MODEL-DRIVEN DEVELOPMENT OF MONITORED WEB SERVICE COMPOSITIONS

Christof Momm, Thomas Detsch, Michael Gebhart, Sebastian Abeck

Universität Karlsruhe (TH), Institute of Telematics, C&M IT Research,
Zirkel 2, 76128 Karlsruhe, Germany
{momm | detsch | gebhart | abeck}@cm-tm.uka.de

Abstract. Supporting business services through Web service compositions (WSC) as part of service-oriented architectures (SOA) involves various runtime monitoring requirements. The implementation of these requirements results in additional development activities. In this paper we propose a systematic approach to the development of monitored WSC based on the principles of model-driven software development. The approach helps to reduce the general complexity as well as to maintain coherency of the resulting solution in case of changing requirements.

1 Problem Statement

To offer innovative and profitable business services, companies require IT support that is tightly aligned with the corresponding business processes and highly adaptive in case of changes. These requirements can be met by employing service-oriented architectures (SOA). Here, business processes are automated - in whole or in part – through Web service compositions (WSC) [1, 2]. These WSC rely on internally or externally provided Web services (WS). Both the WSC and the WS are offered as IT services by dedicated service providers. The functional and quality-related properties are contractually fixed by means of service level agreements (SLA).

In this scenario, the quality of business services depends on the "quality" of the business processes - the business process performance - and the quality of the implementing WSC along with the included WS. Thus, to determine the quality of business services it has to be possible to monitor business process performance on basis of WSC measurements and to monitor the IT-related WSC properties [3]. The establishment of an effective monitoring requires additional development activities: (**i**) the specification of meaningful (process or service management) indicators; (**ii**) the identification of WSC runtime measurements required for calculating the indicators; (**iii**) the configuration or implementation of a corresponding WSC instrumentation; (**iv**) the configuration of the employed management tool.

The particular monitoring requirements are highly specific to the regarded business services. Also, there are intrinsic interdependencies between the functional and the monitoring implementation. Coherency between them has to be always maintained.

Hence, a systematic development approach is necessary that takes into account the monitoring requirements from the very first [4-6].

In current practice, management issues are rather considered subsequent to the functional development in terms of configuring specific management tools or frameworks (like [7, 8]). The employment of specific tools leads to solutions that are not portable, whereas the subsequent treatment of management issues increases the risk of inconsistencies. The latter problem is intensified due to the generic nature of existing management solutions. They usually support the management of arbitrary resources and therefore necessarily abstract from concrete instrumentation code. This code however forms the bridge between the functional and the management implementation. Without regarding the instrumentation, it is hard to trace the impact of changes in the functional or the management implementation. Furthermore, the developer has to deal with generic configuration models. This results in redundant activities and an unnecessary high complexity.

To overcome these drawbacks, we contribute an integrated approach to the development of monitored WSC, which leverages the principles of model-driven software development [9]. This helps to reduce complexity and allows flexible adaptations in case of changing functional or management requirements while maintaining an overall consistency of the solution.

1.1 Motivating Example

In this section, we introduce a simplified real-life scenario motivating the generation of monitored WSC. The scenario is situated in the field of higher education. In the following, we focus on a service-oriented IT support for the process of managing examinations. From a business perspective, the university departments and affiliated research groups are responsible for this process. As to the existing IT support, an examination management system (EMS) is already offered by the central administration. The functionality of the EMS is now exposed through atomic WS, which are also provided by the central examination. These WS basically wrap functionality already offered by the EMS, such as registering for exams or capturing exam results. The business process on the other hand is supported by process-oriented, long-running WSC. To a certain degree, the exam management process is specific to the study courses offered by the departments. Thus, the departments account for developing, adapting and operating their specific WSC. In the following, we focus on the sub process “Process Registrations” as shown in **Fig. 1**. This simple scenario starts with the receiving of an exam registration. Afterwards it is validated by an external WS. Depending on the validation result either a confirmation or reject message is replied.

