# Model-based Management of Web Service Compositions in Service-Oriented Architectures

Christof Momm, Christoph Rathfelder

Software Engineering
FZI Forschungszentrum Informatik
Haid-und-Neu-Straße 10-14
76131 Karlsruhe
{momm, rathfelder}@fzi.de

**Abstract:** Web service compositions (WSC), as part of a service-oriented architecture (SOA), have to be managed to ensure compliance with guaranteed service levels. In this context, a high degree of automation is desired, which can be achieved by applying autonomic computing concepts. This paper particularly focuses the autonomic management of semi-dynamic compositions. Here, for each included service several variants are available that differ with regard to the service level they offer. Given this scenario, we first show how to instrument WSC in order to allow a controlling of the service level through switching the employed service variant. Second, we show how the desired self-manageability can be designed and implemented by means of a WSC manageability infrastructure. The presented approach is based on widely accepted methodologies and standards from the area of application and web service management, in particular the WBEM standards.

## 1 Introduction

Today, companies require IT support that is tightly aligned with their business processes and highly adaptive in case of changes. These requirements can be met by employing a Service-Oriented Architecture (SOA) [vdAtHW03]. In SOA, functionality required for executing business processes is provided by atomic web services (WS) or by web service compositions (WSC). A WSC is designed in a strictly process-oriented way and implements fully automated parts of business processes or even long-running workflows [LRS02].

Each service – composite or atomic - is characterized by the fact that it is operated by a service provider and the terms of use are contractually fixed by means of Service Level Agreements (SLA). Such an SLA may for instance constitute that a WSC has to adhere to a certain response time constraint. While providing the service the provider has to assure the compliance with the corresponding SLA. To this end, the provider has to be able to monitor the actual service levels and be able to react to detected SLA violations as part of his service level management. These management functions should be automated as far as possible [SMS+02]. Only in this way the vision of an "on-demand" provisioning of the offered (composite) web services can be reached [DDK+04].

An automated service level management for WSC can – at least partly - be achieved by applying autonomic computing concepts, as for instance presented in [IBM04]. The managed resources in this context are the WSC. These resources should be equipped with self-management capabilities, which are realized through autonomic managers. More precisely, the autonomic managers implement so-called intelligent control loops, which generally comprise a monitor, analyze, plan and execute function. To implement these functions, the managed resources, in our case the WSC, have to provide an adequate manageability interface allowing for both the monitoring and controlling of the resources. This is enabled though sensors and effectors added to the managed resource, which is also referred to as "instrumentation".

As part of our preliminary work, we already presented the design and implementation of a manageability infrastructure for WSC based on the Web-based Enterprise Management (WBEM) [1] standards [MMRA07, Rat07]. In this way, a standardized manageability interface is offered that allows a fine-grained monitoring of WSC as part of an SLA-driven management. The required monitoring instrumentation is based on sensors added to the WSC.

This paper now focuses on the controlling instrumentation of WSC and the enhancement of a WSC manageability infrastructure by incorporating autonomic management concepts, in the following referred to as self-manageability. In this context, we regard the autonomic managers themselves to be part of the manageability infrastructure. We furthermore assume the WSC to be of semi-dynamic nature [ZK06]. This means that the composition logic itself is static but several variants of the included services are available that differ with regard to the service levels they offer [TP05]. The concretely employed service variants may be selected dynamically during or prior to the execution.

The actual contribution of this paper is twofold. First, we present and discuss different approaches to a controlling instrumentation (i.e. effectors) of WSC. To ensure universal applicability, we focus on BPEL-based WSC. Additionally, the proposed solutions do not rely on any vendor-specific extensions of the employed BPEL engines, i.e. they are platform-independent. To minimize the effort for instrumenting the WSC we show how generative techniques could be used for an automation of this process. Second, we show how self-manageability can be designed and implemented by means of a tailored WSC manageability infrastructure. Here, we leverage the WBEM standards to obtain a flexible solution that may easily be integrated into existing management environments.

The management requirements and proposed solutions are demonstrated by means of a concrete scenario taken from the field of higher education. Before introducing this scenario, we first provide an overview of the relevant related work.

[1] http://www.dmtf.org/standards/wbem/

## 2 Related Work

The controlling and eventually self-management capabilities for WSC may be used within an SLA management infrastructure. In literature, two major solutions for a SLA-based management of WS and WSC have been presented, which rely on an instrumentation of the managed resources and a manageability infrastructure. In [DDK+04], a solution for an automated SLA-driven management on basis of Web Service Level Agreements (WSLA) is presented. However, this solution mainly focuses on monitoring SLA compliance of atomic WS and does neither adequately support the monitoring nor the controlling of WSC. In [DK03], the approach is extended by a WBEM-based monitoring infrastructure. WSC monitoring is now partially supported, but management capabilities are not covered either. In [SMS+02], a competing solution is presented which supports an automated SLA compliance monitoring of atomic WS and WSC. However, the solution represents a very proprietary approach as the manageability interface is not built on standards. Furthermore, the solution is also limited to monitoring capabilities.

