A Model-Driven Approach for Monitoring Business Performance in Web Service Compositions

Christof Momm
Software Engineering
FZI Research Center for Information Technology
Karlsruhe, Germany
momm@fzi.de

Michael Gebhart, Sebastian Abeck
Research Group Cooperation & Management
Universität Karlsruhe (TH),
Karlsruhe, Germany
{gebhart | abeck}@cm-tm.uka.de

Abstract— **Supporting business services through Web service compositions (WSC) as part of service-oriented architectures (SOA) involves business performance monitoring requirements. Their implementation results in additional development activities. To support these activities, we already contributed a model-driven approach to the development of monitored WSC as part of our preliminary work. In this paper, we present an extension to this approach, which focuses on supporting the specification and transformation of indicators to an executable implementation. To reduce development effort for this particular task, we provide a template-based mechanism for defining performance indicators. In combination with our preliminary work, now fully monitored WSC can be generated automatically from platform-independent design models. We demonstrate the applicability of the overall approach by instantiating an integrated development process for a target platform based on IBM SOA products and showing its application for a sample business process along with monitoring requirements.**

Keywords- Web Service Compositions; Service-Oriented Architectures, Business Performance Monitoring; Model-Driven Software Development; Business Process Management

I. INTRODUCTION

Companies demand an IT support that on the one hand is strongly aligned with their business processes and on the other hand is highly adaptable in case of changing processes. For achieving this, the employment of Service-Oriented Architectures (SOA) is heavily promoted. Here, business processes are consequently realized through Web service compositions (WSC) on the SOA's business process layer [1], most commonly by using the Business Process Execution Language (BPEL) [2]. As one major benefit of this approach, it is now possible to directly monitor process or business performance within the uniform process implementation, even close to real-time. For this purpose, a variety of existing business (activity) monitoring tools can be employed, like for instance [3][4]. Nevertheless, this requires the development of additional artifacts and components. The specific monitoring tool has to be configured with very company-specific performance indicators, whose effective computation relies on (low-level) measurements about the WSC execution, like the runtime of single activities. This in turn requires a corresponding monitoring instrumentation of the WSC, for instance based on sensors. To create a monitoring that complies with the monitoring requirements and is consistent with the functional WSC, a systematic development approach is necessary that takes into account the monitoring requirements from the very beginning [5][6].

In current practice, the monitoring concerns are considered subsequent to the functional development by configuring specific monitoring tools. In this context, we identified several problems. (1) The employment of specific tools leads to solutions that are not portable. Migrating to another tool or framework would simply be too complex and costly. (2) The subsequent and isolated treatment of management issues and the generic nature of existing management solutions increase complexity and the risk of inconsistencies. Usually, the management of arbitrary resources is supported and thus the tools necessarily abstract from concrete instrumentation code. This code however forms the bridge between the functional and the management implementation. Without regarding the instrumentation at design time, it is hard to trace the impact of changes in the functional or the management implementation. Furthermore, when focusing on WSC the generic monitoring models result in redundant and error-prone modeling activities. (3) Regarding the specification of indicators along with their underlying calculation rule, the currently available monitoring models do not support the definition of reusable templates. Each indicator has to be defined in its entirety, even if the calculation rule only differs slightly from already modeled rules. This additionally results in unnecessary complexity.

To overcome the first two drawbacks, we already contributed a model-driven approach to the development of monitored WSC [7][8]. However, this solution so far is limited to the generation of monitorable WSC, which offer structured management information required for calculating indicators through a manageability interface. In this paper, we present an extension to this approach, which additionally supports the specification and transformation of indicators to an executable implementation, resulting in a fully monitored WSC. For this purpose, we contribute complementary metamodels for defining indicators and their underlying calculation rules on top of basic WSC management information. In this context, we observed that these calculation rules may be reduced to several basic calculation patterns or functions, like for instance the duration as the difference between a start and end time of an activity. In

existing solutions however, each calculation rule has to be fully defined for each indicator on basis of concrete management information, for instance represented by concrete business events or concrete managed elements. A reuse of reoccurring calculation patterns is only supported through a complex and error-prone copy and paste. To this end, we present metamodels allowing a specification of reusable calculation templates, which can later on be applied to multiple concrete indicators.

As proof of concept, we demonstrate the whole approach by means of a scenario we initially developed in cooperation with IBM Business Services. We instantiate the integrated development process for a target platform based on IBM SOA products and show its application for a sample business process along with monitoring requirements. The presented solution focuses on the definition of instance indicators, which refer to single process instances. However, it can easily be extended for the definition of aggregated indicators by introducing a new type of calculation template.

II. MOTIVATING EXAMPLE

To motivate our contribution and to exemplify our solutions, we first introduce a simplified real-life scenario, namely IBM's SOA showcase Panta Rhei [9]. Panta Rhei is a traditional watch manufacturer that wants to extend its current portfolio by an innovative fitness training service based on a fitness watch. More precisely, the measurements collected by the watch (blood pressure, body temperature and heart rate) are combined with a health check service offered by a second company (Telehealthcare). The resulting training service basically helps the customer to continuously improve her training schedule based on the measurements and the medical feedback created by professional health personnel. Figure 1 shows the process Update Training Schedule as a part of the complete scenario.

