Towards a Manageability Infrastructure for a Management of Process-Based Service Compositions

Christof Momm, Christoph Rathfelder, Sebastian Abeck
Cooperation & Management, Institute of Telematics
Universität Karlsruhe (TH)
76128 Karlsruhe, Germany
{momm | crathfel | abeck}@cm-tm.uka.de

***Abstract*— The management of process-oriented service composition within a dynamic environment, where the employed core services are offered on service marketplaces and dynamically included into the composition on basis of Service Level Agreements (SLA), demands for a service management application that takes into account the specifics of process-oriented compositions and supports their automated provisioning. As a first step towards such an application, in this paper we introduce the conceptual design for an architecture and implementation of an interoperable and flexible manageability infrastructure offering comprehensive monitoring and control functionality for the management of service compositions. To achieve this, our approach is based on well-understood methodologies and standards from the area of application and web service management.**

Keywords-SOA; Serivce Management; Serivce Compositions; Manageabilty; Instrumentation;WBEM; CIM

I. INTRODUCTION

Today, companies demand for an 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) [1, 23]. Thereby, the basic functionality required for accomplishing the business processes is offered in terms of core (web) services. These web services are operated by a **service provider (SP)** and the terms of use are contractually fixed by means of a Service Level Agreements (SLAs). The core services are dynamically assembled to service compositions implementing fully automated and reusable parts of business processes [32].

The service compositions are operated by a separate provider **(composition service provider, CSP)** and the customer in turn negotiates the quality level implied by the corresponding business process on basis of SLAs. As proposed by [5] in future the core services along with the ressources required for their execution as well as the service compositions are offered on service marketplaces on basis of service offers. Thereby, most approaches assume that services will be contracted and bound on short notice, down to single service invocations that are traded [22, 24]. Within this scenario the composition service provider will face numerous challenges while pursuing his primary service management activities.

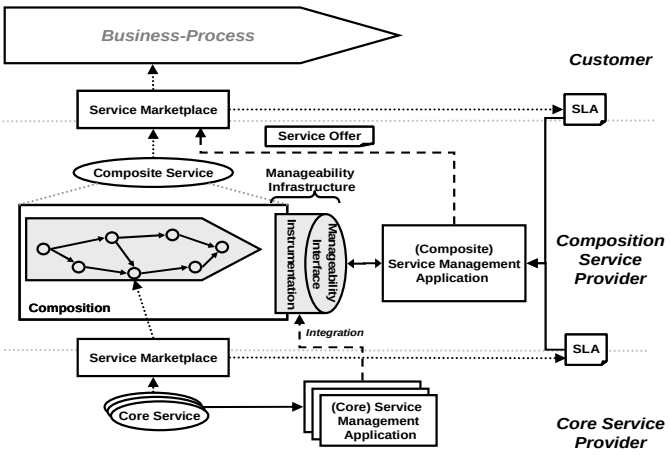

Figure 1. Initial Scenario

A. Service Composition Description & Offering:

The service description comprises a description of a service's functional and non-functional capabilities. The specified service quality may be either expressed as negotiable parameters [25] or fixed service levels [32]. The quality of service compositions generally relies on the usage profile, the quality levels offered by the employed core services and the performance of the execution environment. Hence, feasible service offers, which have to be continuously generated, depend on these influencing factors. A major challenge for the CSP will be the determination of service offers based on these variables [9, 31].

B. Core Service Selection:

Within the service selection the core services best suitable for providing the service composition request with respect to the offered quality are chosen. In this context, the CSP faces the challenge of continuously optimizing this selection, for instance concerning the costs, while meeting the constraints implied by the conducted SLA [15][38]. The assumption that for each service request a new selection is performed causes further difficulties.

C. Core Service Negotiation:

Within the sketched scenario the process of negotiating the qualitative service levels is performed by employing an adequate market mechanism as for instance presented in [22]. The challenge for the CSP will be the integration of these mechanisms into the IT supporting his service management processes. In alternative approach would be to enable a bilateral or multilteral negotiation of the service levels [8, 24].

D. Service Level Monitoring and Control for Service Compositions:

This service management activity on the one hand is concerned with the monitoring of the quality effectively achieved while performing the offered service compositions, the generation of SLA reports for the customer on basis of this monitoring information and the detailed analysis of detected SLA violations [28]. On the other hand, control mechanisms for a timely intervention in case of (predicted) SLA violations are incorporated. In the first case, one general challenge will be to extend the monitoring to the level of single service instances to enable an SLA reporting for the single service invocations sold on the service marketplace. Secondly, if an SLA violation has been detected, the CSP has to include SLA reports of the involved SPs in his analysis to determine, whether he or another SP has to take the responsibility. Thereby, single core service requests have to be correlated with single service composition requests. This requires for a cross-organizational information exchange [30]. Concerning the control mechanisms, the fact that single, stateless service requests are traded leads to the situation that it is not possible for the CSP to intervene during the execution. Hence, adequate mechanisms are required for predicting SLA violations [9]. In this way, an accurately timed reconfiguration is rendered possible.

To tackle these numerous challenges the CSP strongly relies on a on a management application for composite services which is tailored to meet the new requirements implied by this specific scenario [30]. According to the presented challenges we identified the following management functions such a management application should support:

(1) Service composition design (Specification of functional and non-functional capabilities along with internal process flow)

(2) Offer generation for service composition

(3) Determination of requirements concerning the core services and optimized selection at runtime

(4) Runtime generation and submission of quotes for required core services as well as conclusion of contracts

(5) (Runtime) reconfiguration of the service composition

(6) Continuous SLA compliance monitoring and generation of SLA reports for provided service requests

(7) SLA violation analysis

(8) SLA violation forecast

To facilitate these management functions an adequate **manageability interface (MI)** for the service compositions offering monitoring information and control functions is required in the first place.

