural approach of a domain-specific service selection system.

mismatch needs to be considered and compensated accordingly. The *Transformer* component is required to apply interface mediation to enforce compliance with the service interface as defined in the process. As an example, the IPlus and the Phonica roaming partner services provide equal functionality with regard to the ManagedRoaming process. However, this does not necessarily mean that both services provide the same interface. In particular, the request messages expected by the Phonica service could differ in their contents from the messages expected by the IPlus service. The same is true for response messages provided by these partner services. Therefore, to use the Phonica roaming service instead of the service provided by IPlus, the Transformer applies certain *transformation rules* for both outgoing and incoming messages. Such an approach loosens up the constraints for alternative services because it only requires *semantical* equivalence, not necessarily *syntactical* equivalence. For our illustrating example, this means that newly emerging roaming partners can be contracted by Phonyfone without the need to adapt the MR process to deviant interfaces provided by these partners.

The Monitor, Selector, and Transformer components leverage a *Data Access* layer to store and retrieve data persisted in the Datastore, such as the QoS information associated with a partner service, the available alternative services for a particular service, or the transformation rules for interface mediation.

When considering the service selection rules defined in VieDASSL, the resulting *VieDASSL Rule* is a tree of rules, with its root rule assigned to the Selector component. The Selector invokes this top level rule for each alternative service, resulting in the root rule recursively invoking all its child rules to evaluate the score of each relevant service. The (re)creation of such a VieDASSL Rule lies in the responsibility of the *Rule Builder*, which parses and verifies selection strategy definitions.

The selection strategy definitions that serve as input for the Rule Builder are passed in by the *Remote Admin*. The Remote Admin decouples the Service Selection Runtime from the *User Interface*. It enables the user to perform service management tasks such as defining the service selection strategies, preparing services for runtime replacement by marking them replaceable, importing alternative services, displaying and comparing QoS data. The User Interface provides an authoring system that supports the domain expert with syntax highlighting and statement completion. Finally, a QoS model browser displays available QoSSymbols and supports the domain expert in creating additional QoSSymbols as needed.

3.2.2 Conceptual Approach

Applying our domain-specific service selection runtime to the ManagedRoaming process enables users 1) to monitor the QoS of the partner services and 2) to define a selection strategy for the roaming partner service. The domain expert does not have to deal with a complex setup to access QoS data of the services involved in process execution. Instead, she can concentrate on the analysis of this information. To leverage the service selection features for the Roaming partner service, the domain expert selects the related service from the list of services captured by the Monitor and marks it replaceable. Only services that are marked replaceable are considered to be replaced by alternative services. What follows is the import and assignment of additional services, e.g., the *Phonica* service. Then, a selection strategy needs to be defined. The strategy shown in Listing 4 provides a

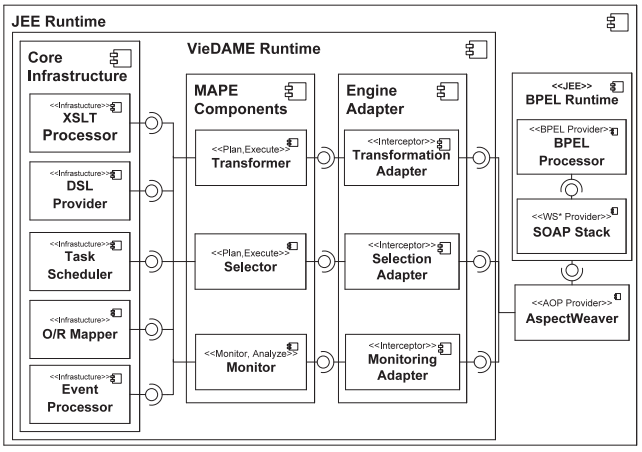

Fig. 4. VieDAME system architecture.

good1 starting point, as it tries to maximize performance and dependability as well as takes cost efficiency into account. Clearly, the considered domain-specific attributes need to be added to the QoS model by defining them in the User Interface. This is done by creating a QoSSymbol for the attribute (e.g., CallSetupFee, cf., QoSSymbol in Listing 1) and defining a value range. The new QoS attribute is immediately available in the service selection runtime. Its values for relevant service operations, e.g., `registerRoamer`, can be set in the User Interface, and the QoSSymbol can be used in service selection strategies. Finally, transformation rules need to be specified since the interface of the originally used roaming partner service (IPlus) does not match the Phonica service interface.

3.3 Implementation

The previous section discussed the design of the domain-specific service selection runtime. In this section, we briefly present a Java-based prototype implementation for BPEL, and highlight important technical details.

Fig. 4 outlines the overall system architecture of the *Vienna Dynamic Adaptation and Monitoring Environment* (VieDAME) using a UML component diagram. VieDAME provides the basis for the implementation discussed hereinafter. In VieDAME, supporting a vendor's BPEL implementation implies providing an *Engine Adapter* for the specific product. Currently, ActiveBPEL [30] is fully supported and support for Apache ODE [31] is under development. The experiments presented in Section 4 are based on ActiveBPEL 3.0.

Conceptually, the *BPEL Runtime* and the *VieDAME Runtime* are deployed together in a *JEE Runtime* such as JBoss Application Server [32]. After the deployment of a BPEL process definition, the BPEL Runtime creates a process instance, which is executed by the *BPEL Processor*. A *SOAP Stack*, such as Apache Axis, is used by the BPEL processor to handle `<invoke>` activities. This type of activity triggers communication with external services.