Each of the previously introduced approaches supports the flexible negotiation of discrete or continuous service level parameters. This results in a vast variety of offered service variants and implies enormous challenges for the service provider concerning the provisioning and management of the eventually provided service variants. In [TP05], a more pragmatic approach is presented, based on a language for defining web service offerings [TPP03]. This allows the specification of different discrete variants of one WS in terms of service offerings. In contrast to WSLAs, the customer cannot freely negotiate all kinds of service level parameters, but may rather choose the predefined service variant most appropriate for him. A corresponding management infrastructure is presented in [TMPE04]. The scope of this infrastructure is limited to the monitoring of atomic WS. However, the idea of offering discrete service variants of one service serves perfectly well as a basis for (autonomically) controlling and managing the service level of a WSC. On the one hand, this is because algorithms and protocols used by the WSC provider to determine and negotiate the optimal service allocation for a WSC are much simpler. On the other hand, support for dynamic negotiation and provisioning for the offered services is not required.

Given a discrete set of service variations for a service included in a WSC, the WSC provider still requires a clear understanding of the dependencies between the service levels offered by the WS and the resulting service level of the WSC. More precisely, detailed knowledge of how the service level parameters are composed according to the (functional) composition pattern is needed. This aspect is particularly addressed in [JRGM05]. In [ZLD+05], a completive approach is presented, which also addresses the optimization of the service selection for dynamic WSC. However, this infrastructure builds on a proprietary workflow engine. The issue of an interoperable and standard-based instrumentation and manageability interface is not addressed. Furthermore, the automated adaptation of the WSC is triggered by changing service offerings or user preferences. An adaptation on basis of self-manageability model as part of an intelligent control loop is not considered.

In [Kap05], an interesting approach to the specification of such self-manageability policies is presented. The authors propose to create health models based on finite state machines to model the autonomic behaviour as a starting point for the manageability design. Unfortunately, it is not shown how these models are actually implemented by means of a manageability infrastructure.

With regard to the controlling instrumentation, the concept of parameterized web services flows described in [KLB05] represents a very promising approach. This allows the fully dynamic selection of included services at runtime by adding and evaluating corresponding mapping rules within the WSC. Unfortunately, these selection rules may not be changed at runtime. They are rather set at design time. Our solution can therefore be regarded as complementary to the aforementioned approach.

## 3 Motivating Example and Self-Manageabililty Requirements

In this section, we introduce a simplified real-life scenario motivating the application of a semi-dynamic service selection. On the basis of this scenario, we present major requirements for a corresponding manageability infrastructure and the controlling WSC instrumentation. Furthermore, assumptions and limitations implied by the scenario are pointed out.

The scenario is situated in the field of higher education. More precisely, we are working on a project that is concerned with the development of a SOA which is supposed to support study and administrative processes [FJL+06]. In the following, we focus on a service-oriented IT support for the process of managing examinations. An examination management system (EMS) is offered by the central administration. The functionality (e.g., registering for exams or capturing exam results) of the EMS is exposed through atomic WS, also provided by the central examination. The examination management process is supported by process-oriented, long-running WSC. The departments account for developing, adapting and operating their specific WSC, as they are specific to the study courses offered by them.

The WSC provider solely accounts for ensuring the guaranteed service levels. The service level of a WSC thereby relies on the workload, the service levels of the composed WS and the performance of the execution environment, in particular the employed composition engine. The WSC has to be adjusted to a changing workload. The WS provider – the central administration – offers several variants of the same service with regard to the provided service level. Depending on the given workload, the WSC provider – the university departments – may choose the most appropriate variant of the WS.

The long-running WSC supporting the process of managing examinations, is illustrated in Figure 1. This WSC requires the RegistrationService and the ExaminationResult-Service provided by the EMS and an additional TaskManagementService supporting user interactions. As the different operations offered by the *ExaminationMgmt-Composition* are used via a web-based portal, the response time (RT) of each operation,

particularly the registration, must not exceed 2 sec. All included WS are offered by the central administration in two variants, which represents a simplified setting. All variants guarantee a RT of less than 1 sec, but with different workload limits, expressed in requests per minute (req/min). Variant B handles 500 req/min and A tolerates up to 1000 req/min. The offered WS variants are provided through different service endpoints.

