Manageability Design for an Autonomic Management of Semi-Dynamic Web Service Compositions

Christof Momm, Ignacio Pérez Hallerbach, Sebastian Abeck
Universität Karlsruhe (TH), Institute of Telematics, C&M IT Research, Zirkel 2,
76131 Karlsruhe, Germany
{momm, hallerb, abeck}@cm-tm.uka.de

Christoph Rathfelder
FZI Research Center for Information Technology, Software Engineering
Haid-und-Neu-Straße 10-14
76131 Karlsruhe, Germany
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.**

Keywords- Web Service Composition Management; Autonomic Computing; Manageability Design; Instrumentation; WBEM; CIM

I. 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). In SOA, functionality required for executing business processes is provided by atomic web services (WS) or by web service compositions (WSC) [1].

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). 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 control the service execution in order to prevent SLA violations. These management functions should be automated as far as possible [2]. so that the vision of an "on-demand" provisioning of services can be reached [3].

An automated service level management for WSC can – at least partly - be achieved by applying autonomic computing concepts as presented in [4]. 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 monitoring, analyze, plan and execute function. Hence, the managed resources, in our case the WSC, have to provide an adequate manageability interface allowing monitoring and controlling access. This interface is also referred to as "instrumentation".

This paper focuses on the controlling instrumentation of WSC based on the Business Process Execution Language (BPEL) and the enhancement of a WSC manageability infrastructure by incorporating autonomic management concepts, in the following referred to as self-manageability. The approach is based on the design and implementation of a manageability infrastructure for WSC based on the Web-Based Enterprise Management (WBEM) standards [5]. We assume the WSC to be of semi-dynamic nature. 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 [6]. The concretely employed service variants may be selected during or prior to the execution. In this way, the service levels of WSC can be controlled.

The 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 without any vendor-specific extensions of the employed BPEL engines. Second, we show how self-manageability can be designed and implemented by means of a WSC manageability infrastructure. Here, we leverage the WBEM standards to obtain a flexible solution that may easily be integrated into existing management environments.

II. Related Work

The monitoring and 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. In [3], 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 [7], the approach is extended with interfaces using the Common Information Model (CIM), which allow an integration into traditional management application. In [2], 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. In [6], an approach that focuses on discrete web service offerings is presented. In contrast to WSLAs, the customer cannot freely negotiate all kinds of service level parameters, but may rather choose from predefined service variants. A corresponding management infrastructure limited to the monitoring of atomic WS is presented in [8] However, the idea of offering discrete service variants of one service serves perfectly well as a basis for (autonomically) controlling the service level of a WSC. This is because algorithms and protocols to determine and negotiate the optimal service allocation for a WSC are much simpler.

Given a discrete set of service variations for a WS 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. This aspect is particularly addressed in [9]. In [10], a completive approach is presented, which also addresses the optimization of the service selection for dynamic WSC. A corresponding execution infrastructure is provided in [11]. However, this infrastructure builds on a proprietary workflow engine. Furthermore, the automated adaptation of the WSC is triggered by changing service offerings or user preferences. An adaptation on basis of self-manageability is not considered.

In [12], an interesting approach to the specification of such self-manageability policies is presented. The author proposes 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 [13] represents a very promising approach. This allows a dynamic selection of included WS 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. So our solution can be regarded as complementary to the aforementioned approach.

III. CONTROLLING INSTRUMENTATION DESIGN

To provide self-manageability a controlling instrumentation of the WSC is required in the first place. More precisely, non-functional extensions of the WSC implementation are necessary that allow a dynamic reconfiguration of the actually employed service variants at runtime. In this context, two major requirements must be met: Support for reconfiguration of running WSC instances and applicability to 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 WS 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 responsible configuration provider or a local properties file or database. In the latter case, the proxy has to offer an interface to the provider for updating this information.

This approach has some advantages. First, the proxy can easily be generated as its interface is 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. This is why - after implementing this approach - we looked for a less resource demanding alternative.