In VieDAME, many components leverage *Aspect Oriented Programming* (AOP) to provide their functionality. As an example, an Engine Adapter is comprised of a *Monitoring Adapter*, a *Selection Adapter*, and a *Transformation Adapter*. These components are implemented as *Interceptor*s, woven into the base system, i.e., the BPEL Runtime, by the *Aspect Weaver*. This allows us to transparently adapt and extend the message flow in an arbitrary way, without explicit changes in the base system. Although the code base of the BPEL Runtime could be extended by traditional means, i.e., by using subclassing, changes in the base system would always imply changes in the VieDAME components, e.g., the Engine Adapter. By using AOP, we minimize the coupling between our system and the underlying composition runtime. Moreover, AOP allows for selectively enabling and disabling certain features of the VieDAME Runtime. Therefore, in case certain environments require monitoring features only, it is possible to completely disable the Selector and Transformer components.

Previously, we described how the *Monitor* component analyzes the message context and checks whether a representation of the related process, service, and operation was already captured and persisted. For the implementation of such a Monitor component, we decided to distinguish between *real time* and *historical* monitoring data. Historical data, which are primarily used for visualization in the user interface, are stored in a relational database (DB). Details about the *write-behind* semantics for historical data, which were implemented to decrease database load, can be obtained from [23]. To handle (near) real-time information, we incorporated a *Complex Event Processing* (CEP) [33] engine. In VieDAME, CEP is provided by the Open Source project Esper [34], which is comparable to commercial counterparts, both in terms of language expressiveness and performance [35]. From a technical point of view, the Monitor component emits operation invocation *events* that are considered by the Esper runtime. Besides the low processing times Esper can provide, another advantage lies in the ability to define *views* on the continuous stream of events that are emitted by the Monitor. This allows us to restrict the time window that is considered for the calculation of certain QoS data. As an example, the following statement selects the average response time of the QuerySubscriber service during the past 5 minutes:

Listing 5. EPL Example

```
1 select
2  avg(executionTime)
3 from
4  OperationInvocationEvent.win:time(5 min)
5 where
6  operation.name = "querySubscriber"
```

Please note that the statement above is not complete and is listed for demonstration purposes only. As Listing 5 highlights, Esper features a SQL like *Event Processing Language* (EPL), which is capable of expressing temporal and logical context such as the average response time over the past 5 minutes in the given example. The integration of real time and historical data is provided by the Esper's SQL support. It uses the Java Database Connectivity (JDBC) standard and allows us to join real-time event streams against JDBC data sources. Moreover, these JDBC data sources are fed with the real-time data in the write-behind fashion mentioned above. This turns the real-time data into historical data after a certain amount of time.

Regarding the *Selector* component, our implementation makes extensive use of the scripting support [36] of current Java Virtual Machines (JVM). In particular, we leverage the `ExpandoMetaClass` feature of Groovy [37], as well as the Builder Pattern [38], to implement the required runtime adaptiveness. The *Adaptive QoS Model* is persisted in the DB and can be accessed and modified by the object/relational mapping (ORM) support provided by VieDAME's Infrastructure Core described below. The interface mediation that the *Transformer* component provides is built upon XSLT transformations. The related stylesheets are also stored in and retrieved from the VieDAME DB.

The *MAPE Components* discussed so far all rely on the *Infrastructure Core* of VieDAME. As its name implies, it provides infrastructural services like ORM (Hibernate [39]), task scheduling (Quartz [40]), interfaces for remote administration as well as the dynamic language support. The Spring Framework [41] is used to configure and wire all components.

From an end-user perspective, all interaction with VieDAME is done via the VieDAME User Interface (UI). The JBoss Seam [42] based user interface covers all functionalities required to leverage VieDAME as a domain-specific service selection runtime. As an example, the actual Selector definition is supported by an integrated editor based on EditArea [43], which features syntax highlighting, auto completion for the VieDASSL syntax, and a special context menu that serves as a QoS model browser.

4 Evaluation

The following experiments are based on a BPEL implementation of the ManagedRoaming sample process from Section 2. We present the details of the evaluation scenario setup, followed by the results of our performance tests, and finally, we discuss the findings of the evaluation.

4.1 Scenario Setup and Results

To provide a high level of isolation for the performance measurements, three different hosts took part in the setup. In particular, VieDAME was deployed on a dedicated host so as not to bias the results by the processing time the ManagedRoaming Web services and the LoadRunner setup needed. 1) An Intel Core2Duo clocked at 2.66 GHz was used to host the ActiveBPEL BPEL engine and the VieDAME instance, featuring 4 GB of main memory. 2) The orchestrated Web services that took part in the BPEL process were installed on a 2.6 GHz AMD X2 5050e-based machine, equipped with 2 GB of RAM. 3) Finally, the LoadRunner console and load generator were hosted on a 2.4 GHz Intel Core2Duo laptop, backed by 2 GB memory. Gigabit Ethernet was used to connect the three machines.

Except for the Windows Vista Enterprise-based laptop that hosted the LoadRunner installation, all other components were running Linux 2.6.30. JBoss Application Server 4.2.3 GA was used as the JEE container for ActiveBPEL and VieDAME. The database was installed on the VieDAME host. Postgres 8.1 was used.