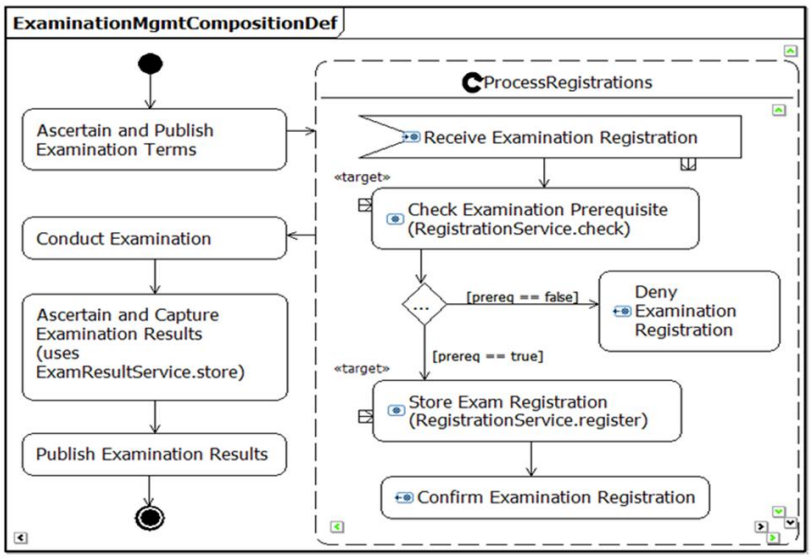


Figure 1. ExamMgmtService - Composition Definition

We focus on the looped activity for processing registrations. This activity is started after the examination terms have been assessed. Registrations submitted by students are first received using the WSC's operation register, then validated and stored by invoking the respective operations offered by the RegistrationService. Depending on the validation result, a confirmation or denial message is returned to the caller, again through the operation register. As a result, the response time of this operation is mainly determined by this sub-process. Using QoS composition patterns as presented in [JRGM05], the following response time for the whole operation can be estimated.:

$$RT_{WSC.register} = RT_{RegistrationService.check} + P(prereq == true)\ RT_{RestrationService.register}$$

For the sake of simplification, we neglect the constant time factor required by the BPEL engine itself. Given the available offerings, in the worst case ($P = 1$) this would lead to an aggregated RT of less than 2 sec, which in any case complies with the given requirement. As for the workload (WL), we need to know how it is distributed to the included service operations:

$$WL_{RegistrationService.check} = WL_{WSC.register} \quad (1)$$
$$WL_{RestrationService.register} = P(prereq == true)\ WL_{WSC.register} \quad (2)$$

In this scenario, a feasible self-manageability policy would be to switch to Variant A of the RegistrationService as soon as an average workload greater than 500 req/min has

been detected during the last hour. Otherwise, Variant B is used. With regard to the instrumentation this implies that on the one hand it is necessary to monitor the workload. On the other hand, a mechanism for changing the actually employed service variant at runtime is required, even for already running WSC instances. In our case, for each examination one long-running instance of the WSC is created.

Note that this is a very simple scenario intended to demonstrate self-manageability requirements. Within the service selection, we particularly neglect the costs of each service variant and present a very limited, static service offering. A larger variety of dynamically changing offerings as well as the inclusion of cost would result in a complex optimization problem, as presented in [GJ05]. Solving this problem would then be part of the intelligent control loop (plan function) implemented by an autonomic manager. However, this aspect is part of our future work.

## 4 Controlling Instrumentation Design

To provide self-manageability a controlling instrumentation of the WSC is required in the first place. More precisely, extensions of the WSC implementation are needed that allow a dynamic reconfiguration of the actually employed service variants at runtime. We already pointed out two major requirements this instrumentation must meet:

- Support for reconfiguration of running WSC instances
- Applicability for all kinds of BPEL engines

Taking these requirements into account we identified two feasible approaches. The first one represents the employment of proxy WS. In this case, a proxy is generated for each included service which offers the same WS interface as the original WS. The WSC includes only the proxy WS. When calling it, the proxy determines the service endpoint of the actual WS variant, invokes it and returns the result to the WSC. The endpoint may either be retrieved from a configuration services or from a local properties file or database. In the latter case, the proxy has to offer an interface for updating this information.

This approach has some advantages. First, the proxy can easily be generated as interfaces are identical to the original WSDL and internal logic is straight forward. Second, configuration changes directly affect all running instances as well as instances that will be newly created without having to explicitly change/reconfigure them. As a major drawback, this solution in either case requires at least one additional call of the proxy. If the configuration service is asked for the endpoint, another additional call is needed. This is why – after implementing this approach – we looked for a less resource demanding alternative.

The constraint of being platform-independent led us to the employment of dynamic endpoint references, as proposed by the BPEL 1.1 standard [ACD+03]. This mechanism allows a dynamic reconfiguration of the service endpoint for a given *partnerLink* at

runtime by using a standard <*assign*> activity. However, the WSC has to be provided with the information on which endpoint it has to use for a particular *partnerLink*. Moreover, it has to be possible to change this configuration information within a running WSC instance. This calls for extensions of the BPEL-based composition definition as well as the provided WSC interface in terms of the corresponding WSDL. Figure 2 shows an instrumentation pattern for extending arbitrary BPEL definitions with controlling capabilities.