This process is executed, if the customer decides to get a new training schedule. Once a new training schedule has been created or an existing one has been updated, the customer has to accept it. If she does not accept the schedule she can write a comment and the trainer has to create a reviewed training schedule. As soon as the customer agrees the new training schedule is stored.

Regarding the monitoring requirements, several instance indicators are of interest: (1) The duration of the complete process, (2) the duration to create a new or update an existing training schedule, (3) the cost for creating a new or update a training schedule, (4) the cost to analyze a new training schedule. The cost is defined as multiplying a constant by the duration.

Here, some similarities of the indicators can be identified. The calculation rules of indicator (1) and indicator (2) are both based on the duration pattern that can be described as: *Activity.EndTime – Activity.StartTime*. The calculation rules of indicator (3) and indicator (4) both can be described as: *Duration * Constant*. The constant in this calculation rule has to be replaced with the cost per time unit. Thus, using templates avoids a redundant definition of calculation rules.

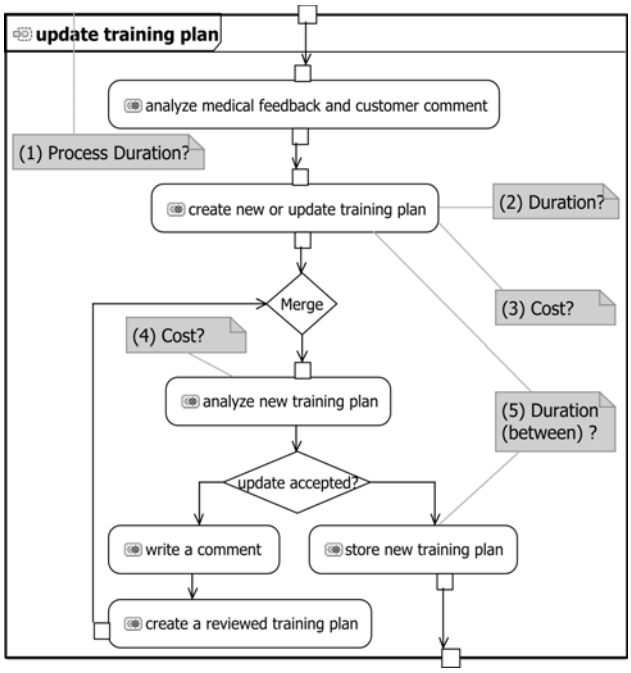

Figure 1. Sample UML 2 activity diagram

III. OVERVIEW TO THE OVERALL APPROACH

This section provides a brief overview to the overall approach, which represents a variation of preliminary results published in [8].

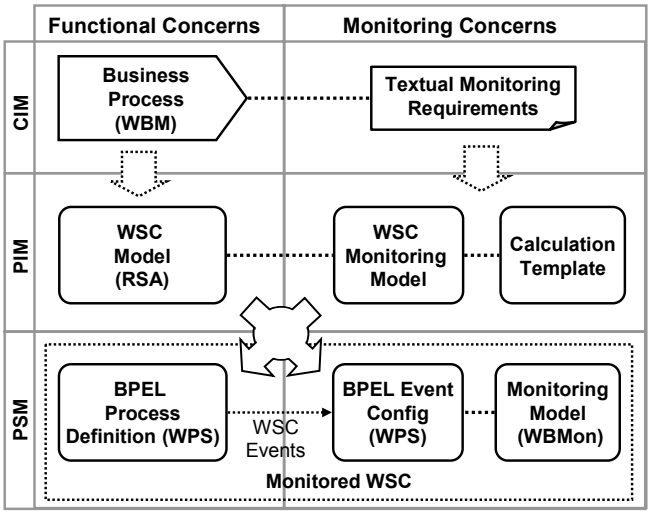

Figure 2. Overview to model-driven approach

As shown in Figure 2, we generally complement an existing model-driven approach for developing WSC, as for instance presented in [10], by monitoring concerns through introducing additional metamodels and transformations. In this paper, we focus on a model-driven development process as proposed by IBM [11][12]. For implementing the Panta Rhei scenario, the IBM Websphere Business Modeler (WBM) is used to design business processes in a

computation-independent way. These models are imported in Rational Software Architect in terms of UML activity diagrams (ADs) and afterwards refined into platform-independent, executable WSC models. Eventually, BPEL process definition are generated via a UML-to-SOA transformation [13], imported into the Websphere Integration Developer (WID) and manually completed.

On each level of abstraction the functional models are complemented by corresponding monitoring models. On the CIM level, we propose to capture the monitoring requirements in terms of informal, textual specifications. These specifications are then manually transformed into a platform-independent WSC monitoring model. This metamodel abstracts from specific composition engines as well as management tools and allows the specification of (instance) indicators including the operational semantics for calculating them. In this context, we introduce a separate metamodel on the PIM level that supports the specification of reusable calculation templates. In doing so, calculation rules are defined on basis of placeholders instead of concrete values. Each calculation rule eventually refers to runtime information about the executed WSC. Thus, the monitoring metamodel comprises all available WSC runtime measurements in terms of managed elements. For each functional element in the WSC metamodel, a corresponding managed element is available in the monitoring metamodel containing the management-relevant information in terms of properties.