TABLE 5
Partner Service QoS Attributes

Web Service	Delay	Factor	Failure Rate
QuerySubscriber	250ms	90	0%
QueryOrder	180ms	90	0%
CreateSubscripton	300ms	90	0%
CreateBillingRecord	500ms	90	0%

The Web service framework JBossWS [44] was used to create JAX-WS compliant implementations for the partner services mentioned in Section 2.1. To achieve realistic numbers regarding the throughput and response times of the participating services, every Web service supports an additional `setQos()` operation. This operation allows us to dynamically set certain attributes, such as processing delay or error probability, together with a random factor to simulate service quality. For the exact parameter values assigned to the Phonyfone-hosted Web service instances, please refer to Table 5. The given delay is an upper bound; the real delay is calculated using the formula $delay * \frac{random.nextInt(factor)}{100}$.

The commercial, industry-leading load testing suite HP LoadRunner [45] was the tool of choice to create a realistic high load scenario. Three different test schedules were created, featuring 25, 50, and 100 concurrent virtual users. Due to length restrictions, only the results of the 50 and 100 users scenarios are shown below. A test duration of 50 minutes, with an additional ramp up time for the virtual users, was set. Moreover, a randomized pacing time between 1,000 and 2,000 ms was applied between consecutive requests. The input MSISDNs for the process instantiating `register` requests were taken randomly from a predefined list.

Similarly to our previous test setups (refer to [23]), the tests were run in four different configurations. In stage 1, s default ActiveBPEL instance was stressed, with the VieDAME extension completely disabled; in stage 2, VieDAME was stressed in a monitoring-only configuration to highlight the performance penalty introduced by the monitoring capabilities (cf., Section 3.2.1). The monitoring-only configuration was created by disabling the selection and transformation adapters (cf., Section 3.3); in stage 3, a VieDAME enabled ActiveBPEL instance featuring an response time Static Selector that was tested; and finally, for stage 4 the VieDASSL Selector shown in Listing 6 was set up. This conditional selector tries to minimize the CallSetupFee, VoiceTrafficRate, and DataTrafficRate in case the average response time of the related operation is lower than 400 ms over the last 5 minutes. If the response time exceeds the 400 ms threshold, the selector picks the service that provides the lowest 5 minute average response time as well as the highest availability of all alternative services. For the VieDAME enabled test stages, an extended Transformer was configured to compensate the interface mismatch between the IPlus roamer registration service and the Phonica roamer registration service. This means that the full set of VieDAME features, including Monitoring, Service Selection, and Interface Transformation, was stressed.

TABLE 6
Roaming Partner QoS Settings

	IPlus	Phonica
Delay (ms)	150/1500/2500	250/1500/2500
FailureRate (%)	0/0/0	0/0/0
CallSetupFee (€)	0.05	0.03
VoiceTrafficRate (€)	0.32	0.32
DataTrafficRate (€)	0.9	0.78

Listing 6. VieDASSL Selector for ManagedRoaming

```
1 selection{
2     inCase(responseTimeMs5Min, less,
      400.0){
3         min(callSetupFeeEur, 0.3)
4         min(voiceTrafficRateEur, 0.4)
5         max(dataTrafficRateEur, 0.3)
6     } {
7         min(responseTime5MinMs, 0.49)
8         max(availabilityPercent, 0.51)
9     }
10 }
```

During the load test, we simulated a *service outage* to highlight the real-performance gain in process execution time when replacing a slow service with a reasonable performing alternative. It is assumed that the service invoked by the RegisterRoamerAbroad activity, e.g., IPlus, provides two service instances which are hidden behind a load balancer. This is a common approach to increase service performance and dependability in real-world applications. For our experiment, we assume that one of the two service instances behind the load balancer fails. Therefore, the load balancer excludes the faulty service instance from its service queue. This means that the remaining instance must service all incoming requests, resulting in higher response times for the client, i.e., the BPEL process.

SoapUI [46] was used to schedule the outage by creating a test case that includes only two test steps: 1) A delay test step that introduces a 25 minute wait state, and 2) the actual invocation of the `setQos()` operation to increase the processing delay of the RegisterRoamerAbroad partner service. The default values, i.e., before the service outage, for the `setQos()` operation, as well as the values during the outage simulation for 50 and 100 user scenarios are listed in Table 6. The delay factor was set to `90` in all cases. Furthermore, the table also lists the values of the rates and fees that are considered for service selection by the stage 4 VieDASSL Selector.

Based on the values from Table 6, the Static Selector used in the stage 3 setup favors the default IPlus service due to slightly better response times. The VieDASSL Selector of stage 4 chooses the Phonica service since it does not violate the configured response time threshold and offers better domain-specific QoS.

Of course, a process executed in the stage 1 and stage 2 scenarios can only invoke the default (IPlus) service. Due to these preferences, we increased the processing time of the IPlus service in the stage 1 and stage 2 runs during the service outage simulation to 1,500 ms (50 users scenario) and 2,500 ms (100 users scenario), while for the stage 4 run, we increased the processing time of the Phonica service to 1,500 and 2,500 ms for the 50 and 100 users scenarios, respectively.