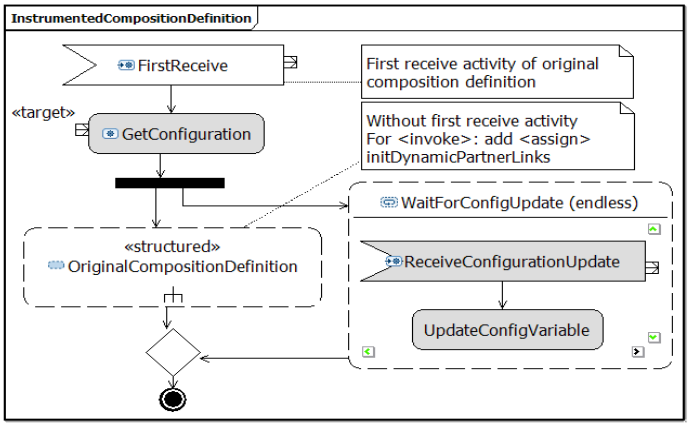


Figure 2. Controlling Instrumentation - BPEL Alternative

An additional invocation activity, that is inserted after the first receive activity, retrieves the service variant configuration from a *configuration service*. This information is stored in a newly added BPEL variable. An AND split divides the execution path into two branches executed in parallel. The first branch holds the original composition definition as an embedded sub process. Additionally, an <*assign*> that initializes the dynamic *partnerLinks* has to be added before each <*invoke*> activity. The second branch enables the asynchronous receiving of configuration updates. Once a new configuration has been received, the local configuration variable is updated. To continuously provide the possibility of reconfiguration, these activities are placed within an endless loop. The composition terminates as soon as the original composition situated in the first branch terminates, as these branches are joined by means of an OR join. Unfortunately, the combination of AND split and OR join is not explicitly supported by BPEL yet. Thus, in the BPEL-based implementation, a standard <*flow*> activity is used, which corresponds to an AND split with an AND join. To terminate the second branch, a custom fault event is thrown after the original composition has completed. The fault event is caught in an empty exception handler added to the outmost <*scope*>. In this way, the whole composition is terminated.

It is obvious that the BPEL-based instrumentation is a more efficient approach than the proxy alternative in terms of management-related overhead at runtime. Additional service invocation activities are only required once at the beginning and as soon as a reconfiguration is actually desired by the manager. Nevertheless, at design time this approach causes a higher complexity. This problem is addressed in the following section.

# 5 Automated Generation of a Bpel-Based Controlling Instrumentation

In this section we present an XSLT for the automated transformation of a given BPEL composition definition into an instrumented composition definition. The following code snippet displays the structure of a typical BPEL definition.

```
<process name="SomeWSC" […]>
   <variables>[…]</variables>
   <partnerLinks>[…]</partnerLinks>
   <correlationSets>[…]</correlationSets>
   <faultHandlers>[…]</faultHandlers>
   activity+
</process>
```

Such BPEL definitions represent the typical source XML for the transformation. The XSLT fragment given below operates on this XML and produces the instrumented WSC.

```
[…]<process name="SomeInstrumentedWSC" […]/>
 <variables>
   <variable name="ConfigData" […]/>
   <variable name="ProcessID" […] />[…]
   <xsl:call-template name ="SourceWSC_GlobalVars"/>
 </variables>
 <partnerLinks>
   <partnerLink name="ConfigurationService" […] />
   <partnerLink name="ManagementConsumer" […]/>
   <xsl:call-template name="SourceWSC_PartnerLinks"/>
 </partnerLinks>
 <xsl:call-template name="SourceWSC_CorrSets"/>
 <xsl:call-template name="SourceWSC_FaultHandlers"/>
 <sequence>
   <xsl:call-template name="SourceWSC_FirstReceive"/>
   <scope name="InitializeGlobalVariables">[…]
     <invoke name="GetConfigurationData" […]/>[…]
   </scope>
```

```
<scope name="instrumentedActivities">
 <faultHandlers>
  <catch faultName="bpws:forcedTermination"[…]/>[…]
 </faultHandlers>
 <flow>
  <sequence>
   <xsl:call-template name="SourceWSC_Activities"/>
   <throw name="SignalEndOfProcess" […]/>
  </sequence>
  <sequence>
   <while condition="true()">
    <sequence>
     <receive name="ReceiveConfigurationData"[…]/>
     <assign name="UpdateConfigurationData">[…]
     </assign>
    </sequence>
   </while>
  </sequence>
 </flow>
</scope>
</sequence>[…]
```

The transformation complies with the previously introduced instrumentation approach. So we omit detailed explanations at this point and rather focus on particular challenges.

Using XSLT represents a very comfortable way for realizing XML-to-XML transformations. However, we encountered serious problems while trying to implement one particular transformation rule, namely the initialization of the dynamic *partnerLinks* prior to each service invocation. This requires a recursive traversing of the XML tree where each <*invoke*> activity is extended by the additional <*assign*> activity. The implementation of this rule on basis of XSLT turned out to be very intricate. So we decided to use an additional, simple Java program in order to apply this particular modification to the source BPEL definition. Here, a Document Object Model (DOM) is created of the source XML and modified in the previously described way.