In this paper, we develop a transformation that automatically generates an effective monitoring implementation from these platform-independent monitoring models. Here, the WebSphere Business Monitor (WBMon) is used as a specific monitoring tool. Thus, our transformation on the one hand creates a corresponding Monitor Details Model (MDM), which is deployed and executed on the WBMon. Moreover, an event configuration for the WebSphere Process Server (WPS) is generated. In doing so, the runtime measurements required for calculating the specified indicators are delivered to the WBMon through the Common Event Infrastructure (CEI).

IV. Platform-Independent Monitoring Metamodels

This section introduces the different metamodels required for our approach. In summary this is (1) a metamodel for specifying the indicators and the managed elements that represent the WSC model elements as a basis for the WSC monitoring model. (2) A metamodel for defining calculation templates, for instance for calculating durations or costs. (3) A dedicated metamodel for invocation of calculation templates to simplify the application of calculation templates. The template signature metamodel basically corresponds to a function signature and is generated automatically from the template models. Within instances of this metamodel, namely the template invocation models, the placeholders are replaced with concrete elements of a WSC monitoring model. Figure 3 provides an overview to the metamodel architecture required for this approach.

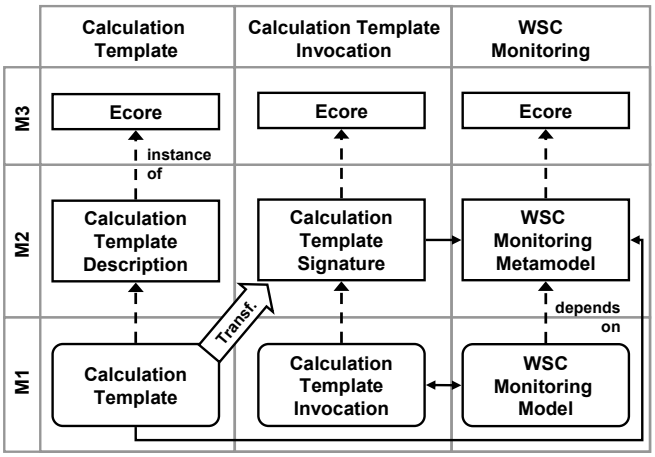

Figure 3. Overall metamodel architecture

Following the Model-Driven Architecture (MDA) [14], the custom metamodels are generally defined on level M2. In contrast to the MDA we use *Ecore* as the meta-metamodel on level M3. The WSC monitoring part fully complies with this concept. In case of the calculation templates however the template models are created on the M1 level, but refer to elements of the WSC monitoring metamodel on the M2 level. Consequently, the derived template signature represents a metamodel on the M2 level. In this way, the referenced placeholder elements on the M2 level can now be instantiated with concrete properties of a WSC monitoring model on the M1 level. To clarify this approach, in the following we first briefly introduce the required metamodels and afterwards demonstrate their application in case of our sample scenario.

A. Required Metamodels

As aforementioned, the WSC monitoring metamodel as already presented in [8] includes a management abstraction of WSC using the concept of managed elements. Figure 4 illustrates the basic structure of the metamodel, which includes managed elements (ME) for describing a WSC as a whole as well as the different internal WSC elements, e.g., a single activity or a decision node. In each case, a complementary pair of managed elements for modeling the management view is introduced. More precisely, we always offer a managed element reflecting information about each executed WSC instance (*WSC_ME_Instance*) by means of properties and one that holds information related to the general definition of the WSC, like static configuration settings that are already available at design time (*WSC_ME_Definition*) [15]. This particularly includes the references to the functional model. Regarding the (runtime) properties, in case of activities four basic properties can be identified. *StartTime*, *EndTime* and *ElapsedTime* are used for time-based monitoring, while *LoopCount* is used to monitor the control flow of loops. Note that the complete metamodel includes more managed elements and properties than this.

Instances of the managed element meta classes may partly be generated automatically from a (functional) WSC model. Specific instance indicators however always have to

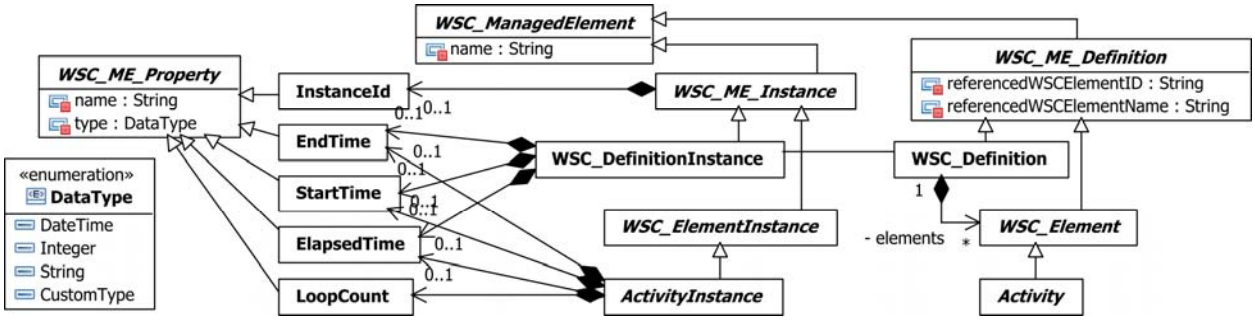

Figure 4. WSC monitoring metamodel – managed elements (excerpt)