4.2 Discussion and Findings

Fig. 5 illustrates the results of the load tests. The first column shows the results for the 50 users scenarios, while the second column presents the numbers for the 100 users scenarios. The two upper diagrams display the average transaction response time (since the LoadRunner console checks the validity of the response, this can also be referred to as the *RoundTripTime* [17]). The diagrams in the middle highlight the transactions per second, i.e., the number of completed ManagedRoaming process executions per second. Finally, the lower two diagrams show the average CPU usage during the test runs. Table 7 summarizes the numbers for the 100 users scenario, as well as the normalized costs incurred by the roaming partner service invocations that have to be paid by Phonyfone.

The results before the service outage at 25:00 show a processing overhead introduced by VieDAME. While the results of the stage 1 and 2 test runs show that the impact of the Monitor component on overall system performance is vanishingly small, the performance penalty of the stage 3 and 4 scenarios is not negligible. Results from these stages show a lower number of transactions per second (TPS), a higher average response time (AVGRT), and a higher CPU usage. While the slight performance penalty in both TPS and AVGRT seems to grow in favor for the standard ActiveBPEL instance in scenarios with a higher number of concurrent users, the CPU usage remains almost constant. Moreover, the results for the stage 3 setup and the stage 4 setup are almost identical, meaning that the components involved in the domain-specific service selection do not introduce any additional processing overhead. During the service outage (from 25:00 on), the results clearly show the performance optimizing characteristics of the VieDAME setups. The high response time of the IPlus roaming partner results in higher AVGRT and a lower number of TPS in the standard ActiveBPEL environment for both the 50 and 100 users scenarios during the remainder of the test runs. In contrast, both the stage 3 VieDAME tests as well as the stage 4 results show only a short period of lower process performance caused by the outage. For both VieDAME scenarios, overall process performance is brought back to where it was for the rest of the load test. However, the stage 4-based process executions have two advantages over the stage 3 VieDAME installation. First, the domain-specific selector minimizes the domain-specific QoS aspects associated with the roaming partners, while tracking both the response time and availability of the alternative roaming service providers. Thus, the domain-specific selector effectively saves costs while not compromising process performance in terms of availability and execution time. Second, when comparing the time needed for the selector to switch over to a better performing alternative, the time-window-based QoS features that were enabled in the stage 4 setup can considerably minimize the critical time frame where

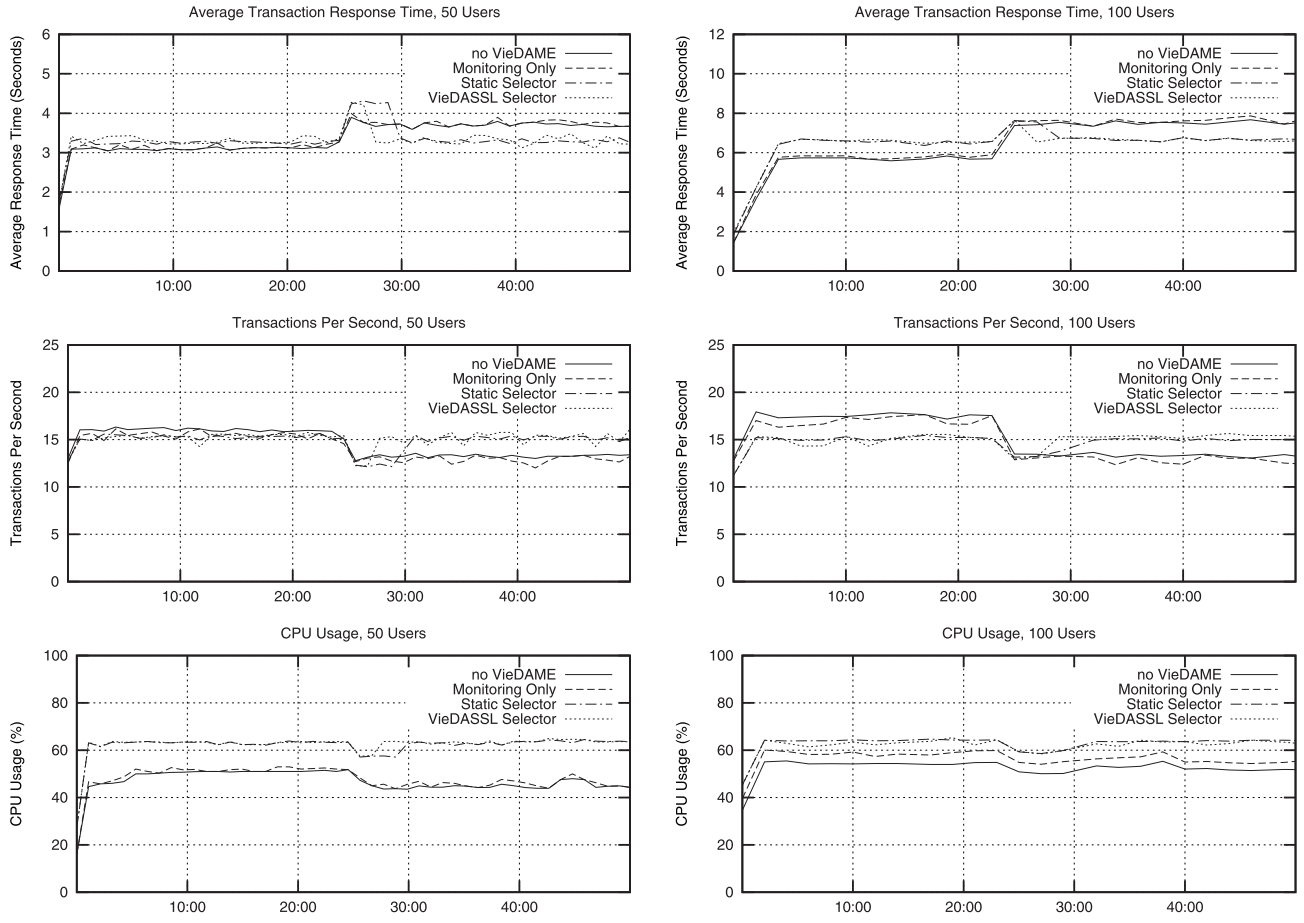

Fig. 5. Response times and transactions per second during VieDAME Loadtest.