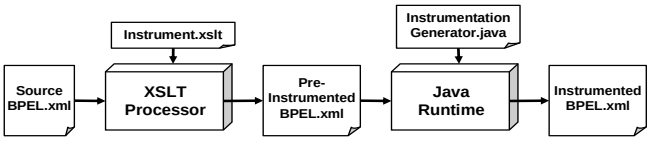


Figure 3. Complete BPEL-Instrumentation Procedure

To finalize the instrumentation, the corresponding WSDL has to be modified as well. This transformation is very similar to the BPEL transformation. Therefore, we do not provide detailed explanations in this case. Basically, an operation *updateConfigurationData* along with the necessary message and type definitions is added to the first *portType*. The input type definition of this operation is defined as follows.

```
<complexType name="ConfigurationDataType">
 <sequence>
    <element name="SelectedWSVariant" maxOccurs="unbounded">
       <complexType>
          <sequence>
             <element name="address" type="string"/>
             <element name="defaultAddress" type="string"/>
          </sequence>
          <attribute name="ID" type="string"/>
          <attribute name="AssociatedServiceID" type="string"/>
       </complexType>
    </element>
 </sequence>
</complexType>
```

Moreover, the type definitions required for the previously introduced BPEL extensions are added. Within the *partnerLinkType* for the BPEL process itself; a further role, *ManagementProvider*, is inserted. Finally, a correlation property alias for the configuration update message is appended.

# 6 WSC Manageability Design

In this section we now focus on the actual manageability design. This includes the specification of a basic self-manageability model and a corresponding management information model. The approach is demonstrated by means of the motivating example presented in section 3.

The self-manageability model defines the autonomic behavior, namely the control loop, which is implemented by an autonomic manager. We decided to use a finite state machine to specify this aspect. This basically follows the health models presented in [Kap05], but in an adapted and simplified way. Figure 4 shows the self-manageability model for the processing of registrations. That is a basic control loop for adjusting the procured response time by dynamically selecting a suitable service variant of the employed *RegistrationService*.

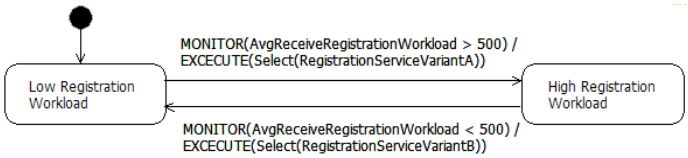


Figure 4. Self-Manageability Model for ProcessRegistrations

Accordingly, the states reflect the current registration workload situation. Note that both states are healthy. We omitted the unhealthy states for each workload level, because they would be too ephemeral. Each transition comprises two parts: First a condition that leads

to its triggering and second an action that is executed. A state transition is triggered in case certain conditions for relevant metrics are met. The observation of the metrics and the evaluation of the conditions are realized by the monitoring function. In our case, the transition from state "variant b selected" is triggered as soon as the monitor detects a threshold exceedance for the registration workload. This results in an action, namely the selection of service variant A, as part of the execute function.

The required metrics, actions as well as necessary monitoring and configuration information have to be specified in terms of a management information model. In the following, we present a corresponding WSC information model based on the Common Information Model (CIM) [DMT99]. This model basically represents an extension of the CIM metrics model, comprises particular WSC management information and reflects the specific structure of WSC. The model elements required for the monitoring of WSC have already been presented in [MMRA07, Rat07]. Here, managed elements (ME) for the WSC as a whole, the different internal WSC elements and the included WS are specified. For each ME, information about each executed WSC instance and information related to the general definition of the WSC, like configuration settings, is distinguished. The *UnitOfWork* concept serves as the basis for all execution-related WSC ME. The definitional ME on the other hand are modeled by means of a corresponding *UnitOfWorkDefiniton*. In the following, we present excerpts of the WSC information model that are most relevant to enabling the desired self-manageability. First, we focus on the definition of the required metric (see Figure 5).

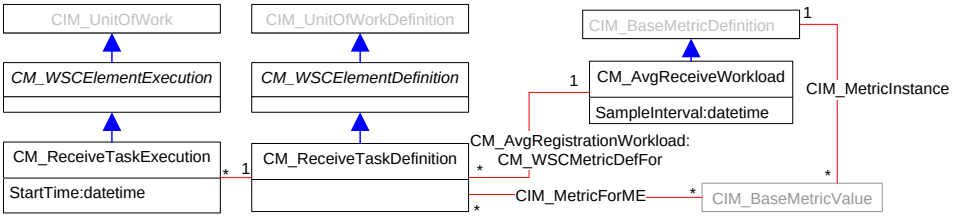


Figure 5. WSC Information Model – Metric Definition

By extending a *CIM_BaseMetricDefintion*, we first define a metric for the average receive workload. This generic metric definition reflects the average number of received requests per minute within a specified sample interval. In the case of the required registration workload, the sample interval is set to 3 hours. Furthermore, this metric is associated with a CM_ReceiveTaskDefinition for the activity "Receive Examination Registration" (see Figure 1). This allows for navigating all executed instances, each represented as an instance of *ReceiveTaskExecution*. As these elements contain a parameter *StartTime*, the number of created instances within the specified sample interval is rendered possible. This is how the metric is calculated.

In addition, the WSC information model has to store information about the available service variants and offer means for assigning the actually selected variant. The following model (Figure 6) fragment shows the proposed solution to this problem.