Concering the process performance, we identified two relevant indicators for this sub process. The first one indicates the duration of the activity Validate Registration. The second indicator reflects the duration of the whole sub process. These two indicators represent the very basic monitoring requirements for our WSC. So at design time, a monitoring model that complements the functional WSC model and formalizes these requirements in a platform-independent but still executable way has to be created.

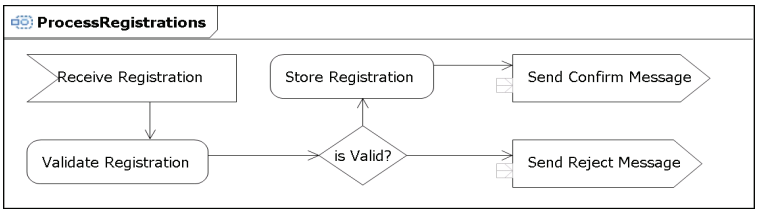

Fig. 1. Sub process "Process Registrations"

Within the specific management infrastructure, the calculation and evaluation of the indicators is handled by an external management application. So on the one hand, an adequate monitoring configuration for the specific management application has to be created on basis of the platform-independent monitoring model. On the other hand, the WSC has to be enhanced by a management interface providing the management information required for calculating the indicators and enabling their integration into the management application. In case of our sample indicators this would simply be the start and end times of the monitored activities. To retrieve this information, our WSC has to be instrumented in an adequate way.

2 Solution

2.1 Overview to the Approach

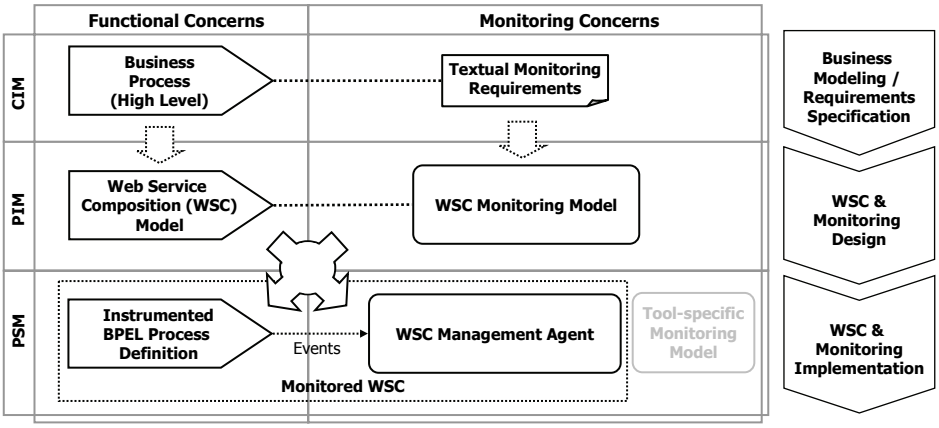

Fig. 2. Approach to Model-driven to WSC Manageability Design

According to **Fig. 2**, our solution extends a model-driven development process for WSC like [10, 11] by monitoring concerns. Thus, we distinguish between existing functional and additional monitoring models on three different layers of abstraction, namely the Computation-Independent Models (CIM), Platform-Independent Models

(PIM) and Platform-Specific Models (PSM). The platform in this case comprises the employed specific composition engine along with the specific management tools that are supported.

The CIM level includes models for specifying the business processes along with the monitoring requirements, which are expressed in terms of indicators and objectives for process or IT service management. The specification of the indicators is of informal and declarative nature. Meaningful indicators for business processes may for instance be derived from high-level business goals [12]. The resulting models are still independent from an IT support.

The PIM level is now concerned with the design of the IT support. In our case, the business process models are first refined to executable WSC models. Additionally, we offer a WSC monitoring metamodel allowing for a formal specification of the indicators including the operational semantics for calculating them. As the calculation eventually refers to runtime management information about the execution of WSC, all available WSC runtime measurements are provided by the monitoring metamodel. This part of the metamodel basically follows the structure of the Common Information Model (CIM). But unlike this, the provided managed elements are tailored to the specifics of WSC.