process performance is affected by the service outage. The domain expert can also adjust the selector thresholds and observation periods, e.g., last n invocations instead of a fixed time period, in case of changed requirements.

Besides performance and dependability advantages, other main benefits of our approach have to be considered with regard to the stakeholders. From a system engineer's perspective, who leverages VieDAME as a framework, choosing the message level as the layer for interception and adaptation of service interaction results in broad compatibility with existing and future composition platforms. This design decision also minimizes potential side effects compared to operating on other abstraction levels, such as the in-memory process representation or the schema level. Additionally, the clean component oriented architecture of our approach can be easily extended to cover additional requirements.

Considering the domain expert perspective, our approach provides an intuitive and goal oriented language to describe domain-specific service selection requirements. The main advantage of leveraging VieDASSL is the achievement of a clear separation of concerns in large enterprises. Domain experts should not be required to learn a programming language to define service selection strategies. In fact, they should focus on the business-related aspects of service selection. Moreover, the runtime adaptability features of VieDASSL and the underlying QoS model implementation address very short-term market requirements. These requirements might be imposed by a regulator or law enforcement, but also by customer feedback or a competitor's new offering. Minimizing time to market (TTM) is crucial, not only for telco enterprises as shown in this paper, but for the majority of other businesses. Our approach can be leveraged to effectively minimize TTM, resulting in a key advantage in today's highly competitive businesses.

TABLE 7
100 Users Scenario Results Summary

Performance Metric	Stage 1	Stage 2	Stage 3	Stage 4
Transactions per second	13.98	13.52	15.02	15.13
Average response time	6.18	6.19	5.82	5.76
CPU usage	50.77	53.59	62.55	61.74
CallSetupFee	2097	2097	2097	1677.6

5 Related Work

There is a recognizable trend that SOAs are becoming mainstream and the underlying infrastructures, such as composition middleware, cloud platforms, and services, increasingly adopt principles of self-adaptive software systems [10]. The benefits are software applications that are better manageable and more robust toward requirements changes during their lifetimes, which in turn reduces the operational costs. Software applications adopting self-adaptation mechanisms typically implement an adaptation

TABLE 8
Comparison of Existing Approaches

MAPE Phase	Criteria		Baresi et al. [19], [47]	Charfi and Mezzini [20]	Erradi et al. [21]	Zeng et al. [16]	Moscinat et al. [22]	Michlmayr et al. [48]	Our Approach
M	QoS Model	Deterministic Attr.	+	∼	+	+	+	+	+
		Non-deterministic Attr.	+	∼	–	+	∼	+	+
		Design-time Ext.	+	+	∼	–	+	+	+
		Runtime Ext.	–	–	–	–	–	–	+
M,A	QoS Monitoring	Real-time Data	+	–	+	∼	∼	+	+
		Temporal Aspects	–	–	–	–	–	–	+
P	Selector Specification	Language-based	–	+	+	–	∼	+	+
		Logic-based	+	–	–	–	–	–	–
E	Selector Adaptation	Design-time	+	+	+	–	+	+	+
		Runtime	–	–	–	–	–	–	+
E	Service Mediation	Message-based	+	+	+	-	-	+	+
		Mediation Flows	∼	+	∼	-	-	-	+

– not supported ∼ partly supported + fully supported

loop, consisting of several process, sensors, and effectors [12]. Variants of this adaptation loop are the MAPE-K loop from the Autonomic Computing space [24], [29], as introduced in Section 3.2, or adaptation management, introduced by Oreizy et al. [10].

In the following, we focus on related approaches from the Service-Oriented Computing [4] context that implement parts or the complete adaptation loop and compare it to our work. We categorize them according to five criteria, as illustrated in Table 8. For each criterion, we additionally show the specific MAPE phase to illustrate where the proposed approach fits into the adaptation loop. We use the abbreviations M, A, P, E as described in Section 3.2. The *QoS Model* criteria identifies whether deterministic and non-deterministic QoS attributes are supported. The remaining subcriteria classify whether the QoS model implementation is extensible at design and/or runtime. *QoS Monitoring* identifies whether an approach supports monitoring of real-time QoS data within a composition and whether temporal aspects are considered (e.g., the average response time over the last 5 minutes or 100 measurements). The next two criteria address the *Selector Specification and Adaptation*. The classification of the selector specification roughly follows the taxonomy from Delgado et al. [49]. We leverage language-based approaches (e.g., based on an imperative, object-oriented, or functional language) or approaches based on a logic. The adaptation subcriteria define whether a selector can be dynamically adapted at runtime or is just simply adaptable at design-time, thus requiring a redeploy and reload of the process. Finally, *Service Mediation* describes whether the approach is able to deal with service interface heterogeneities. We distinguish between 1) simple message-based mediation, where the original message can be transformed to the message expected by the target service (e.g., using XLST), and 2) mediation flows, where the mediation can have multiple steps and even call other services as part of the mediation execution (similar to what is supported by an Enterprise Service Bus [50]).