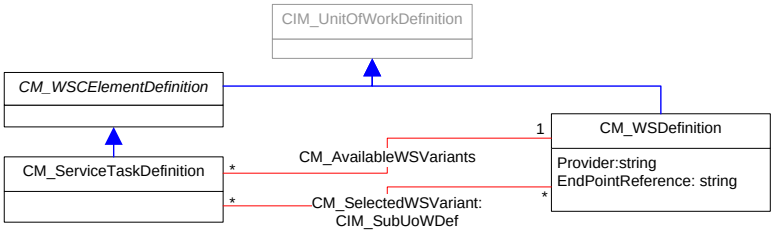


Figure 6. WSC information model –Serivce Variant Configuration

External WS are generally invoked within the scope of service tasks, as part of a WSC composition definition. A *CM_WSDefinition* is created for each available service variant and associated with the corresponding *CM_ServiceTaskDefintion* through the custom association *CM_AvailableWSVariants*. This association implies that all linked WS definitions are compatible with the service task, meaning their offered interfaces thoroughly match. This is indicated by equivalent *ServiceIDs*. The actually used WS variant is specified by means of the custom association *CM_SelectedWSVariant*. This selected WS variant has to be contained in the set of available WS variants. The responsible CIM provider supports a modification of this association. Thus, the required action of reconfiguring the service selection corresponds to a modification of this association. The provider then uses the WSC instrumentation to effectively change the selection. A detailed explanation of the selection procedure is provided in the following section.

# 7 WSC Manageabilitiy Infrastructure Implementation

In this section, we present the implementation of a WSC manageability infrastructure which is based on our preliminary work [MMRA07, Rat07]. Accordingly, a manageability infrastructure for the monitoring of WSC was already available. In this case, the monitored WSC is implemented on basis of the Oracle BPEL Process Manager. The manageability infrastructure is built on the management architecture proposed by WBEM [Eck03]. As a CIMOM we employed the Java-based WBEMServices [2].

As the interface between the CIMOM and associated CIM provider is not standardized, provider implementations for a specific CIMOM cannot typically be used with other CIMOMs without modification [DKG04]. Therefore, we draw a distinction between a CIMOM-specific and CIMOM-independent part (see Figure 7). The CIMOM-specific part comprises different CIM provider responsible for the managed WSC elements. Each provider implements the specific interfaces defined by the WBEMServices framework and performs the necessary data mapping between the CIMOM-independent part and the specific CIMOM, where we placed the actual processing logic.

[2] http://wbemservices.sourceforge.net/

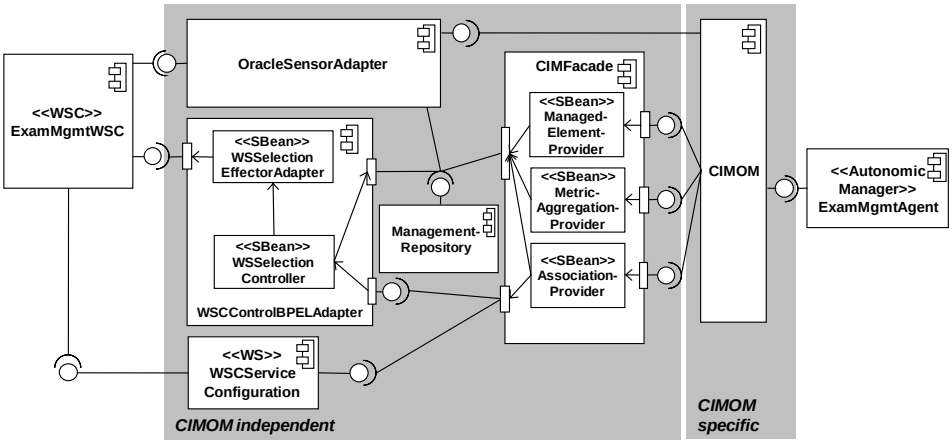

Figure 7. WSC Manageability Infrastructure Implementation

Since the employed execution environment is based on a Java application server, we decided to build the CIMOM independent part using Enterprise Java Beans (EJB3). Here, the component *CIMFacade* contains generic provider implementations for handling association, managed elements and aggregated metrics. The interfaces of these session beans match the standardized interfaces supported by a CIMOM. This allows an easy migration to another CIMOM implementation [DKG04]. Entity beans are used to persistently store the management information. These are subsumed under the component *ManagementRepository*. The required monitoring instrumentation of the WSC on the other hand is based on Oracle-specific sensors added to the WSC definition. The communication with this active instrumentation is handled by the *OracleSensorAdapter*, which is comprised of a message-driven bean and specific session beans that account for updating the managed elements and generating respective CIM indications.