In this paper we focus on the platform-independent conceptual design of a **manageability infrastructure (MIS)** required for providing this interface. As pointed, the design of the MIS architecture has to take into account that management information of the included core services has to be in integrated on instance level and an adquate **instrumentation** of the service compositions is avaliable.

In the following section the requirements for an MI that supports the specified management functions are pointed out. On basis of these requirements in chapter III we present a platform-independent management information model, which allows for expanding the management to the internal, process-based composition logic down to the level of single instances. In chapter IV we introduce a platform-independent architectural design of the MIS. In the follwoing chapters V and VI its main components are described in detail. In particular, we present a technical concept for integrating the core service management and an interoperal approach for instrumenting the service compositions. Having specified the platform-independent design of the MIS, in chapter VII we show how this design can be mapped to existing management platforms and standards in order to ease the integration with the core services management and to reduce the implementation effort. For this purpose, we propose to employ the Web-Based Enterprise Management (WBEM) standards [34]. Finally, the major related work in the context of a SLA-based web serivce managent is discussed. Thereby, the focus is set on the MIS design within these approaches.

For our approach the following limitations and assumptions exist. We assume that the service composition implement fully automatable parts of business processes without human interactions. The negotiation support (3) and the service composition design (1) are excluded as we do not consider these aspects a part of the MIS. Concerning the offered quality or SLA parameters we focus on the response time in the first instance. Or to put it in a more general way, we focus on a SLA-aware performance management.

II. Requirements for the Manageability Interface and the Manageability Infrastructure

To allow for a continuous SLA compliance monitoring and the generation of SLA Reports (6) the MI has to provide the CSP with information about all available service compositions and for each executed instance the actual response time has to be made available.

As already pointed up, for the analysis of detected SLA violations (7) management information of the employed core services has to be integrated. In particular, the MIS has to gather information about the response time of each core service instance executed within the corresponding service composition instance. As the SP by default does not know in which particular service composition his core service was used and least of all which particular instance of the service composition called his service, these kind of meta information have to be provided within scope of each of the CSP's requests.

This calls for an extension of the functional composition model. Concretely, each activity which handles the service invocation (i.e. service task) has to be extended in a way, that this additional meta information can be processed. The SP uses this information to refer to the requesting service composition the SLA reports he delivers to the CSP. In doing so, the response time of each executed service instance can also be provided by MI. In general, the CSP and the involved SPs have to agree on a common protocol exactly specifying the required message exchange.

As SLA violations on instance level may only be monitored ex post, quality forecast mechanisms (8) are highly desirable. On basis of these forecasts the CSP can timely initiate appropriate counter measures, for instance reconfiguring the employed core services (5). To enable a forecast, historical and instantaneous information about the service composition's usage profile is required in the first place. On basis of information the expected future response time can be calculated by employing an adequate forecast model (e.g. regression), for instance as introduced by [7, 29]. If an SLA violation becomes probable, the CSP should be able to determine, whether it is more efficient to scale his execution environment, especially the employed BPEL engine, or to include more efficient core services. Thereto detailed monitoring information about the service compositions' internal process logic is needed. At minimum, the response time of all service activity instances should be captured. In this way, the CSP can determine the time needed for messaging and for the processing. If he factors the response time of the involved core services provided by the SP into his management information base he is enabled to decide on the adequate measures in case of a predicted SLA violation. To sharpen up the forecast, one may further include information about the internal process flow, like for instance which path has been taken in case of conditional branches in order to determine how this decision influences the performance. The previously sketched forecast mechanism can also be used by the CSD for the generation of offers (2) as well as the determination of the requirements for the core services (3), as in these cases the decision is also based on an estimation of the future quality subject to the influencing factors. Hence, no additional information is required from the MI.

Once a reconfiguration of the service composition becomes necessary, either to prepare a service instance in order to fulfill a conducted agreement or to prevent a predicted SLA violation, the MI has to provide the appropriate control functionality. Hence, the MI has to allow for setting the included core services, especially their endpoint reference.

In the following chapter we present a management information model which meets these requirements.

III. INFORMATION MODEL

The information model forms the ultimately heart of each management architecture and specifies a description framework for the managed resources. It represents a management view of the system, therefore only the parameters relevant to management have to be modeled [16]. Due to the fact that SLAs are concluded for each service call, the information model has to differ between global information about a composition and information that is specific to each invocation of the composition. Thus, our information model distinguishes between two different types of managed objects (MO): MOs holding general data like configuration settings and the MOs representing instances. As pointed out in previous chapter it is not adequate to treat the composition as a whole. So it is necessary to consider that a composition consists of different elements, e.g. service tasks or gateways. Additionally a service task uses a service which is provided by the SP. Therefore the information model includes elements which describe the composition as a whole, the different composition elements and the used services. And as mentioned above for each MO a corresponding instance MO has to be provided.