To put this specification into operation, we first propose an automated generation of an additional manageability infrastructure as shown in our preliminary work [13]. This basically follows the WS management pattern "Operational" presented in [14]. Accordingly, platform-specific models of a "monitored" WSC component comprising the instrumented WSC component along with a WSC management agent have to be generated. At this, the WSC instrumentation is generated automatically by configuring appropriate sensors, which we assume the employed composition engine supports. Through an additional WSC manageability interface the runtime information represented in terms of managed WSC elements is offered to other management tools. These account for calculating the specified indicators. So a specific monitoring model has to be created from the platform-independent indicator specification. This however is part of our future work.

Using this approach, a WSC (monitoring) developer can focus on the definition of the required indicators. Unnecessary technological details as well as the complex generic structures of existing monitoring models are hidden to her. This information is added automatically by applying an appropriate transformation to a specific platform. Portability of the solution is increased through the platform-independent nature of the WSC monitoring metamodel as well as a modular internal structure of the generated management agent, which as far as possible is independent from particular composition engines or management standards.

2.2 WSC Monitoring Metamodel

Since the metamodel design is based on CIM concepts, the monitoring view on a WSC is modeled through corresponding managed elements, as also proposed by [14, 15]. To support arbitrary WSC models (e.g. based on the Business Process Modeling Notation (BPMN) or UML activity diagrams), we decided on creating an independent

metamodel rather than extending an existing metamodel for the functional WSC design.

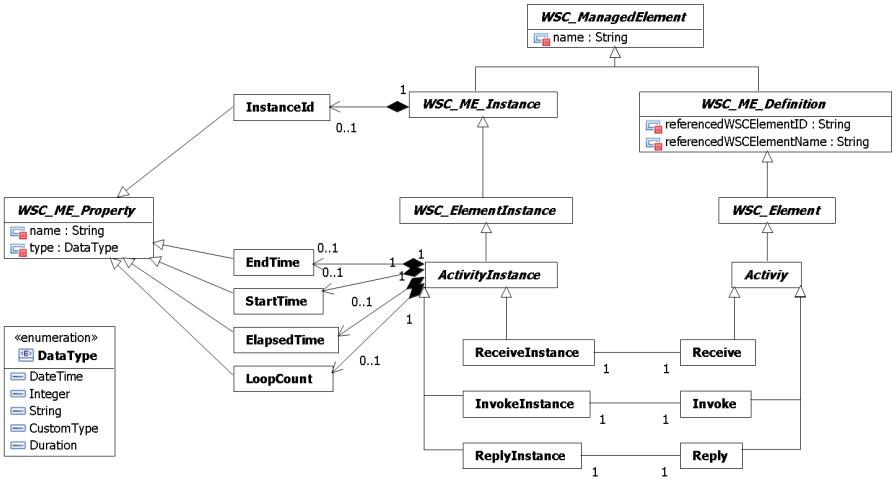

Fig. 3. Extract of WSC Monitoring Metamodel

Within our metamodel, for each functional element of the WSC model, e.g. a single activity or a decision node, a complementary pair of managed elements is available for modeling the management view. More precisely, the metamodel always offers a managed element reflecting information about each executed WSC instance (WSC_ME_Instance) and one that holds information related to the general definition of the WSC, like configuration settings (WSC_ME_Definition). To minimize the number of references to the functional model, the WSC_ME_Definition class holds a pointer to the corresponding functional element. The WSC_ME Instance classes on the other hand comprise properties reflecting the available runtime monitoring information. A common property for example is the InstanceId that provides a unique identifier for each executed instance of the monitored WSC.

The metamodel extract provided in **Fig. 3** shows the three pairs of managed elements required for modeling the management view on the different types of activities in WSCs, namely Receive, Invoke and Reply. All are based on the Activity or ActvityInstance class respectively. Besides the InstanceId four additional properties can be monitored at runtime. StartTime, EndTime and ElapsedTime are used for time based monitoring, while the property LoopCount is used to monitor the control flow – in this case loops. Note that in **Fig. 3** we omitted the meta-classes required for specifying indicators, as they are not part of the transformation focused in this paper.