These components so far allow a very fine-grained monitoring of WSC, down to the level of single instances and internal WSC elements such as service tasks or gateways. To support self-manageability through dynamic selection of WS variants at runtime further components and modifications are required. First, we introduce a simple *AutonomicManager* component implementing the state machine defining the control loop presented in section 6. The agent's monitor function polls the metric provider for detecting a threshold exceedance for the average registration workload. The control function on the other hand uses the association provider to change the selected WS variant by modifying the association *SelectedWSVariant.* This configuration is provided to the WSC by the WS *WSCServiceConfiguration*, which basically maps the information to an XML-based configuration specification the WSC understands. With a proxy-based instrumentation these extensions would already be sufficient. But when using the BPEL-based instrumentation, all currently running WSC instances additionally have to be updated with the modified configuration. This particular requirement is tackled by the *WSCControlBPELAdpater*. Here, the session bean *WSSelectionEffectorAdapter* provides a unified interface to the respective management operation offered by the WSC. The *WSSelectionController* on the other hand assures that the configuration update is

propagated to all relevant WSC instances. The currently active instances are identified by querying the *ManagementRepository* for all *WSCExecution* objects for the respective *WSCDefinition* where the status equals "*active*". Then for each retrieved *WSCExecution* object, the operation *updateConfigurationData* is invoked through the *ServiceSelectionEffectorAdapter*. In this context, the WSC instance identifier as part of the *WSCExecution* object and the WSC endpoint reference along with the current configuration as available from the respective *WSCDefinition* object are particularly required. The complete interaction for changing a WS variant selection is summarized on Figure 8. Note that this model is simplified for the sake of clarity. The CIMOM-specific providers are left out and meaningful method names are used.

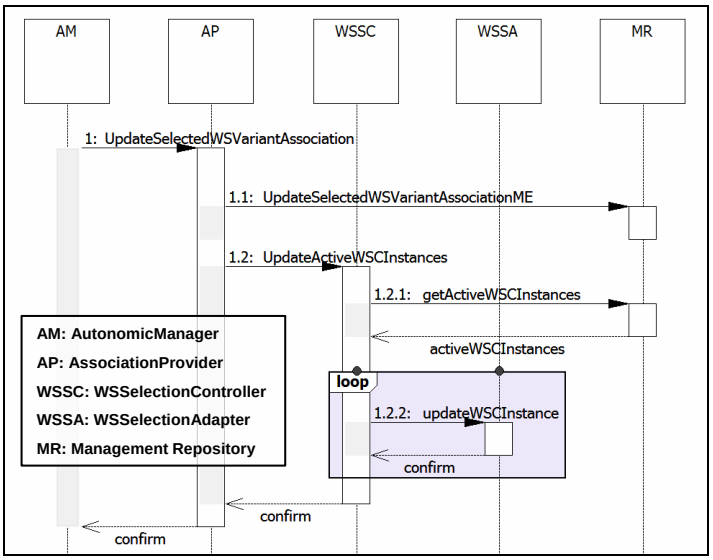


Figure 8. Sequence for Changing selected WS Variant

## 8 Discussion and Outlook

In this paper, a pragmatic approach to the conceptual design and implementation of a WSC manageability infrastructure with support for self-manageability has been presented. To this end, different techniques for realizing the required controlling instrumentation have been introduced. In addition, we showed how the BPEL-based instrumentation can be generated automatically. So far, however, the solution is limited to semi-dynamic WSC. Yet, by incorporating approaches to parameterized WSC [KLB05], it could be further enhanced to offer support for fully dynamic WSC.

As to the self-manageability design, the scope is so far limited to very simple scenarios. Here, further research on the modeling of autonomic behavior for more complex scenarios is required. In this case, the employment of finite state machines could result in an unacceptable amount of states. An alternative to this would, for instance, be the employment of Event-Condition-Action (ECA) rules or management policies [JJSC03].

As mentioned earlier, an optimization of the selection that works on a larger variety of service variants and also takes into consideration cost aspects is not yet supported either. The related work has so far neglected the usage profile as an additional constraint for the optimization problem. This is because linear programming approaches are not sufficient in this context.

As to the scenario, the general question arises whether the employment of load balancing on the WS level would be a superior approach for automatically adjusting to a given workload. This research question has not been addressed in this paper. However, one argument against load balancing is that it causes more complexity for the WS provider with regard to accounting and billing as well as the provisioning of the services. As far as the optimization of the employed hardware is concerned, there might also be a disadvantage. Knowledge about the business processes is not included in the optimization process. In contrast to the WSC provider, the WS provider does not know about workload peaks implied by the business process. Consequently, it is harder for the provider to anticipate workload peaks and react to them. In the case of semi-dynamic service selection, the WSC provider may specify policies for service selection derived from business process knowledge.

Our current research particularly focuses on a methodology for an automated generation of the WSC manageability infrastructure along with the required WSC instrumentation. On the one hand, this comprises the design of domain-specific meta models that allow for the modeling of manageability aspects as part of an integrated development process of WSC. On the other hand, transformations to a fully functional manageability infrastructure are of special concern. This way, modifications to the specific target platform, like the employment of a different BPEL engine or the support for different management protocols, can be achieved by defining specific transformations.

# 9 Literaturverzeichnis


[ACD+03] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, and S. Thatte. Business Process Execution Language for Web Services Version 1.1, 2003.