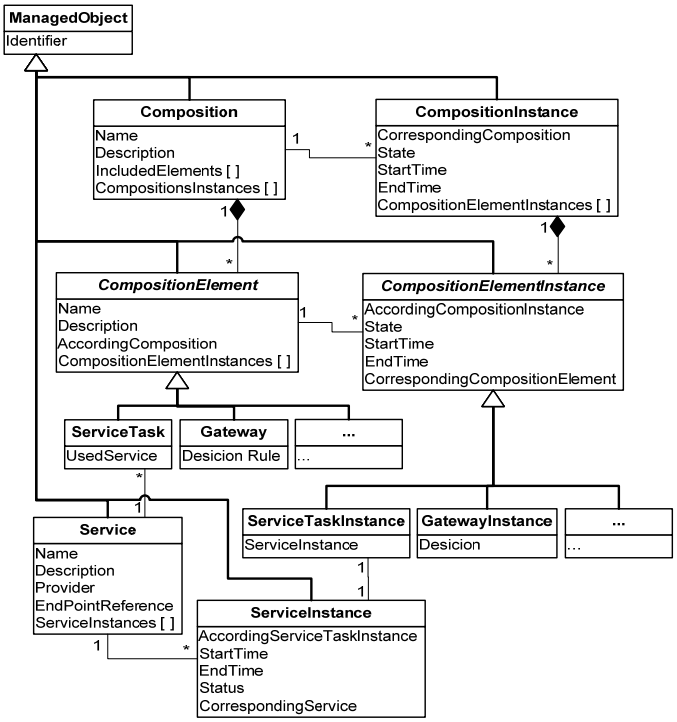

Figure 2. Information Model

According to Figure 2 all elements within the information model are derived from *ManagedObject*, so every element includes an identifier. A *Composition* comprises the inherited identifier, a name and a description. Additionally it contains a list holding the elements (*CompositionElements*) it consists of and a list referring to the currently active and completed instances (*CompositionInstances*). Thereby, a *CompositionElement* is an abstract class which comprises the data applying to all different elements of a composition. In addition to the attributes identifier, name and description it has a reference to the according composition and corresponding instances. *ServiceTask* and *Gateway* are examples of specialized *CompositionElements* with additional attributes relevant for management of these special types of composition elements. The Decision Rule is an additional part of a Gateway and the ServiceTask is extended by a reference to the service it uses (*UsedService*) whereas a *Service* can be referred by one or more *ServiceTasks*. Hence, to change the service employed within a *ServiceTask* only the reference has to be changed. Like all previous elements the service includes an identifier, a name

and a description. Additionally it holds information about the provider (in this scenario a SP), a reference to the endpoint, where the service can be found, and a list off all existing instances of this service.

For each call of the composition a new *CompositionInstance* is generated. It has a unique identifier and a reference to the composition it relates to (*AccordingComposition*). Furthermore, a *CompositionInstance* can take different states: *active, stopped, failed, and completed.* Thus it is possible to query all instances of a composition which are completed or lead to a failure. To ascertain the actual values for the SLA parameter response time within scope of the SLA compliance monitoring it is necessary to measure the execution time of a composition. But instead of saving the execution time within the information model, the start time and end time are made available for each instance. Hence, it is possible to query all instances which have been started in a specified timeslot. In this way, the usage profile of a service composition, which is for instance required by the forecast mechanisms, can be determined as well. Analogue to the *CompositionInstance* the *CompositionElementInstance* holds an identifier, a status, a start time and end time. Like the *CompositionElement* it is an abstract class with all the attributes needed by different elements of a composition. Whereas also a reference to the according composition instance and a reference to the *CompositionElement* belong to these attributes. The *GatewayInstance* represents an example for a specific *CompositionElementInstance* which is extended by the result of the decision. Accordingly, for the *CompositionElement* of type *ServiceTask* a respective *ServiceTaskInstance* is introduced. In addition to the inherited attributes, this concept includes a reference to the *ServiceInstance* employed within its execution. Like all other instance elements the *ServiceInstance* has the attributes *Identifier*, *Status*, *StartTime* and *EndTime*. But except the identifier these attributes can not be measured within the composition itself. These values have to be provided by the according SP. Hence, an integration of the service management is needed. Due to the references between all the classes within the information model it is then possible to request each service instance belonging to a service composition or each *CompositionInstance* that is associated with a service call.

IV. Architecture of the Manageability Infrastructure

The information needed for an effective composition management is defined within the information model. But as only the data and their semantics and not the accessibility are specified within the information model, furthermore a manageability interface is needed which can be used by the manager and its management application to interact with the managed objects of manageable IT system [14, 18]. The MI is then realized by the MIS, which is responsible for extracting as well as processing the management relevant data according to the information model and making them accessible through the MI.

Thereby, the reading access to the data defined in the information model by management applications are handled through getter-Operation. These operations use the functionality made available by the MIS to acquire the data and return it to the management application. Due to the fact that only the services used by service tasks can be changed without creating and deploying a new composition, the reference to the used service of a service task is the sole parameter which is also writeable. A writing access to this parameter is forwarded to the responsible setter-operation which uses the procedure described in chapter VI.B for changing the service called by the service task.