be created manually. This is supported by the meta classes presented in Figure 5. Accordingly, an (instance) *Indicator* operates on *WSC_ME_Properties*. Its calculation is triggered through an *UpdateRule*, which is activated in case a certain *Indication* (i.e. event) arrives. So far an *Init* or *Update* indication for signaling changes of property values may be defined. The calculation rule for an indicator is generally defined through a reusable calculation template or its signature respectively. So the indicator only holds a reference to the applied template.

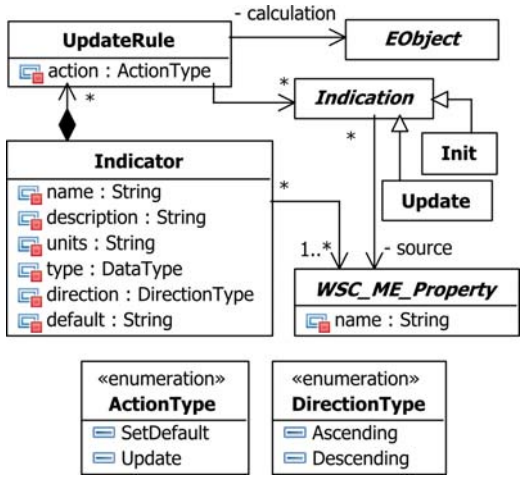

Figure 5. WSC monitoring metamodel - indicator specification

The structure of such a calculation template is defined through the calculation template metamodel as shown in Figure 6. The metamodel so far supports the definition of arithmetic expressions. These are defined by creating a tree of *Calculation* elements, where leafs may be either a *Constant* or a *ReferencedValue* holding pointers to a particular *WSC_ManagedElement* as well as *WSC_ME_Property* on the M2 level. So *ReferencedValue* represents the essential meta class for realizing the template mechanism as it acts as a placeholder within the calculation template. When applying the template, these placeholders are replaced by the concrete elements and properties available within a WSC monitoring model on the M1 level.The actual calculation rule is specified by nesting different types of operations. *UnaryOperation* is used for defining a unary operation on another single *Calculation* element, whereas *BinaryOperation* performs a binary operation on two input elements that are specified through the associations *operand1* and *operand2*. To convert a value from a specific data type to another a *ConvertedCalculation* can be used. All available unary operation types, binary operation types, data types and conversion types are provided by means of enumerations.

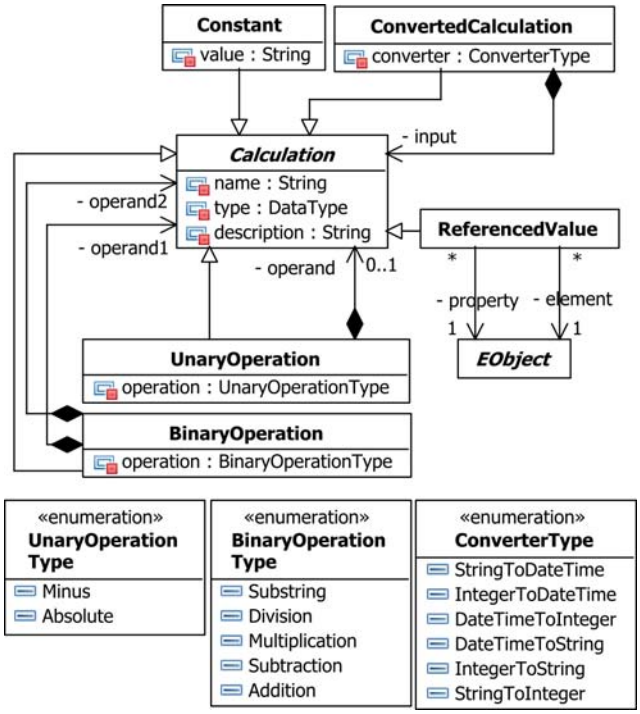

Figure 6. Calculation template metamodel

B. *Specification of Indicators based on Reusable Calculation Templates*

In this section we demonstrate the application of the previously introduced metamodels by means of the scenario introduced in Section 2. As already pointed out, each calculation rule for an indicator is defined through a calculation template. Thus, the monitoring developer first creates a calculation template model on the meta level M1 as shown in Figure 7.