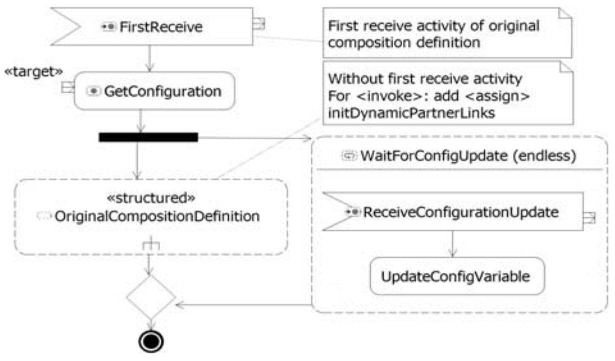
FirstReceive
First receive activity of original composition definition
«target»
GetConfiguration
Without first receive activity
For <invoke>: add <assign> initDynamicPartnerLinks
WaitForConfigUpdate (endless)
«structured»
OriginalCompositionDefinition
ReceiveConfigurationUpdate
UpdateConfigVariable

Figure 1. Controlling Instrumentation - BPEL Alternative

The second approach uses dynamic endpoint references. This mechanism allows a reconfiguration of the actually invoked service endpoint for a given *partnerLink* at runtime. 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. Figure 1 shows an instrumentation pattern for extending arbitrary BPEL definitions with controlling capabilities.

First, an additional invocation activity retrieves the service variant configuration from a configuration provider published as a WS. This information is stored in a newly added BPEL variable. An inserted AND split divides the execution path into two branches executed in parallel. The first branch holds the original composition definition without the first receive activity as an embedded sub process. Moreover, an <*assign*> that initializes the dynamic *partnerLinks* is added before each embedded <*invoke*> activity. The second branch enables the asynchronous receiving of configuration updates which are stored in the local configuration variable. 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 standard BPEL does not support OR joins the employed <*flow*> activity is terminated through a custom fault event that 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.

We argue that the BPEL-based instrumentation is a more efficient approach than the proxy alternative in terms of management-related overhead at runtime. This is because additional service invocations 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. For this reason, we implemented an XSLT based transformation to extend the BPEL composition definition with the necessary management code.

IV. WSC Manageability Design

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 [12], but in an adapted and simplified way. Figure 2 shows a self-manageability model. That is a basic control loop for adjusting the response time procured by a service operation through dynamically switching between the available service variants for the included Service.

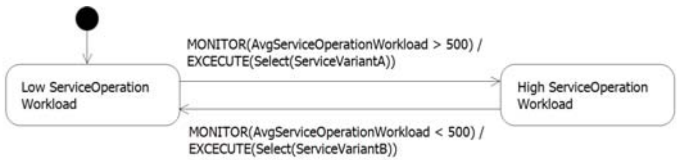
MONITOR(AvgServiceOperationWorkload > 500) /
EXCECUTE(Select(ServiceVariantA))
Low ServiceOperation Workload
High ServiceOperation Workload
MONITOR(AvgServiceOperationWorkload < 500) /
EXCECUTE(Select(ServiceVariantB))

Figure 2. Sample Self-Manageability Model

Each transition comprises two parts: A condition that leads to its triggering and 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.

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 CIM. The model elements required for the monitoring of WSC have already been presented in [5]. 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. 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 3).

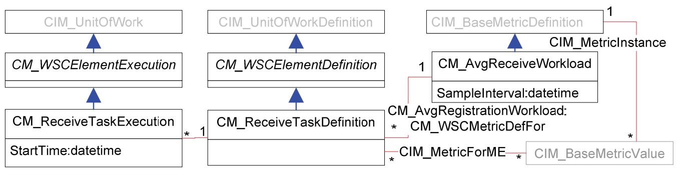

Figure 3. WSC Information Model – Metric Definition

By extending a *CIM_BaseMetricDefintion*, we first define a metric for an average receive workload. This generic metric definition reflects the average number of received requests per minute within a specified sample interval. Furthermore, this metric is associated with a *CM_ReceiveTaskDefinition*. This allows for navigating all executed instances, each represented as an instance of *ReceiveTaskExecution.* In addition, the WSC information model has to store information about the available service variants and offer means for assigning the actually selected variant (Figure 4). The following model fragment shows the proposed solution to this problem.

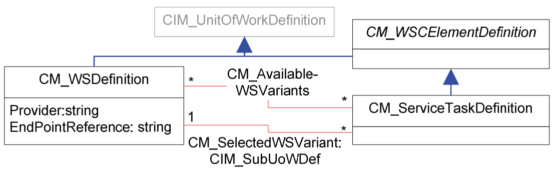

Figure 4. WSC information model –Serivce Variant Configuration

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. The actually used WS variant, which has to be contained in the set of available WS variants, is specified by means of the custom association *CM_SelectedWSVariant*. 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.

V. WSC Manageabiltiy Infrastructure Implementation

In this section, we present the implementation of a WSC manageability infrastructure following the previously introduced design. This implementation is based on our preliminary work [5]. Accordingly, a manageability infrastructure for the monitoring of WSC was already available.