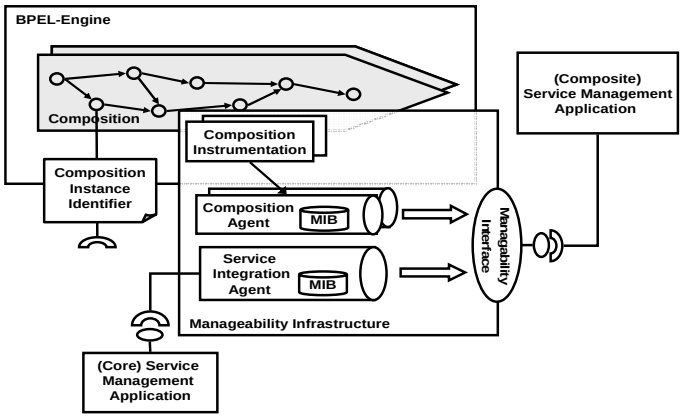

Figure 3. MIS-Architecture Overview

The MIS architecture comprises two different types of management agents, the *Composition Agents* and the *Service Integration Agent*. Every agent has his own **management information base (MIB)** reflecting the part of the information model the agent is responsible for. Each service composition is associated with one dedicated *Composition Agent* that accounts for the composition and all its instances. To provide the management information defined by the previously introduced information model each composition has to be extended by an adequate composition instrumentation [20]. This instrumentation offers the required management information to the agent and hence must be able to measure the duration of the whole composition and the single composition elements, for example service tasks and gateways. The information about service calls can not be provided by the composition instrumentation, this information has to be provided by the SP itself. To ascertain the response time of a service instance he employs a core service management application, which is not being further regarded within this paper. To associate the duration of service instance with the service task instances, each service call has to be extended by an identifier of the composition instance. The SP includes this identifier into is reports. In this way, the *Service Integration Agent* can integrate the measurement data into the information model and provide them to the manageability interface.

In the following chapters the introduced components are explained in detail, beginning with the integration of the core service management.

V. Integration of the Core Service Management

As pointed up before, a tight integration of the core service and the composite service management is required to enable a comprehensive SLA management for the service compositions. Hence, in this chapter we'll propose a communication model for the interaction between the SP and the CSP.

Basically, the SP has to provide the CSP with detailed SLA reports comprising information about the sold services (or instances) with respect to the quality (in our case the response time) he agreed on. As the CSP wants to correlate this information with management information about the service composition instances he offered, the SLA reports for the core services have to contain a reference to the corresponding service composition instances. However, the SP will allocate the service instance not until the service is actually requested [9]. Hence, the instance identifier has to be communicated with the service call. It may not be included in the corresponding SLA as it is not available at that time. For the desired solution to this problem we identified the following requirements:

- The existing core service descriptions (i.e. WSDL [37]) and the internal implementation of the core services should be retained unchanged.
- The solution should be platform-independent and interoperable, meaning that it abstracts from specifics engine used for executing the compositions.

Hence, we arrived at the conclusion, that the necessary meta information (in our case the identifier) have to be included into the SOAP header [35] of each core service request performed by the service composition. By attaching an appropriate message interceptor into the execution environment for the core services, the SP can process this information without changing the existing core service. Unfortunately, the integration of information into the SOAP header might cause extravagant expenses for the CSP depending on which engine he employs. BPEL for instance does currently not support the assignment of internal variable values into the header of a message. So in case a pure BPEL engine like for instance the *Oracle BPEL Process Manager* is used a workaround would become necessary. In case of other composition engines (e.g. *Microsoft BizTalk 2006* this can be handled by simply customizing the outgoing message pipe. But as we strive for an interoperable solution and BPEL is widely accepted as a standard for composing web services, we decided on an approach that works with every engine supporting BPEL.

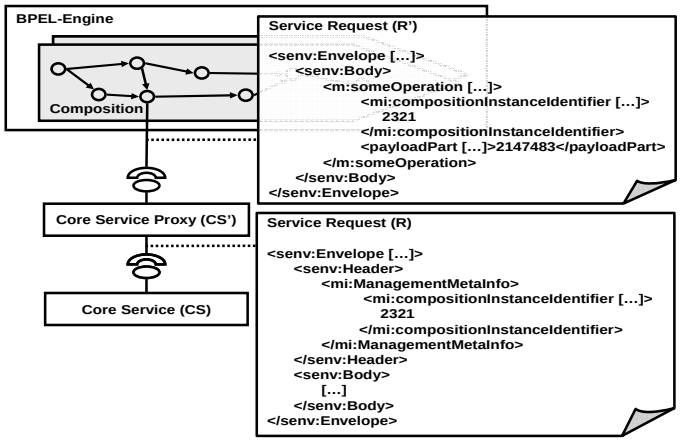

Figure 4. Integration of the Composition Instance Identifier

Thus, for each core service CS employed in the service composition we propose to generate a proxy service CS' that is deployed within the CSP's execution environment and invoked instead of CS. Thereby, each request message *R* is transformed to *R'* by adding an additional part holding the composition instance identifier. In doing so, the identifier can be set within the BPEL process definition by using standard BPEL activity "assign". Within the proxy CS' the identifier is extracted from *R'* and placed in the SOAP header of the actual request message *R*. The required transformation of both the core service's WSDL and the SOAP messages can be done by employing XSLT [37].

As already mentioned, the SP can extract the identifier by employing a message interceptor. The interceptor then has to transfer the meta information to the local management agent responsible for providing the core service response times. The management agent includes the information into his local MIB. Thereto an extension of the used information model is necessary. This aspect is not being further regarded within this paper.