Fig. 4 shows a sample instance of the WSC monitoring metamodel for the previously introduced monitoring requirements. Accordingly, for the activity Validate Registration a corresponding InvokeInstance object including the property ElapsedTime is created. This property already represents the desired duration of the activity. The second indicator requires two managed elements as the time span from the Receive Registration activity to the Send Confirm Message activity has to be monitored to get the duration of the regarded sub process. Value changes of the properties are indicated

through an Update indication. This can be used to trigger the (re-)calculation of an indicator.

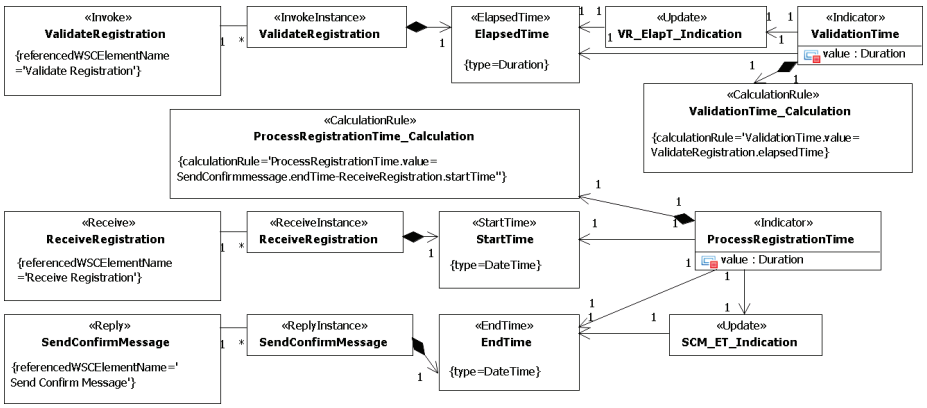

Fig. 4. Sample WSC Monitoring Metamodel

Note that basic managed elements could be generated automatically from the corresponding WSC model. Elements that are not required all for calculating the indicators could later on simply be discarded. This transformation from the WSC model to the WSC monitoring model however is part of our future work.

2.3 Generation of Monitored WSC

The model-driven generation of monitored WSC is accomplished by a transformation based on our WSC monitoring metamodel. This results in a monitored WSC, which is comprised of a WSC manageability infrastructure along with an adequate instrumentation of the WSC. **Fig. 5** provides an overview to the general architecture of a monitored WSC in a platform-independent way. Here we use a UML 2 component diagram extended by stereotypes for modeling WSC. These extensions are based on [16].

Accordingly, on the one hand a WSCComponent is created from the functional WSC model. On the other hand, a ManagementAgent component is generated from the WSC monitoring model. Both components are composed to the composite component MonitoredWSCComponent, which offers the functional Service through a ServiceInterface as well as the custom ManageabilityInterface through an additional port. In addition, a management application may subscribe for indications concerning the state of particular managed elements through a RequiredInterface.

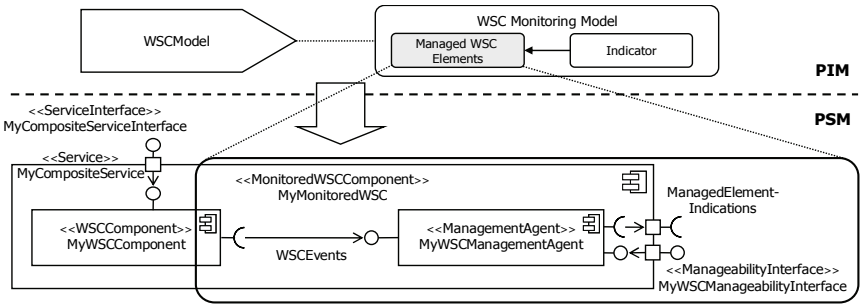

Fig. 5. General Architecture of Monitored WSC

Invocations of the ServiceInterface are handled by the functional WSCComponent, whereas requests on the management interface are delegated to the ManagementAgent. This component basically accounts for converting the event-driven monitoring information delivered by the WSCComponent to the static management view defined by the WSC monitoring model. It therefore knows the information model specified in terms of managed elements and comprises components for creating, storing and retrieving the corresponding objects as well as for generating the respective indications. For this purpose an interface for receiving instrumentation events from the WSCComponent is needed. This may for instance be a callback operation that is actively invoked by the WSCComponent or an adapter to a message queue.