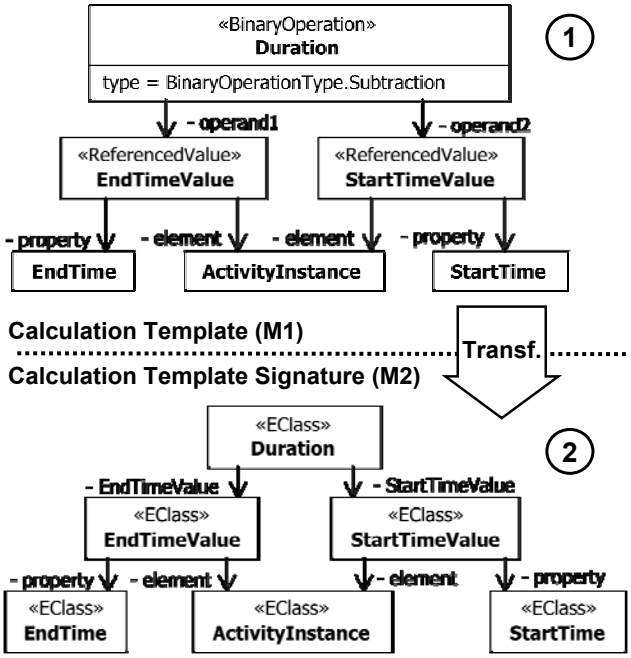

Figure 7. Calculation template and template signature for the duration

This example comprises a rather simple but generally required template for calculating durations of activities as the difference between two points in time represented in milliseconds. To this end, a binary operation of type *Subtraction* is defined with *operand1* set to the placeholder *EndTimeValue* and *operand2* referencing a *StartTimeValue*. Both *ReferencdValue* elements refer to an *EObject* of type *ActivityInstance* and a *WSC_ME_Property* of type *EndTime* or *StartTime* respectively. Note that these placeholders have to exist in the WSC monitoring metamodel. The completed calculation template is published in a template repository and may be used for specifying indicators. When applying a calculation template to an indicator however, the developer is only interested in two things about the template: (1) The name of the template and (2) the placeholders that have to be replaced. Thus the calculation template model is transformed into a calculation template signature (Figure 7, right-hand side). This signature only contains the name of the template and the placeholders. To apply a calculation template model the user instantiates the corresponding calculation template signature model. Hence, the generated calculation template signature in fact represents a metamodel on the M2 level. Consequently, to combine an indicator with a calculation template, the corresponding signature metamodel has to be instantiated. In case of our example, the generated signature only slightly differs from the calculation template, as the related calculation rule is rather simple.

Figure 8 now illustrates the application of this signature for defining an instance indicator identified in Section 2, namely the duration from receiving a new or updated training and storing the final training plan, which may involve several iterations.

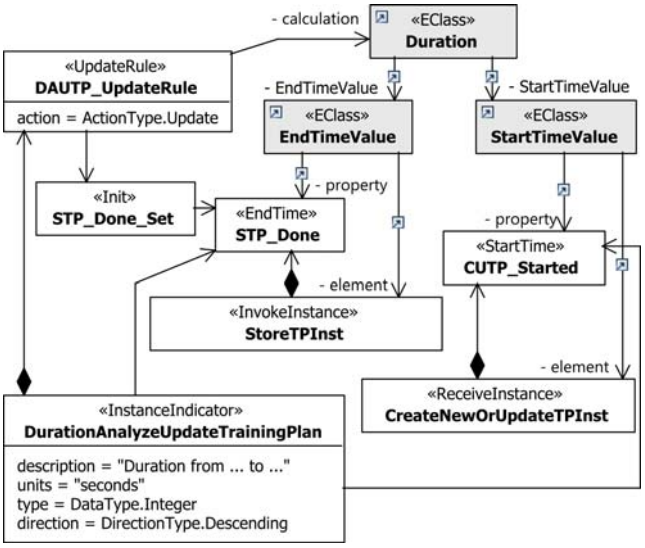

Figure 8. Complete specification of indicators

The required indicator is specified within the WSC monitoring model. For calculating its value, runtime information about two activities are required, namely an *EndTime* and a *StartTime*. Thus, for both activities the corresponding managed elements along with the necessary properties are created and referenced by the indicator. As defined by the *UpdateRule* the indicator is calculated in case the *EndTime* is set for the first time. At this point, the previously created calculation template is used. Hence, a template invocation model is instantiated on basis of the signature metamodel. Here, the placeholders are replaced by the properties of the WSC monitoring model.

V. TRANSFORMATION TO PLATFORM-SPECIFIC MODELS

This section focuses on the design of a transformation to a specific target platform that automatically generates a fully executable monitoring implementation on basis of the previously created, platform-independent monitoring models. Note that to transform a calculation rule, both a valid calculation template model and the corresponding calculation template invocation model is required. This is because only the template contains the actual calculation rules for the indicators within the WSC monitoring model, whereas the calculation template invocation models only maps the actual properties to the defined placeholders. To demonstrate the approach we focus on the IBM SOA product portfolio as one specific platform. Here, the execution of WSC as well as their monitoring is already supported in an integrated way. The different engines are able to communicate with each other without having to create further adapters. Nevertheless, as shown in Figure 9, still some IBM-specific artifacts have to be generated.

This is (1) The Monitor Details Model (MDM) for monitoring information concerning one process instance and (2) a WPS Event Configuration for defining the corresponding WSC instrumentation based on sensors. In the following we focus on the generation of the MDM, whereas

further information on the transformation required for WSC instrumentation can be found in [7].