Furthermore, the SLA reports provided to the CSP have to be extended by the composition instance identifier. These SLA reports are communicated to the *Service Integration Agent*, who transforms this management information to the presented information model and afterwards integrates them in its MIB. For the communication model between the SP's management application and the integration agent we decided on a trap directed polling mechanism. In this way, the communication overhead is minimized and timely processing of the data can be ensured. In particular, if an SLA violation is detected or anticipated on level of the core services the CSP should to be informed immediately, whereas the regular reports can periodically be delivered according to a predefined time interval. To implement this mechanism the SP's management application has to be enabled for sending notification events to the integration agent. For this purpose, the SP requires the specification of the agent's service endpoint along with the callback operation used for the notification. The agent on the other hand has to be provided with a specification of the service endpoint and the operation providing the full reports. The exchange of these parameters and the management information requires a mutual agreement between the SP and the CSP. Therefore it should be included in the SLA negotiation process and the parameters as well as the specification of the information revealed by the SP should be documented in the contracted SLAs.

VI. Instrumentation of the Service Compositions

As mentioned before, for the compositions an adequate instrumentation has to be set up. This instrumentation on the one hand provides the responsible *Composition Agent* with the management information needed to update the information model used for the monitoring. On the other hand, it has to facilitate a controlling intervention of the composition management, in particular a reconfiguration of the employed core services.

A. Monitoring Instrumentation

In chapter 2 we argued, that the monitoring of the service composition has to be extended to the internal process-oriented composition logic. The presented information model already takes this requirement into account. Now, a suitable approach for gathering the necessary runtime information (i.e.

instrumentation) needed. Thereby, we also strive for a solution that is applicable with all conceivable composition engines supporting BPEL. To realize the instrumentation, a very common approach in literature and practice is to query the compositions engines' audit trail for the required information [27, 19]. The audit trail usually holds the complete execution data of a composition instance. This logging mechanism along with an API required for accessing the information is provided by all composition engines. However, until now neither a standardized audit data specification nor a standardized interface is supported by the vendors. One has to mention, that such a standard has been proposed by the Workflow Management Coalition (WfMC) in the form of the Interface 5 as part of the Workflow Reference Model [17]. But this standard could not be established.

So we principally argue against using the engines audit trail interface and propose to rather extend the composition models by adequate sensors, as also promoted by [3, 4, 26]. We thereby assume that for the sake of flexibility the compositions are defined through composition models which are then being mapped to accordant BPEL process definitions. As the BPMN has been designed to be the upcoming standard for this purpose, and its application has been proved in various publications, for instance [13], we decided on building our solution on this specification. Figure 5 illustrates the approach.

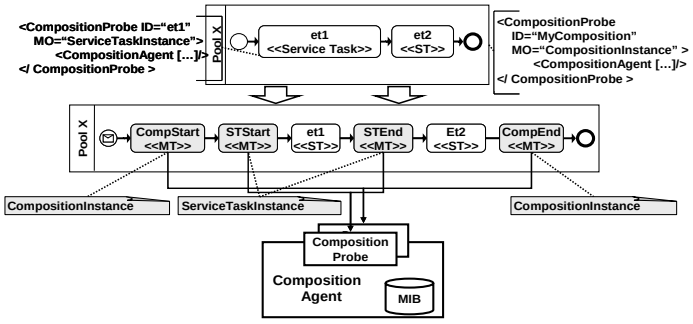

Figure 5. Generation of an Instrumented Composition Model based on BPMN

In order to obtain the required monitoring information from the service composition we propose a transformation of the BPMN-based composition model. The upper pool depicts a standard BPMN composition model containing two service tasks that are executed in sequence. As we are only interested in monitoring composition and service task instances, in case of more complex models all other activities would just be ignored. To enable the monitoring the *Composition Agent* employs suitable *Composition Probes* that provide the monitoring information for a dedicated *ManagedObject* (MO), in our case *CompositionInstance* and *ServiceTaskInstance.* Thereby, the probe basically accounts for a life-cycle monitoring of its associated MO and the propagation of state changes to the *Composition Agent*. A *Composition Probe* gathers the required monitoring information about the MO on basis of monitoring messages delivered by *Management Tasks* which are placed in the composition model. A monitoring message thereby contains a monitoring data object, which only represents the BPMN version of the MO. A *Management Task* on the other hand is a special kind of *Service Task.* But in contrast to those it only provides a one-way communication to the associated *Composition Agent*. The *Management Tasks* required for a *Composition Probe* have to be placed at appropriate positions in the existing composition model. These positions depend on the concrete type of the MO. The process of inserting the *Management Tasks* works as follows: In case of the *Composition Probe* responsible for the whole composition instance two *Management Tasks* are added to the composition model, namely one right after the *StartEvent* and one just before the *EndEvent.* The first *Management Task* provides information about the determined process instance identifier and the starting time whereas the second one only adds the end time. The instrumentation of a service task instance works similar.

It becomes clear, that for each MO a fixed procedure for adding the necessary instrumentation can be identified. Hence, the automation of these procedures can be realized by means of adequate model transformations, for instance defined through the QVT (i.e. queries, views and transformations) language. In this way, the instrumentation can be performed by simply annotating the existing BPMN model. Thereby, the *Composition Probe* along with the associated MO and the responsible *Composition Agent* have to be specified. However, since an *Management Task* requires as much processing time as any other *Service Task* and the calls are blocking the execution of the essential tasks, the presented solution might cause problems regarding the overall performance [27]. To overcome this problem we propose to place the *Composition Agent* on the same server and reconfigure its binding from SOAP to a faster binding (e.g. Java binding).