As the interface between the CIMOM and associated CIM provider is not standardized, provider implementations for a specific CIMOM cannot typically be used with another CIMOM without modification [14]. Therefore, we draw a distinction between a CIMOM-specific and CIMOM-independent part (see Figure 5). The CIMOM-specific part comprises different CIM provider responsible for the managed WSC elements. Since the employed execution environment is based on a JEE application server, we decided to build the CIMOM independent part on Enterprise Java Beans (EJB3). The *CIMFacade* contains generic provider implementations, which allow an easy migration to another CIMOM implementation. Entity beans subsumed under the *ManagementRepository* are used to persistently store the management information. As described in [5], we use Oracle-specific sensors added to the WSC definition which communicate with the *OracleSensorAdapter*. In case another WSC execution environment or instrumentation approach, like polling the engine's audit trail, is used, this adapter component can easily be replaced by an adequate one.

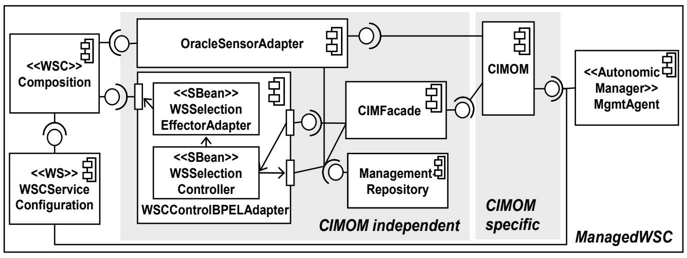

Figure 5. WSC Manageability Infrastructure Implementation

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 presented in section IV. The agent polls the metric for detecting a threshold exceedance and modifies the association *SelectedWSVariant* to change the selected WS variant. This configuration is provided to the WSC by the WS *WSCServiceConfiguration*. 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. This particular requirement is tackled by the *WSCControlBPELAdpater*. Here, the *WSSelectionEffector-Adapter* provides a unified interface to control the WSC. The *WSSelectionController* assures that the configuration update is propagated to all relevant WSC instances. The currently active instances are identified by querying the *Management-Repository* for all *WSCExecution* objects for the respective *WSCDefinition* where the status equals "*active*". Then for each retrieved *WSCExecution* object, the operation *updateConfigurationData* is invoked.

VI. 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. So far, however, the solution is limited to semi-dynamic WSC. 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 optimization of the selection that considers cost aspects is not yet supported either. 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. 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.

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

References

[1] F. Leymann, D. Roller and M.-T. Schmidt, "Web services and business process management", *IBM Systems Journal*, vol. 41, no. 2, 2002.

[2] A. Sahai, V. Machiraju, M. Sayal, A. van Moorsel and F. Casati, "Automated SLA Monitoring for Web Services", *Proc. 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2002)*, Springer-Verlag, 2002, pp. 28-41.

[3] 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*, vol. 43, no. 1, 2004, pp. 136-158.

[4] IBM, "An architectural blueprint for autonomic computing", 2004; http://www-03.ibm.com/autonomic/pdfs/ACBP2_2004-10-04.pdf.

[5] C. Momm, C. Mayerl, C. Rathfelder and S. Abeck, "A Manageability Infrastructure for the Monitoring of Web Service Compositions", *Proc. 14th HP-SUA Workshop*, 2007.

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

[7] M. Debusmann and A. Keller, "SLA-Driven Management of Distributed Systems Using the Common Information Model", *Proc. 8th IFIP/IEEE International Symposium on Integrated Network Management (IM 2003)*, 2003.

[8] V. Tosic, W. Ma, B. Pagurek and B. Esfandiari, "Web Service Offerings Infrastructure (WSOI) - a management infrastructure for XML Web services", *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS 2004)*, 2004, pp. 817-830 Vol.811.

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

[10] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam and Q.Z. Sheng, "Quality driven web services composition", *Proc. Proceedings of the 12th international conference on World Wide Web*, 2003, pp. 411-421.

[11] L. Zeng, B. Benatallah, A.H.H. Ngu, M. Dumas, J. Kalagnanam and H. Chang, "QoS-aware middleware for Web services composition", *IEEE Transactions on Software Engineering*, vol. 30, no. 5, 2004, pp. 311-327.

[12] V. Kapoor, "Services and autonomic computing: a practical approach for designing manageability", *Proc. 2005 IEEE International Conference on Services Computing*, 2005.

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

[14] M. Debusmann, M. Schmidt, M. Schmid and R. Kroeger, "Unified service level monitoring using CIM", *Proc. Seventh IEEE International Enterprise Distributed Object Computing Conference*, 2003, pp. 76-85.