As already pointed up, the required instrumentation has to be configured individually. This procedure is highly specific to the employed composition middleware, which in our case is the IBM Process Server. The process server delivers the configured monitoring events through the IBM-specific Common Event Infrastructure (CEI) [17], which in turn is based on the Java Messaging Service (JMS). So the automated WSC instrumentation involves the generation of monitoring configuration file enabling the monitoring functionality of the IBM Process Server and the integration of a CEI adapter within the management agent. In the following, we briefly explain how the model-to-model transformation from the WSC monitoring model to the IBM's monitoring configuration model, which is defined by an XML schema [17], works.

Fig. 6 shows a part of the transformation on basis of a concrete example. Here, the defined managed element Invoke with one ElapsedTime property is mapped to a monitoring configuration. For every managed element an EventSource class is generated, which instruments the corresponding functional element. Depending on the annotated property, different events for the event source are defined. In this case, the ENTRY and EXIT events are required. At runtime, the ENTRY provides the start time of the monitored activity and the EXIT event holds the end time. So the property ElapsedTime can be calculated on basis of this information. A transformation rule like this is defined for each managed element and property.

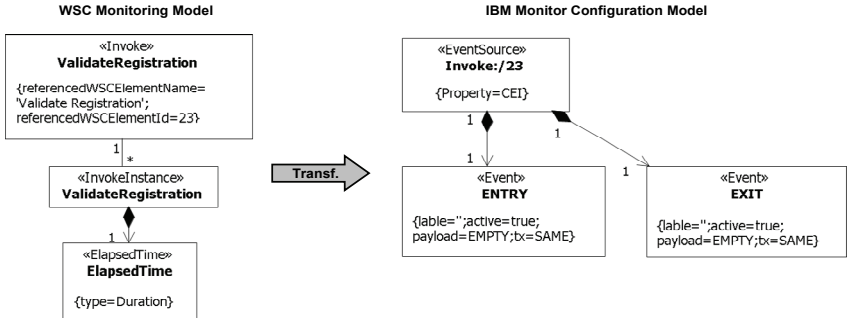

Fig. 6. Generation of the Process Server Instrumentation by Example

3 Evidence the Solution works

To yield benefits from our approach, an efficient tool support is crucial. The WSC monitoring developer on the one hand requires a comfortable editor in terms of a user frontend to create custom platform-independent WSC monitoring models. On the other hand, a transformation that automatically generates the platform-specific backend code has to be provided. **Fig. 7** summarizes our current prototypic implementation of the tool support.