B. Control Instrumentation

Concerning the control functionality we pointed up that a mechanism for reconfiguring the employed core services at runtime is required. Thereby, it should be possible to change the core service assignment for active as well as inactive compositions or composition instances. To be consistent with the previously sketched approaches this mechanism should also rely on an interoperable solution. One way for achieving this would be the employment of dynamic endpoint references as proposed by BPEL standard [2]. In this case a distinction between active and inactive compositions has to be made.

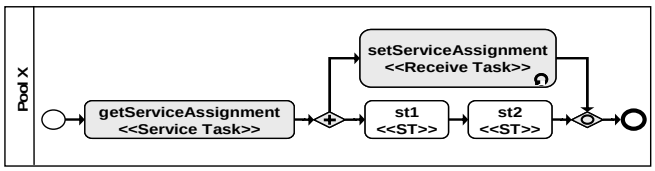

Figure 6. Dynamic Service Assignment by Extending BPEL

The service assignment (i.e. the service endpoint references) for a composition can be stored locally (for instance in a flat file) and loaded into BPEL variables at the beginning of each composition. As BPEL only supports the invocation of web services, an adequate Wrapper-Service would have to be designed. To change the assignment for composition instances that are still active, an additional operation that allows for setting the endpoints is required. This additional operation could be implemented by inserting an additional looped *Receive* activity located in a branch of a newly created parallel

flow into the composition model. The original composition logic is moved to the second branch of this flow. Both the additional *Receive* activity along with the "management flow" as well as the extra service task at the beginning used for loading the configuration could be generated automatically. However, we figured out that this BPEL-based approach leads to several drawbacks:

- The performance further suffers from the additional management-related tasks.
- Further synchronization and compensation logic is required in order to prevent synchronization problems of the two parallel branches.
- A further wrapper service has to be generated.

An alternative solution to this problem is to extend the already presented core service proxy by the functionality for changing the service assignment. Thereby, the service assignment is stored locally and the core service proxy provides an addition operation to the composition agent for reconfiguring the assignment. Then for each call the core service proxy receives from the service composition the specified endpoint is set dynamically.

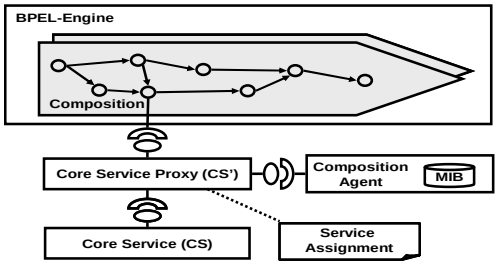

Figure 7. Generation of an Instrumented Composition Model based on BPMN

As this extension of the core service proxy could also be generated automatically and a modification of the assignment directly affects all active composition instances as well as the inactive compositions we prefer this approach.

VII. IMPLEMENTATION OF THE MANAGEABILITY INFRASTRUCTURE

In this chapter we outline an implementation of the introduced manageability architecture on basis of widely accepted management standards and technologies. To ensure a seamless integration of our solution into existing management environments, we decided to employ the set of management and internet technologies proposed by the Web-Based Enterprise Management (WBEM) standard for enterprise computing environments provided the Distributed Management Task Force (DMTF). Thereby, the objective is to enable the management of systems and applications regardless of their instrumentation type through employing a common standard [34]. This common standard is already widely used for the management of distributed applications and has recently been extended by the WS-Management [11] standard, which enables the management of and through web services. Thus, an MI based on these standards can be seamlessly integrated into SOAs. Furthermore, an implementation of these standards is included in *Microsoft Windows* since version 2000 in the form of the Microsoft Windows Management Instrumentation (WMI). But any other product supporting the WBEM standards could also be employed. Therefore, we consider it as the most appropriate foundation for a service composition management.

The core element of WBEM is the Common Information Model (CIM) [6, 10] which is used for modeling the management perspective on a system or application. CIM thereby represents an object oriented meta model based on UML and distinguishes between a core and a common model for describing the managed objects. By means of inheritance the standard set of elements can also be extended by specialized concepts. To yield the various benefits from this existing management solution based on WBEM, the information model has to be available in terms of the CIM. So the information model for service composition introduced in chapter III has to be transformed into the CIM. One could ask now why we didn't use CIM in the first place. The problem thereby is that CIM very accurately specifies the semantics of the modeling elements, and until now it is particularly designed for the management of (distributed) applications and systems. The specifics of service compositions have not been explicitly considered yet. Nevertheless, we figured out that the CIM metrics model [12] as part of the common model offers suitable concepts for modeling the management information required for service compositions. At least, all time- or transaction-based measurements are very well supported. In other cases it might not be applicable without custom extensions. In the following we will show, how our information model can be mapped to the CIM metrics model.

The CIM metrics model introduces the concept of a *UnitofWork* which pretty much corresponds to an activity or task within the composition model. From a management perspective, the processing time for a unit of work is of interest (*StartTime, ElapsedTime, Status*). Hence, the CIM metrics model already provides concepts for measuring these indicators within running instances of the unit on basis of predefined metrics. Therefore, to specify a unit of work a distinction is made between its definition and the running instances. In addition, it is possible to divide a *UnitofWork* into further subunits. Note that the concepts for the measurement of the duration have to be already seen as part of the functional management model [16]. We will use these concepts for instance to provide the overall composition duration along with the durations of each service task to the manager. These measurements may then be included in the SLA reports or employed for an SLA violation analysis/forecast.