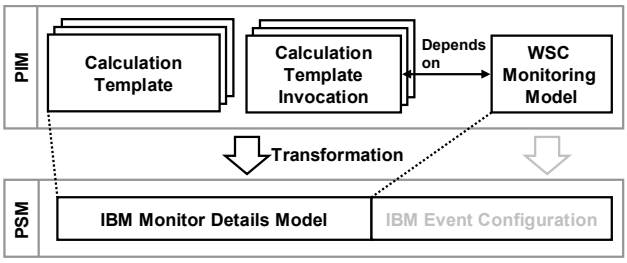

Figure 9. Transformation from PIM to PSM

In the following, all the transformations required for generating the target MDM from our platform-independent models are described. The key element of a MDM represents a *MontoringContext*, which is used to model a monitoring abstraction of a "real-world" object, in our case a WSC. Such a context is instantiated through an *InboundEvent* and comprises several *Metrics*. The calculation of a *Metric* or *Counter* is triggered through a dedicated *InboundEvent* or a *Trigger*. Further details on this model can be found in [16]. Figure 10 illustrates how WSC monitoring models are transformed into this structure.

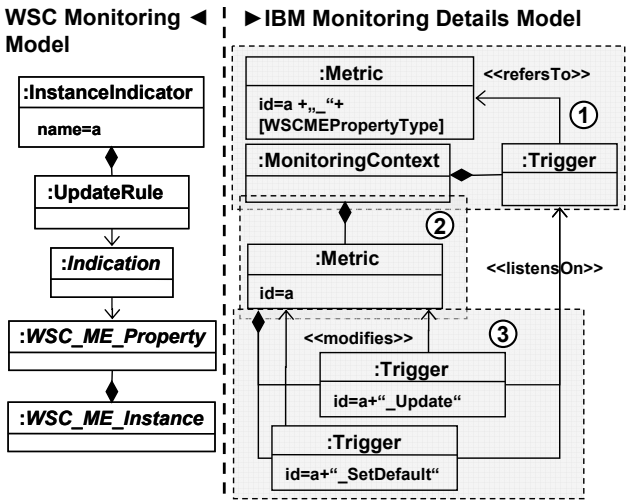

Figure 10. Transformation to MDM

For each WSC monitoring model, which always refers to a single WSC, a root element of the MDM along with a single Monitor Context is created. In a next step all instance elements (*WSC_ME_Instance*) along with their properties are transformed into Metrics within the MDM, whereas the associated indications are mapped to *Triggers*. In a next step, all the indicators are mapped to further *Metrics* within the same *MontioringContext*. Afterwards, additional *Triggers* are generated for each *UpdateRule*. One *Trigger* to set the current value of the indicator to the default value and one to set it to the value specified by its calculation rule. Both triggers on the one hand listen on the triggers that have been previously created for the corresponding indication. On the other hand they are associated with the corresponding indicator *Metric*, which now is updated once a property change is detected through the indication trigger. In this case, the *Trigger* executes the specified calculation rule as specified within a calculation template model.

The generation of this *Map* works as follows. It starts with the element *Calculation* referenced by an *Indicator*, which is transformed into an expression within a *Metric Value Map* for the already transformed indicator. A *Binary Operation* or *Unary Operation* is mapped to a corresponding arithmetic operation or a function and a *Converted Calculation* is transformed to a function representing the chosen conversion. A *Constant* on the other hand can be mapped without changes, whereas a *Referenced Value* is transformed into the corresponding property *Metric*. Since a Referenced Value acts as a placeholder the name of the actual Metric has to be looked up within the corresponding template invocation model. As *Binary Operation*, *Unary Operation* and *Converted Calculation* require input parameters the transformation is continued in a recursive way.

VI. IMPLEMENTATION OF TOOL-SUPPORT

Figure 11 provides an overview to the tools we implemented for supporting the development of monitored WSC.

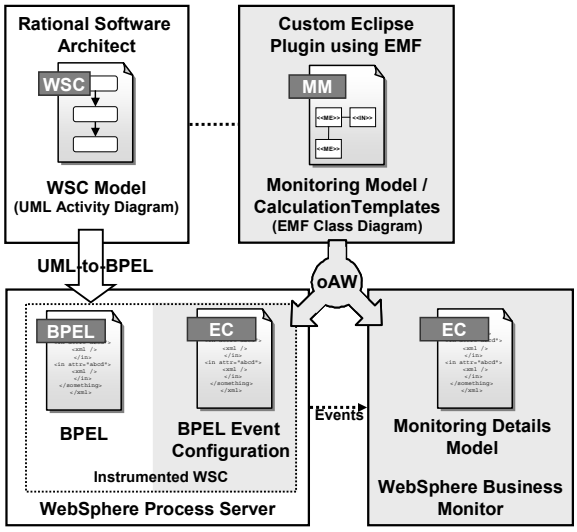

Figure 11. Overview to tool-support

To implement our approach we decided on the Eclipse Modeling Framework (EMF) for creating the metamodels along with corresponding model editors. The transformations for generating the calculation template signature and the platform specific monitoring models are implemented using the openArchitectureWare (oAW) framework, in particular the model-to-model language *Xtend*. Additionally we created an eclipse plugin that supports the creation of templates and WSC monitoring models and the convenient creation of calculation template invocations.