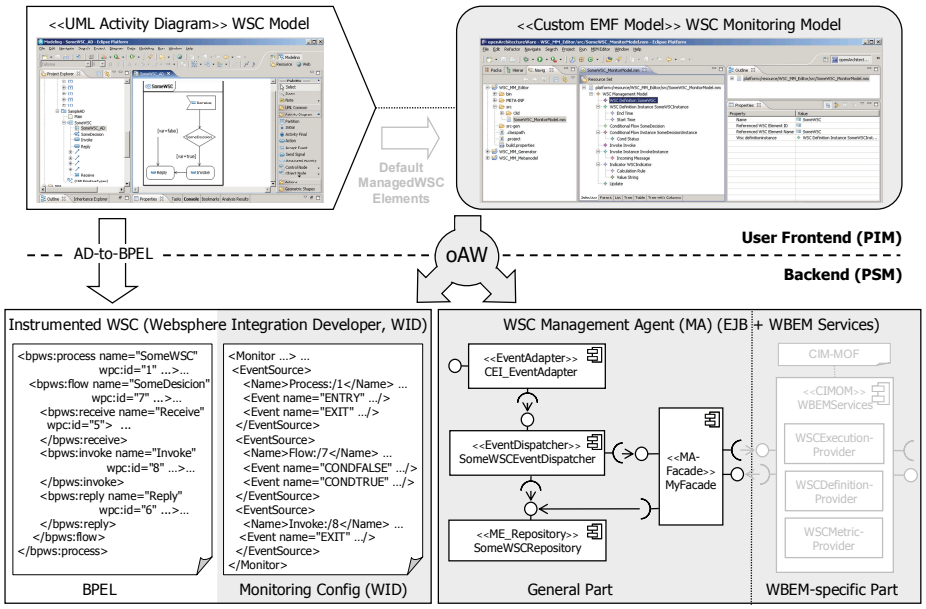

Fig. 7. Overview to Prototypic Implementation of the Approach

Regarding the user frontend, we decided on the Eclipse Modeling Framework (EMF) to provide a custom ECore model along with a simple graphical editor that allows the developer to create the complete monitoring model. Additionally, we are working on a transformation that automatically generates an initial monitoring model from the existing functional WSC model based on UML activity diagrams (AD). A feasible alternative to EMF to this would have been the creation of a UML profile. But as we do not consider a standard UML diagram as a suitable graphical representation and we do not want to confront the WSC monitoring developer with the inherited complexity of UML, we decided against this option.

Having developed the functional WSC model along with the corresponding WSC monitoring model, a transformation is executed that automatically generates the corresponding monitored WSC component. For creating the functional WSC component, which is comprised of the BPEL definition along with a matching WSDL, an existing AD-to-BPEL transformation is used. Unfortunately, the generated code is not fully executable yet and therefore has to be manually completed. In our case, we used the WebSphere Integration Developer (WID) to create the executable WSC. This aspect is not further regarded. In case of the monitoring, a custom transformation based on openArchitectureWare (oAW) is used to generate all additionally required monitoring artifacts and components. This, on the one hand, is the WID monitoring configuration, which basically specifies all the events necessary for providing the management information defined in the WSC monitoring model. Thus, for each kind of managed element property a corresponding transformation that generates the required events is available.

On the other hand, the management agent that eventually provides the specified management information is generated. To support different management standards like Web-based Enterprise Management (WBEM) or Java Management Extensions (JMX), we first create generally required components on basis of Enterprise Java Beans (EJB) using oAW xPand templates. A specific EventAdapter is used to receive management events delivered by the Common Event Infrastructure (CEI) and transform them into an internal representation. An EventDispatcher correlates the received events with the associated ManagedWSCElement instance. More precisely, the property values are updated or a new instance is created ME_Repository, which is comprised of an Entity Bean for each ManagedWSCElement. In addition, a corresponding indication is triggered, like “ME updated”. All information – managed element instances along with the associated indications - is made accessible through the MA-Façade. To support a particular management standard, an additional specific façade component is required for translating the information into the specific data formats or protocols. As indicated on **Fig. 7**, in case of the WBEM standard this would be a CIM Object Manager (CIMOM) configured with the WSC Monitoring Model expressed in terms of the Common Information Model (CIM). The mapping between the objects and indication provided by the general façade and CIM would be accomplished by appropriate CIM provider components. The generation of WBEM-specific components is planned. Here, we will reuse the design of our WSC manageability infrastructure [13] to a large extend. Particularly a CIM-based WSC information model is already available, which in future will be generated automatically from the WSC monitoring model.