The approach presented by Baresi et al. [19] and Baresi and Guinea [47] shares many commonalities with our approach. They propose the Web Service Recovery Language (WSREL), which allows us to define complex recovery strategies that include retry and rebind of Web services. These recovery strategies can be seen as some form of selectors that can be defined at design-time; however, they are not adaptable at runtime. In [51], the same authors focus on service selection approaches where self-healing of service compositions is the primary concern.

Charfi and Mezini [20] propose an AOP extension for BPEL by defining a simple pointcut language in XML. Although they focus on different QoS attributes—specified by the Web service stack, e.g., transactions, reliable messaging, and security—their AOP approach can be used to implement selectors which can be adapted at design-time only.

Erradi et al. [21] propose an adaptive middleware with a policy-aware selection mechanism. They have a fixed set of QoS attributes that they can measure and use in the adaptation policies. These policies can only be adapted at design time.

Zeng et al. [16] proposed a foundational approach for QoS-aware service selection and optimization of composite services. Their approach uses a small set of given QoS attributes that can be used as part of the optimization. The selection of a specific service is determined by the optimization algorithm and cannot be adapted without reexecuting the optimization. Conceptually similar approaches have been proposed in [52], [53], [54], [55], [56]. However, most approaches use different service selection techniques to optimize the overall QoS of a composition (e.g., skyline computation [56]).

Moscinat and Binder [22] focus on a transparent runtime adaptability of BPEL processes based on a fixed set of QoS attributes. Their main approach is to dynamically adapt the services in a BPEL process (both stateless and stateful) by augmenting the BPEL code with service selection code at deployment time. This implies that they cannot dynamically adapt the service selection logic.

Michlmayr et al. [48] present VRESCo, a QoS-aware middleware that addresses adaptability of service-oriented applications. They propose several infrastructure components and services including dynamic binding and invocation, querying and composition that are based on a unified metadata model. Service selection can be specified by formulating service queries that return a rebinding proxy which implements different QoS-based rebinding strategies

(e.g., on demand, on invocation, etc.). The service selection logic is predefined by choosing among a range of selectors, depending on the application scenario and its need for continuous rebinding.

Table 8 clearly shows that none of the related approaches supports runtime adaptation of the QoS model and selector implementations. This enables us to add, remove, or change attributes within the QoS model implementation and use the newly added QoS attributes in a selector logic without any downtime of the overall infrastructure. This is crucial, especially for high-availability environments such as the telecommunications example from Section 2.

6 Conclusion

This paper discussed our approach for domain-specific service selection for composite services. An authentic business process was used to guide through the explanations and to show the practical relevance of our system. Our prototype implements the concepts we presented for BPEL compliant systems. However, operating on the message level enables the application of our approach for a vast majority of current and future composition platforms. By using AOP as the technology for extending a given composition runtime, our system is extensively unobtrusive to orchestrated composition runtimes.

We introduced a DSL tailored for the task of specifying selection strategies based on an adaptive QoS model implementation. Both the selection strategies and the underlying QoS model implementation support runtime adaptability. These system properties maximize responsiveness to evolving selection strategy requirements as well as minimize the need for system downtimes. The proposed system was tested and evaluated using a case study from the telecommunication domain. The evaluation showed a certain performance penalty, given the assumptions for the system as outlined in Section 1. However, this penalty can be neglected when considering the real performance gain that the adaptive features of our service selection system can render. We also believe, although not validated, that the system performs well when the number of alternative services is higher (e.g., around 50) because the underlying service selection code is not computationally expensive.

For our future work, we plan to evaluate the performance of our system with different BPEL runtimes. This will also lead to additional engine adapters that will be made available. Moreover, we will examine how service composition approaches other than BPEL can be integrated with our prototype. Another area of improvement is support for different units of measurement for VieDASSL and our adaptive QoS model implementation. This feature will further increase the number of applicable alternative services as it enables different units for cost and time. We will continue to investigate how the selection postprocessing approach helps to solve specific problem scenarios where choosing the top scored service does not lead to optimal results. Our research will also focus on the event-driven monitoring capabilities that are incorporated into our prototype. Research will cover how temporal and logical context can be reflected. This is a prerequisite to handling complex monitoring requirements such as detecting fraudulent request patterns based on the real-time analysis of invocation traces.

Acknowledgments

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement 215483 (S-Cube). The authors would like to express their gratitude to Markus Wilthaner for the prototype implementation of VieDASSL. Additionally, they would like to thank the anonymous reviewers for their fruitful comments.

References

[1] I.-L. Yen, H. Ma, F.B. Bastani, and H. Mei, "QoS-Reconfigurable Web Services and Compositions for High-Assurance Systems," *Computer,* vol. 41, no. 8, pp. 48-55, 2008.

[2] C. Huemer, P. Liegl, R. Schuster, H. Werthner, and M. Zapletal, "Inter-Organizational Systems: From Business Values over Business Processes to Deployment," *Proc. IEEE Second Int'l Conf. Digital Ecosystems and Technologies,* pp. 294-299, 2008.