The Eclipse plugin is used for specifying and managing calculation templates and monitoring models. Here, templates can easily be applied to indicators by a Drag&Drop mechanism. Having created a complete monitoring model, the developer simply pushes a button that triggers the batch execution of the specific transformations by invoking an oAW workflow file.

Figure 12 shows an excerpt of the oAW-based transformation rules used for generating an MDM for a given indicator specification along with a referenced calculation template. The way this implementation works corresponds to the explanations we provided in the previous section (see Figure 11). All highlighted sections in the code mark the transformation rules we described there. Only the generation of the maps that contain the calculation rule themselves is not included in this excerpt. This requires an additional rule called *createCalculationExpression*., which is invoked in a recursive manner to process the operator trees specified in the calculation template.

```
create monitor::MetricType this addIndicator
  (InstanceIndicator indicator, WSC_Management_Model model,
    monitor::MonitoringContextType mc,String templatePath):
  […] //init variables
//Set indicator properties
  this.setId(indicator.name) ->
  this.setDisplayName(this.id) ->
  this.setDescription(indicator.description +
    " (" + indicator.units + ")") ->
  this.setDefaultValue(createDefaultValue
   (indicator.^default)) ->
  this.setType(createDataType(indicator.type)) ->
//Create trigger for UpdateRule
  mc.trigger.add(updateTrigger) ->
  mc.trigger.add(setDefaultTrigger) ->
  updateTrigger.setId(this.id + "_Update") ->
  setDefaultTrigger.setId(this.id + "_SetDefault") ->
  […] //Set trigger properties
//Assign indication trigger
  updateTrigger.onTrigger.addAll
    (indicator.updateRules.select(e|e.action.toString()==
    "update").transformUpdateRule(model)) ->
//Assign indication trigger
  setDefaultTrigger.onTrigger.addAll
    (indicator.updateRules.select(e|e.action.toString()==
    "setDefault").transformUpdateRule(model)) ->
//Add calculation rule
  this.map.add(updateMap) ->
  […] //Intialisierung der Maps
  setDefaultMap.setOutputValue(createDefaultValue
   (indicator.^default)) ->
  updateMap.setOutputValue(createCalculationExpression
   (indicator, this, templatePath));
```

Figure 12. oAW-based indicator transformation (excerpt)

The execution of these rules results in an executable Monitoring Details Model. Besides this there are rules for generating the corresponding Event Configuration (WSC instrumentation) as presented in [7]. Both code artifacts are ready to be deployed on a WPS or WBMon respectively. Thus, the plugin along with the built-in transformation hides the complexity of the specific target platform as well as the introduced metamodel architecture to the developer, who can now fully concentrate on the essential specifications.

VII. Related Work

In [17] 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. So far, only a limited and predefined set of QoS parameters (like response time, availability etc.) can be associated with the components' interfaces and transformed to a monitored solution. The approach does not allow for the specification and processing of custom indicators referring to a component's internal behaviour, which in case of WSC can be described with BPEL or an adequate model abstraction. Due to these limitations, this approach does not consider at all a template-based specification of indicators as we presented in this paper.

[18] focuses on 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. The provided metamodel allows to model arbitrary indicators along with the corresponding calculation rules by creating operator trees. Thus, it would be possible to implement our motivating example although the approach focuses on SLA monitoring. However, there is no mechanism provided supporting the reuse of (possibly complex) calculation rules across different SLA models. In this case, the developer has to fall back on a rather error-prone copy&paste. Moreover, the authors assume that there already is a management infrastructure delivering the required (elementary) metrics and therefore do not address the instrumentation required for the managed resources.

[19] presents a quite similar model-driven approach for business performance management, which also supports the specification of arbitrary performance indicators and their automated transformation to executable models. The basic structure of the provided metamodels thereby is similar to the MDM. So calculation rules are defined in terms of maps that operate on (basic) business events. Due to this focus on business performance, this solution suits well for implementing the motivating example. But again, it's not possible to reuse calculation rules across models without a copy&paste and the instrumentation of the managed resources are not considered at all.

Nevertheless, regarding the indicator specification both approaches we previously discussed are so far are more powerful, as they for instance support the definition of aggregated indictors. Thus, they should be considered as complementary to our approach and vice versa.

VIII. Conclusion and Future Work

In this paper we presented an approach to the integrated design and implementation of monitored WSC, which generally helps to ensure consistency by increasing traceability (a) between requirements and implementation and (b) between functional and monitoring models/implementation [5][6].

Compared to the related work, the unique feature of this approach represents the fact, that all components of a monitored WSC are considered on a platform-independent level of abstraction and a fully functional implementation is generated automatically. Besides the specification of indicators, this particularly includes an abstraction of the WSC instrumentation. The platform-independent monitoring metamodel already comprises a precise and focused management abstraction of the managed resource WSC in terms of specialized managed elements. So a WSC (monitoring) developer can focus on the definition of the required indicators on top of properties provided by these managed elements and does not have to cope with instrumentation issues any more. As a result, unnecessary technological details as well as complex structures of generic monitoring models are hidden to her. Platform details are added automatically by applying the corresponding transformation. This general approach also helps to increase portability of the solution. In case the platform (composition engines or management tool) is changed, only the transformations have to be adapted, while the platform-independent models are still valid.