[DDK+04] A. Dan, D. Davis, R. Kearney, A. Keller, R. King, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer, and A. Youssef. Web services on demand: WSLA-driven automated management. *IBM Systems Journal*, 43(1):136–158, 2004. 1014728.

[DK03] Markus Debusmann and Alexander Keller. SLA-Driven Management of Distributed Systems Using the Common Information Model. In *8th IFIP/IEEE International Symposium on Integrated Network Management (IM 2003)*, 2003.

[DKG04] M. Debusmann, R. Kroger, and K. Geihs. Unifying service level management using an MDA-based approach. In *IEEE/IFIP Symposium on Network Operations and Management (NOMS 2004)*, volume 1, pages 801–814 Vol.1, 2004.

[DMT99] DMTF. CIM Specification, Version 2.2. http://www.dmtf.org/standards/cim/cim_spec_v22, 1999.

[Eck03] D. Eckstein. WBEM Infrastructure Introduction. Global Management Conference, 2003. http://www.dmtf.org/events/past/2003/gmc/presentations/GMC03_315pm_BasicconceptsWBEM.pdf.

[FJL+06] P. Freudenstein, W. Juling, L. Liu, F. Majer, A. Maurer, C. Momm, and D. Ried. Architektur für ein universitätsweit integriertes Informations- und Dienstmanagement. In *INFORMATIK 2006*, volume P-93 of *Lecture Notes in Informatics*, pages 50–54, Dresden, 2006. Springer.

[GJ05] R. Grønmo and M.C. Jaeger. Model-Driven Methodology for Building QoS-Optimised Web Service Compositions. In *5th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'05)*, page 68–82, 2005.

[IBM04] IBM. An architectural blueprint for autonomic computing, 2004.

[JJSC03] J. Jun-Jang, J. Schiefer, and H. Chang. An agent-based architecture for analyzing business processes of real-time enterprises. In *Seventh IEEE International Enterprise Distributed Object Computing Conference (EDOC 2003)*, pages 86–97, 2003.

[JRGM05] M.C. Jaeger, G. Rojec-Goldmann, and G. Muhl. QoS aggregation in Web service compositions. In *The 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05)*, pages 181–185, 2005.

[Kap05] V. Kapoor. Services and autonomic computing: a practical approach for designing manageability. In *2005 IEEE International Conference on Services Computing*, volume 2, 2005.

[KLB05] D. Karastoyanova, F. Leymann, and A. Buchmann. An approach to parameterizing web service flows. In *2005 International Conference on Service-oriented Computing (ICSOC'05)*, pages 533–538, 2005.

[LRS02] F. Leymann, D. Roller, and M.-T. Schmidt. Web services and business process management. *IBM Systems Journal*, 41(2), 2002.

[MMRA07] Christof Momm, Christian Mayerl, Christoph Rathfelder, and Sebastian Abeck. A Manageability Infrastructure for the Monitoring of Web Service Compositions. In *14th HP-SUA Workshop*, Munich, Germany, 2007.

[Rat07] Christoph Rathfelder. *Management in serviceorientierten Architekturen: Eine Managementinfrastruktur für die Überwachung komponierter Webservices*. VDM Verlag Dr. Müller, Saarbrücken, November 2007.

[SMS+02] A. Sahai, V. Machiraju, M. Sayal, A. van Moorsel, and F. Casati. Automated SLA Monitoring for Web Services. In *13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2002)*, volume 2506 of *Lecture Notes in Computer Science*, pages 28–41, Montreal, Canada, 2002. Springer.

[TMPE04] V. Tosic, W. Ma, B. Pagurek, and B. Esfandiari. Web Service Offerings Infrastructure (WSOI) - a management infrastructure for XML Web services. In *IEEE/IFIP Network Operations and Management Symposium (NOMS 2004)*, volume 1, pages 817–830 Vol.1, 2004.

[TP05] V. Tosic and B. Pagurek. On comprehensive contractual descriptions of Web services. In *IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE '05)*, pages 444–449, 2005.

[TPP03] V. Tosic, B. Pagurek, and K: Patel. WSOL - A Language for the Formal Specification of Classes of Service for Web Services. In *The 2003 International Conference on Web Services (ICWS'03)*, Las Vegas, 2003.

[vdAtHW03] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, and Mathias Weske. Business Process Management: A Survey. In *International Conference on Business Process Management (BPM 2003)*, volume 2678 of *Lecture Notes in Computer Science*, pages 1–12, Eindhoven, The Netherlands, 2003. Springer.

[ZK06] O. K. Zein and Y. Kermarrec. Static/Semi-Dynamic and Dynamic Composition of Services in Distributed Systems. In *International Conference on Internet and Web Applications and Services*, pages 144–144, Brest Cedex, France, 2006.

[ZLD+05] L. Zeng, H. Lei, M. Dikun, H. Chang, K. Bhaskaran, and J. Frank. Model-driven business performance management. In *IEEE International Conference on e-Business Engineering (ICEBE 2005)*, pages 295–304, 2005.