[3] R. Altman, "SOA Overview and Guide to SOA Research," Gartner Research Report (ID Number: G00201650), July 2010.

[4] M.P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-Oriented Computing: State of the Art and Research Challenges," *Computer,* vol. 40, no. 11, pp. 38-45, Nov. 2007.

[5] T. Erl, *SOA Principles of Service Design.* Prentice Hall PTR, 2007.

[6] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D.F. Ferguson, *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More.* Prentice Hall PTR, 2005.

[7] A. Michlmayr, F. Rosenberg, C. Platzer, M. Treiber, and S. Dustdar, "Towards Recovering the Broken SOA Triangle—A Software Engineering Perspective," *Proc. Second Int'l Workshop Service Oriented Software Eng.,* pp. 22-28, Sept. 2007.

[8] E. Di Nitto, C. Ghezzi, A. Metzger, M. Papazoglou, and K. Pohl, "A Journey to Highly Dynamic, Self-Adaptive Service-Based Applications," *Automated Software Eng.,* vol. 15, pp. 313-341, http://dx.doi.org/10.1007/s10515-008-0032-x, 2008.

[9] Z. Zheng and M.R. Lyu, "A QoS-Aware Fault Tolerant Middleware for Dependable Service Composition," *Proc. IEEE/IFIP Int'l Conf. Dependable Systems Networks,* pp. 239-248, July 2009.

[10] P. Oreizy, M.M. Gorlick, R.N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D.S. Rosenblum, and A.L. Wolf, "An Architecture-Based Approach to Self-Adaptive Software," *IEEE Intelligent Systems and Their Applications,* vol. 14, no. 3, pp. 54-62, May/June 1999.

[11] A. Avizienis, J.-C. Laprie, B. Randell, and C.E. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE Trans. Dependable and Secure Computing,* vol. 1, no. 1, pp. 11-33, Jan.-Mar. 2004.

[12] M. Salehie and L. Tahvildari, "Self-Adaptive Software: Landscape and Research Challenges," *ACM Trans. Autonomous and Adaptive Systems,* vol. 4, no. 2, pp. 1-42, 2009.

[13] D.A. Menasce, "QoS Issues in Web Services," *IEEE Internet Computing,* vol. 6, no. 6, pp. 72-75, Nov./Dec. 2002.

[14] A. Mani and A. Nagarajan, "Understanding Quality of Service for Web Services," http://www.ibm.com/developerworks/library/ws-quality.html, Jan. 2002.

[15] M.C. Jaeger, G. Rojec-Goldmann, and G. Mühl, "QoS Aggregation for Service Composition Using Workflow Patterns," *Proc. Eighth Int'l Enterprise Distributed Object Computing Conf.,* pp. 149-159, Sept. 2004.

[16] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-Aware Middleware for Web Services Composition," *IEEE Trans. Software Eng.,* vol. 30, no. 5, pp. 311-327, May 2004.

[17] F. Rosenberg, C. Platzer, and S. Dustdar, "Bootstrapping Performance and Dependability Attributes of Web Services," *Proc. IEEE Int'l Conf. Web Services,* 2006.

[18] M.C. Jaeger, G. Mühl, and S. Golze, "QoS-Aware Composition of Web Services: An Evaluation of Selection Algorithms," *Proc. On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE,* pp. 646-661, Nov. 2005.

[19] L. Baresi, S. Guinea, and P. Plebani, "Policies and Aspects for the Supervision of BPEL Processes," *Proc. 19th Int'l Conf. Advanced Information Systems Eng.,* pp. 340-354, 2007.