For implementing the transformation that generates the instrumentation we employed the language xTend as part of oAW. It is a functional programming language particularly designed for creating model-to-model transformation on basis of corresponding metamodels. To handle metamodels defined by an XSD another oAW tool is used. The following code snippet represents the implementation of the example shown in **Fig. 4**. It defines the mapping from an invoke element of the WSC monitoring model to the IBM monitor configuration.

```
create MonitorType transform(MIM::WSC_ManagementModel sourceModel,
 String projectName):
     eventSource.addAll(sourceModel.wsc managedelement.
      typeSelect(MIM::Invoke).createEventSource() );

create EventSourceType createEventSource(MIM::Invoke me):
     setProperty( "CEI" ) ->
     setName( "Invoke:/"+me.referencedWSCElementID ) ->
     me.instance.instanceId == null ? "" : event.add(
      createEvent("ENTRY") ) ->
     me.instance.starttime == null ? "" : event.add(
      createEvent("ENTRY") ) ->
     me.instance.endtime == null ? "" : event.add(
      createEvent("EXIT") ) ->
     me.instance.loopcount == null ? "" : event.add(
      createEvent("ENTRY") ) ->
     me.instance.elapsedtime == null ? "" : event.add(
      createEvent("ENTRY") ) ->
     me.instance.elapsedtime == null ? "" : event.add(
      createEvent("EXIT") );
```

The whole transformation process for the instrumentation starts with calling the function transform. Here, at first a new root element of an IBM monitor configuration is created. Afterwards, all available managed element types are traversed. If they are defined in the WSC monitoring model, a corresponding create function is called, which generates the corresponding snipped of the monitoring configuration. In this example, only managed elements of type Invoke are selected and the createEventSource function is called. This function now creates an EventSource element for the IBM monitor configuration and sets the name that references the functional model element. Afterwards, the different possible properties of the regarded managed element are checked. If a property is set in the WSC monitoring model, the required event definitions are created. The ElapsedTime property in the sample model (see **Fig. 4**) for instance requires two events: ENTRY and EXIT.

The oAW transformation is nested in an oAW workflow file. Here, different runtime parameters can be configured, like filenames and paths to metamodels. It can be easily integrated in other software components, like for instance eclipse plug-ins, to achieve a productive tool support.

3 Competitive Approaches

In [6] an approach is presented that promotes an integration of Quality of Service (QoS) concerns into a model-driven development process for component-based

applications. This particularly includes an automated generation of a CIM-based QoS monitoring infrastructure and component instrumentation. The approach is promising but has to be adapted to the specifics of WSC, particularly regarding the monitoring model and the instrumentation.

[18] addresses the model-driven specification of SLAs as an activity that is independent from the functional design. This approach includes the definition of SLA parameters along with the required management metrics/indicators and the rules for calculating them. It is assumed that there already is a management infrastructure delivering the required (elementary) metrics. The instrumentation issue is not further addressed. Thus, our approach should be considered as supplementary to this. However, business performance monitoring requirements are not regarded at all.

In a very similar way, [19] introduces an approach to the model-driven specification of business performance management issues and the automated transformation to executable models is presented. The approach is part of a model-driven business transformation (MDBT) toolkit that supports the specification of monitoring requirements and the automated generation of platform-specific monitored solutions. An overview to this very competitive approach is given in [4], where the MDBT is applied to service delivery management (SDM). In contrast to our solution, the necessary WSC instrumentation including the available WSC-specific measurements is not considered. Furthermore, the ability to integrate into arbitrary existing development and management environments is not in focus.

4 Current Status and Next Steps

In this paper we presented an approach to the integrated design and implementation of monitored WSC. The WSC monitoring metamodel thereby allows creating a management view for an arbitrary functional WSC metamodel on a platform-independent level of abstraction, while still providing explicit associations to the respective functional elements. In this way, coherency of the overall solution may be checked on basis of the design models. As the metamodel is domain-specific for a WSC management, the complexity of modeling the required management information is reduced. The same holds for the provided transformation that automatically generates the monitored WSC. It reduces the complexity significantly and coherency between the design models and the implementation is always ensured.

Regarding the next steps, we will address the specification of indicators, which so far is not executable yet. We rather use the Object Constraint Language (OCL) for a declarative definition. Thus, we are working on an extension of the metamodel that not only allows an executable specification of indicators but also the definition of reusable templates. Based on this extension, we will provide a corresponding transformation to a management infrastructure. Finally, we are working on case studies that demonstrate the application of our approach by means of concrete scenarios – one from the field of business performance management and one concerned with SLA-driven service management. In this context, we also target the development of a more comfortable graphical editor based on the Graphical Modeling Framework (GMF).