The introduction of reusable templates, which so far is not supported by any monitoring tool we observed, leads to a further reduction of complexity for the developer. Especially the simple duration template we presented in this paper turned out to be highly reusable. In the same way, we could leverage templates that refer to costs, conditional branches and loops increase efficiency.

Regarding our future work, we plan an empirical study proving these. Such an evaluation is planned as part of the recently started EU-FP-7 project SLA@SOI (http://www.sla-at-soi.org). This project targets the development of an integrated SLA management framework for service-oriented applications on virtualized infrastructures. To ensure relevance of the results for different industrial domains, the framework will be evaluated within scope of various industrial use cases. In this project, we contribute a methodology and tool support for implementing SLA and business monitoring requirements for WSC. Because we are facing varying monitoring requirements as well as different stacks of technologies within the industrial use cases, this project provides ideal conditions for applying, enhancing and evaluating our approach. To this end, we are currently enhancing the available tool support and extending the approach by a support for aggregated indicators.

ACKNOWLEDGMENT

The research leading to these results is partially supported by the European Community's Seventh Framework Programme ([FP7/2001-2013]) under grant agreement n° 216556.

REFERENCES

[1] A. Arsanjani, Z. Liang-Jie, M. Ellis, A. Allam, and K. Channabasavaiah, "S3: A Service-Oriented Reference Architecture", *IT Professional,* vol. 9, pp. 10-17, 2007.

[2] OASIS, "Web Services Business Process Execution Language (WS-BPEL) Version 2.0". vol. 1.11.2008: OASIS, 2007.

[3] C. McGregor and J. Schiefer, "A Web-Service based framework for analyzing and measuring business performance", *Information Systems and E-Business Management,* vol. 2, pp. 89-110, 2004.

[4] J.-J. Jeng, 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)*, 2003, pp. 86-97.

[5] K. Chan and I. Poernomo, "QoS-aware model driven architecture through the UML and CIM", *Information Systems Frontiers,* vol. 9, pp. 209-224, 2007.

[6] R. Pignaton, J. I. Asensio, V. Villagra, and J. J. Berrocal, "Developing QoS-aware component-based applications using MDA principles", in *Eighth IEEE International Enterprise Distributed Object Computing Conference (EDOC 2004)* 2004, pp. 172-183.

[7] C. Momm, T. Detsch, and S. Abeck, "Model-Driven Instrumentation for Monitoring the Quality of Web Service Compositions", in *EDOC 2008 Workshop on Advances in Quality of Service Management (AQuSerM 08)*, Munich, Germany, 2008.

[8] C. Momm, T. Detsch, M. Gebhart, and S. Abeck, "Model-driven Development of Monitored Web Service Compositions", in *15th HP-SUA Workshop*, Marrakesh, Maroc, 2008.

[9] K. Langer, "SOA in Action", *IT-Director,* vol. 7-8/2007, pp. 18-21, 2007.

[10] S. Roser, B. Bauer, and J. P. Muller, "Model-and Architecture-Driven Development in the Context of Cross-Enterprise Business Process Engineering", in *IEEE International Conference on Services Computing (SCC'06)*, 2006, pp. 119-126.

[11] A. Brown, S. K. Johnston, and G. Larsen, "SOA Development Using the IBM Rational Software Development Platform: A Practical Guide", 2005.

[12] S. K. Johnston and A. W. Brown, "A Model-Driven Development Approach to Creating Service-Oriented Solutions", in *International Conference on Service-Oriented Computing (ICSOC 06)*, 2006, pp. 624–636.

[13] D. Gorelik, "Transformation to SOA: Part 3. UML to SOA": IBM, 2008.

[14] S. J. Mellor, *MDA Distilled: Principles of Model-Driven Architecture*: Addison-Wesley Professional, 2004.

[15] F. Barbon, P. Traverso, and M. Pistore, "Run-Time Monitoring of Instances and Classes of Web Service Compositions", in *IEEE International Conference on Web Services (ICWS'06)*, 2006, pp. 63-71.

[16] U. Wahli, V. Avula, H. Macleod, M. Saeed, and A. Vinther, "Business Process Management: Modeling through Monitoring Using WebSphere V6.0.2 Products (IBM Redbook)": IBM, 2007.

[17] K. Chan and I. Poernomo, "QoS-Aware Model Driven Architecture through the UML and CIM", in *10th IEEE International Enterprise Distributed Object Computing Conference (EDOC '06)* 2006, pp. 345-354.

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

[19] P. Chowdhary, K. Bhaskaran, N. S. Caswell, H. Chang, T. Chao, S. K. Chen, M. Dikun, H. Lei, J. J. Jeng, and S. Kapoor, "Model Driven Development for Business Performance Management", *IBM Systems Journal,* vol. 45, p. 587, 2006.