[20] A. Charfi and M. Mezini, "AO4BPEL: An Aspect-Oriented Extension to BPEL," *World Wide Web,* vol. 10, no. 3, pp. 309-344, 2007.
[21] A. Erradi, P. Maheshwari, and V. Tosic, "Policy-Driven Middleware for Self-Adaptation of Web Services Compositions," *Proc. ACM/IFIP/USENIX Int'l Conf. Middleware,* pp. 62-80, 2006.
[22] A. Mosincat and W. Binder, "Transparent Runtime Adaptability for BPEL Processes," *Proc. Sixth Int'l Conf. Service-Oriented Computing,* pp. 241-255, 2008.
[23] O. Moser, F. Rosenberg, and S. Dustdar, "Non-Intrusive Monitoring and Service Adaptation for WS-BPEL," *Proc. 17th Int'l Conf. World Wide Web,* pp. 815-824, 2008.
[24] J. Kephart and D. Chess, "The Vision of Autonomic Computing," *Computer,* vol. 36, no. 1, pp. 41-50, Jan. 2003.
[25] *IR.50.4.0-2G/2.5G/3G Roaming,* GSM World, 2009, http://www.gsmworld.com/documents/IR50_4_0.pdf, 2009.
[26] *Business Process Modeling Notation (BPMN) Specification, Version 1.0,* Object Management Group—Business Process Management Initiative, http://www.bpmn.org/, 2006.
[27] *VieDAME Project Website,* Oliver Moser, http://viedame.omoser.com, 2010.
[28] C.-L. Hwang and K. Yoon, *Multiple Attribute Decision Making Methods and Applications.* Springer-Verlag, 1981.
[29] *An Architectural Blueprint for Autonomic Computing,* IBM Corporation, http://www-01.ibm.com/software/tivoli/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf, 2006.
[30] *ActiveBPEL Engine,* Active Endpoints, http://www.active-endpoints.com/, 2007.
[31] *Apache ODE,* Apache Software Foundation, http://ode.apache.org/, 2011.
[32] *JBoss Application Server,* Red Hat, http://www.jboss.org, 2007.
[33] D.C. Luckham, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems.* Addison-Wesley Longman Publishing Co., Inc., 2001.
[34] *Esper,* EsperTech, http://esper.codehaus.org, 2009.
[35] *Esper Performance Wiki,* EsperTech, http://docs.codehaus.org/display/ESPER/Esper+performance, 2009.
[36] *JSR 223: Scripting for the Java Platform,* SUN Microsystems, http://www.jcp.org/en/jsr/detail?id=223, 2009.
[37] *ExpandoMetaClass Domain Specific Language with Groovy,* Codehaus, http://groovy.codehaus.org/ExpandoMetaClass+Domain-Specific+Language, 2009.
[38] E. Gamma, R. Helm, R. Johnson, and J. Vlissidesi, *Design Patterns, Elements of Reusable Object-Oriented Software.* Addison-Wesley Longman Publishing Co., Inc., 1994.
[39] *Hiberante ORM,* Red Hat, http://www.hibernate.org, 2007.
[40] *Quartz,* OpenSymphony, http://www.opensymphony.com/quartz/, 2007.
[41] *Spring Framework,* Interface21, http://www.springframework.org, 2007.
[42] *JBoss Seam,* Red Hat, http://www.jboss.org, 2007.
[43] *EditArea JavaScript Source Code ed.,* Christophe Dolivet, http://www.cdolivet.com/index.php?page=editArea, 2009.
[44] *JBossWS,* RedHat, http://www.jboss.org/jbossws/, 2007.
[45] *LoadRunner,* HP, https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-11-126-17^8_4000_100__, 2009.
[46] *SoapUI,* Eviware, http://www.soapui.org/, 2009.
[47] L. Baresi and S. Guinea, "A Dynamic and Reactive Approach to the Supervision of BPEL Processes," *Proc. First India Software Eng. Conf.,* pp. 39-48, 2008.
[48] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar, "End-to-End Support for QoS-Aware Service Selection, Binding and Mediation in VRESCo," *IEEE Trans. Services Computing,* vol. 3, no. 3, pp. 193-205, July-Sept. 2010.
[49] N. Delgado, A.Q. Gates, and S. Roach, "A Taxonomy and Catalog of Runtime Software-Fault Monitoring Tools," *IEEE Trans. Software Eng.,* vol. 30, no. 12, pp. 859-872, Dec. 2004.
[50] *Apache Synapse,* Apache Software Foundation, http://ws.apache.org/synapse/, 2007.
[51] L. Baresi, S. Guinea, and L. Pasquale, "Self-Healing BPEL Processes with Dynamo and the JBoss Rule Engine," *Proc. Int'l Workshop Eng. of Software Services for Pervasive Environments,* pp. 11-20, 2007.
[52] D. Ardagna and B. Pernici, "Adaptive Service Composition in Flexible Processes," *IEEE Trans. Software Eng.,* vol. 33, no. 6, pp. 369-384, June 2007.
[53] F. Rosenberg, P. Celikovic, A. Michlmayr, P. Leitner, and S. Dustdar, "An End-to-End Approach for QoS-Aware Service Composition," *Proc. IEEE 13th Int'l Conf. Enterprise Distributed Object Computing,* 2009.
[54] T. Yu, Y. Zhang, and K.-J. Lin, "Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints," *ACM Trans. Web,* vol. 1, no. 6, pp. 1-26, 2007.
[55] M. Alrifai and T. Risse, "Combining Global Optimization with Local Selection for Efficient QoS-Aware Service Composition," *Proc. 18th Int'l Conf. World Wide Web,* pp. 881-890, Apr. 2009.
[56] M. Alrifai, D. Skoutas, and T. Risse, "Selecting Skyline Services for QoS-Based Web Service Composition," *Proc. 19th Int'l Conf. World Wide Web,* Apr. 2010.

Oliver Moser is currently working toward the PhD degree at the Vienna University of Technology. His research interests include service-based systems and complex event processing. He has more than 8 years industrial experience in designing and implementing large-scale service and provisioning platforms for telecommunication enterprises. He is a student member of the IEEE. More information can be found at http://www.omoser.com.

Florian Rosenberg received the PhD degree in computer science in June 2009 from the Vienna University of Technology. He is a research staff member at the IBM T.J. Watson Research Center in New York. His research interests include service-oriented computing, cloud computing, and software engineering. Currently, he works on virtual image construction and composition in cloud computing to simplify software deployment and configuration. More information can be found at http://www.florianrosenberg.com.

Schahram Dustdar is a full professor of computer science with a focus on Internet technologies heading the Distributed Systems Group, Institute of Information Systems, Vienna University of Technology (TU Wien). From 2004-2010 he was an honorary professor of information systems in the Department of Computing Science at the University of Groningen (RuG), The Netherlands. He is a senior member of the IEEE. He has been an ACM distinguished scientist since 2009. More information can be found at http://www.infosys.tuwien.ac.at/Staff/sd.