References

1. Arsanjani, A., Liang-Jie, Z., Ellis, M., Allam, A., Channabasavaiah, K.: S3: A Service-Oriented Reference Architecture. IT Professional **9** (2007) 10-17
2. Papazoglou, M.P., van den Heuvel, W.J.: Service-Oriented Design and Development Methodology. Int. J. of Web Engineering and Technology (IJWET) (2006)
3. Esfandiari, B., Tosic, V.: Towards a Web service composition management framework. IEEE International Conference on Web Services (ICWS 2005) (2005) 426
4. Kumaran, S., Bishop, P., Chao, T., Dhoolia, P., Jain, P., Jaluka, R., Ludwig, H., Moyer, A., Nigam, A.: Using a model-driven transformational approach and service-oriented architecture for service delivery management. IBM Systems Journal **46** (2007) 514
5. Momm, C., Malec, R., Abeck, S.: Towards a Model-driven Development of Monitored Processes. 8. Internationale Tagung Wirtschaftsinformatik (WI2007), Karlsruhe, Germany (2007)
6. Chan, K., Poernomo, I.: QoS-aware model driven architecture through the UML and CIM. Information Systems Frontiers **9** (2007) 209-224
7. McGregor, C., Schiefer, J.: A Web-Service based framework for analyzing and measuring business performance. Information Systems and E-Business Management **2** (2004) 89-110
8. Keller, A., Ludwig, H.: The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. Journal of Network and Systems Management **11** (2003) 57-81
9. Stahl, T., Voelter, M., Czarnecki, K.: Model-Driven Software Development: Technology, Engineering, Management. John Wiley & Sons (2006)
10. Johnson, S.K., Brown, A.W.: A Model-Driven Development Approach to Creating Service-Oriented Solutions. Int. Conference on Service-Oriented Computing (ICSOC 06) (2006) 624–636
11. Roser, S., Bauer, B., Muller, J.P.: Model-and Architecture-Driven Development in the Context of Cross-Enterprise Business Process Engineering. IEEE International Conference on Services Computing (SCC'06) (2006) 119-126
12. Melchert, F., Winter, R., Klesse, M.: Aligning Process Automation and Business Intelligence to Support Corporate Performance Management. Tenth Americas Conference on Information Systems, New York (2004) 4053-4063
13. Momm, C., Mayerl, C., Rathfelder, C., Abeck, S.: A Manageability Infrastructure for the Monitoring of Web Service Compositions. 14th HP-SUA Workshop, Munich, Germany (2007)
14. Farrell, J.A., Kreger, H.: Web services management approaches. IBM Systems Journal **41** (2002) 212-227
15. Sahai, A., Machiraju, V., Sayal, M., van Moorsel, A., Casati, F.: Automated SLA Monitoring for Web Services. 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2002), Vol. 2506. Springer-Verlag, Montreal, Canada (2002) 28-41
16. Emig, C., Krutz, K., Link, S., Momm, C., Abeck, S.: Model-Driven Development of SOA Services. Universität Karlsruhe (TH), Karlsruhe (2007)
17. Moore, B., Bader, M., Liu, J., Mincov, R., Patil, V.: WebSphere Business Integration Server Foundation: Using the programming API and the Common Event Infrastructure. IBM Redbook. IBM (2004)
18. Debusmann, M., Kroger, R., Geihs, K.: Unifying service level management using an MDA-based approach. IEEE/IFIP Symposium on Network Operations and Management (NOMS 2004), Vol. 1 (2004) 801-814 Vol.801
19. Chowdhary, P., Bhaskaran, K., Caswell, N.S., Chang, H., Chao, T., Chen, S.K., Dikun, M., Lei, H., Jeng, J.J., Kapoor, S.: Model Driven Development for Business Performance Management. IBM Systems Journal **45** (2006) 587