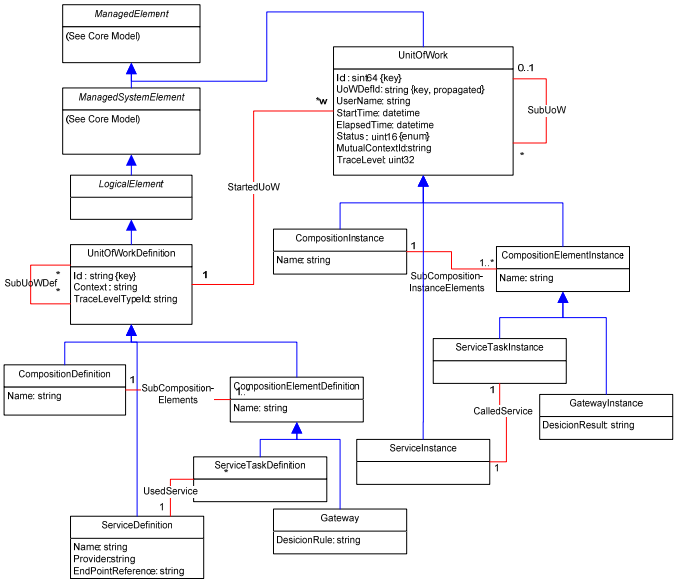

Figure 8. CIM-Model for the Management of Service Compositions

Fig. 8 shows the information model for service compositions adapted to CIM. All classes corresponding to instances of composition elements inherit from *CIM_UnitOfWork*, whereas the definitional classes referring to the whole composition inherit from *CIM_UnitOfWorkDefinition*. To model the references between the definitional classes and their running instances, e.g. composition and composition instance, the relation *CIM_StartUnitOfWork* is used. Due to the fact that associations are also classes within CIM it is possible to define new associations by using inheritance. The hierarchical references between a composition and its sub elements, like service tasks and the employed services, are modelled through specialized associations derived from *CIM_SubUoWDef* and *CIM_SubUoW*. The attributes defined for the concepts in chapter III already match the attributes specified within the CIM metrics model and therefore do not have to be changed.

Since WBEM also proposes a management architecture, the architecture described in chapter IV has also be adapted in order to be compliant with WBEM standards. The core component of the WBEM architecture represents the CIM Object Manager (CIMOM) along with the corresponding CIM repository. The CIMOM component is responsible for the implementation of the protocol-independent semantics of CIM operations. The data of the managed objects are thereby delivered and manipulated through a dedicated provider component. If an out-of-the-box implementation of the WBEM Architecture is employed, one only has to take care of the design and implementation of the provider components along with the required instrumentation for retrieving management information from the managed objects or manipulating them.

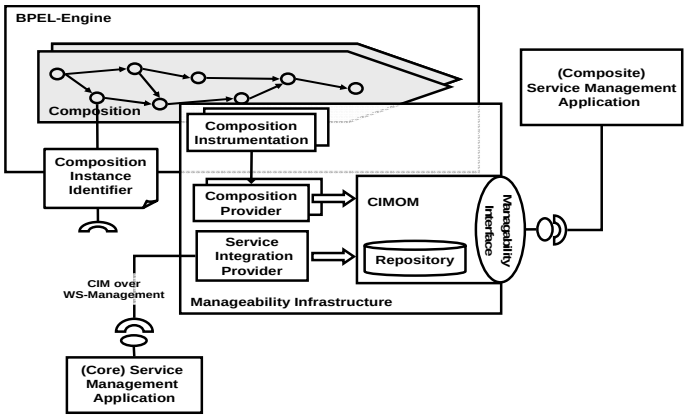

Figure 9. WBEM-conforming Management Architecture

Fig. 9 shows the adapted architecture. It consists of a central CIMOM and a repository containing the CIM definitions and objects for the management of service compositions. Furthermore, two types of provider components are needed, one for the compositions and another one for the service integration. Every composition has its own provider which is created during development of the composition by means of an integrated development process. If a new instance is started it creates a new instance of the *CompositionInstance* class and the corresponding sub classes. With the help of the instrumentation it can react on queries for this composition and answer them. The service integration provider may use CIM-XML over WS-Management, which is still under development at the DMTF. Thereby, the *Service Integration Provider* uses a WS-Management compliant interface provided by the service management application that is operated by the SP. If the SP's management is also built on a WBEM infrastructure the integration of the data relevant for composition management is easy. Otherwise a transformation is required. The CIMOM realizes a standardized interface which can be used by many different management applications, so the MIS is independent from the employed application.

VIII. Related Work

In literature, three major frameworks for an (automated) SLA-based management of web services and web service compositions have been presented. All of these frameworks rely on an instrumentation of the managed resources or objects and a manageability infrastructure, which is more or less pronounced. In the following, we'll discuss these approaches in comparison to our solution. Note that these frameworks already offer comprehensive management functionality and the manageability infrastructure only represents a small part of it. Nevertheless, our approach has some advantages to the currently employed solutions and could be integrated into these frameworks.

In [9, 21] and [24] an extensive framework for the specification, negotiation, (dynamic) provisioning and (compliance) monitoring of services on basis of SLAs, which are defined through Web Service Level Agreements (WSLA) [25] is presented. Thereby, the solution mainly focuses on the management of core web services, in particular the web service contracting, the (optimized) execution along with a runtime

monitoring and (dynamic) provisioning of underlying resources. The manageability interface is provided by a *Metering Service*, which gathers runtime information about the usage profile and the relevant SLA parameters by utilizing an instrumentation based on message interceptors (or handlers) placed within the SOAP engine. The *Metering Service* thereby transforms the information into *Meter Events*, which form the information model. There structure is not further described. Although in [9] it is mentioned that also composite web services are supported through adding parent session identifiers to the *Meter Events*, we argue that the specifics of service compositions are not fully taken into account. In particular, a monitoring of internal the composition logic may not be accomplished by tracking the in- and outgoing message flow. Furthermore, the optimized service selection for compositions and the dynamic assignment of the employed core services, which corresponds to the resource provisioning on the level of core services, are not regarded yet.

[28] present a competing framework that pretty much covers the same functionality. In contrast to the previously introduced approach it is less elaborate concerning the support for service negotiation but on the other hand explicitly supports the SLA-based monitoring of composite services. The required manageability infrastructure is founded on an object-oriented information model based on the ITU-T model, which covers both the management information concerning the core services as well as a process-oriented service composition. Hence, a monitoring of the internal composition logic is rendered possible. However, especially the presented composition instrumentation within the sketched manageability infrastructure represents a very proprietary approach as it builds on execution logs generated by the *Hewlett-Packard (HP) Process Manager*. Thus, in case another engine should be employed the instrumentation adapted as well. Furthermore, although the manageability interface is already exposed as a web service it does yet not rely on a standard like WS-Management in conjunction with CIM in XML. This complicates the integration with other management applications, for instance of the core services. Finally, the MIS does not cover any control functionality for intervening in case of a foreseen SLA violation.

The third notable approach for a SLA-oriented web service management is presented in [30, 31]. Thereby, in [32] at first specification of a language for defining web service offerings – the Web Service Offerings Language (WSOL) – is introduced. This language complements the WSDL that is used for describing the functional aspects of services and additionally allows for the formal specification of important management-information classes, functional and QoS-related constraints and management statements (e.g. price, responsibility). In contrast to WSLAs [25] service offerings are not negotiable but rather refer to fixed variations of one service from which the consumer can choose the one that is most appropriate for him. Hence, this approach perfectly supports a service allocation based market mechanisms in service marketplaces. To proof the applicability of the WSOL a corresponding management infrastructure is presented in [31]. This infrastructure enables amongst others the measurement and calculation of employed QoS metrics, the evaluation of WSOL constraints and service accounting as well as billing operations. Like in [9, 21] the scope of this framework is limited to core web services. The instrumentation and manageability infrastructure also relies on an extension of the employed SOAP engine by WSOI-specific message interceptors or handlers. In summary, the approach to a management infrastructure lacks a clear definition of a manageability interface along with a management information model and monitoring or control functionality that takes into account the specifics of service compositions is missing. These issues are addressed in [30], where the authors point up the general requirements for a service composition management in a very elaborate way. Tangible solutions to the raised research questions are not available yet.

IX. Conclusion and Outlook

In this paper, the conceptual design of a manageability infrastructure tailored to the needs of an SLA-aware management of service compositions has been proposed. The platform-independence and interoperability thereby represent the core contribution of our approach. Furthermore, we showed that the architecture can be mapped to existing, standardized management platforms without further complications. Such an implementation could also be employed within the discussed existing solutions for a SLA-based service management.

The next step for us will be the full implementation of the MIS based on WBEM and WMI and its integration into an existing SOA developed within scope of the project “Karlsruher Integriertes InformationsManagement” (KIM). Hence, we have to provide some initial management functions as well, like performance reports and availability checks. For the future we plan to include more complex functions, like for instance the SLA violation forecast with autonomous adaptation strategies. All of these solutions will be evaluated within the operated *University-SOA*. Concerning our current research, we are focusing on a methodology for an automated generation of the MIS, in particular the instrumentation. Thereby, we decided to build our approach on the principles of the Model-driven Architecture (MDA) as proposed by the OMG. Therefore, we are working on the design of adequate meta- models that allow for a seamless integration of non-functional aspects into the development process of service compositions. In this context, the realization of automated transformations operating on suitable UML profiles for describing the functional and non-functional behavior are of special concern. In addition, our management architecture is enhanced in order to support the monitoring of business process related key performance indicators (KPIs). Thus, not only service compositions that implement fully automated parts of business processes but also compositions fully implementing business processes (or workflows) including human interactions are considered. To put it in a more general way, the requirements for an integrated business process management are incorporated into the presented approach.

REFERENCES

[1] van der Aalst, Wim M. P.; ter Hofstede, A. H. M.; Weske, M.: Business Process Management: A Survey. In: Lecture Notes in Computer Science Band 2678. Springer-Verlag, S. 1-12, 2003.

[2] Andrews, T.; Curbera, F.; Dholakia, H.; Goland, Y.; Klein, J.; Leymann, F.; Liu, K.; Roller, D.; Smith, D.; Thatte, S.; Trickovic, I.; Weerawarana, S.: Business Process Execution Language for Web Services 1.1. ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf, 2003.

[3] L. Baresi, C. Ghezzi and S. Guinea. SmartMonitors for Composed Services: In Proceedings of the 2nd International Conference on Service Oriented Computing, 2004.

[4] L. Baresi, and S. Guinea. Towards Dynamic Monitoring of WS-BPEL Processes: In Proceedings of the 3rd International Conference on Service Oriented Computing, 2005.

[5] Michael Brodie, Christoph Bussler, Jos de Brujin, Thomas Fahringer, Dieter Fensel, Martin Hepp, Holger Lausen, Dumitru Roman, Thomas Strang, Hannes Werthner, Michal Zaremba: Semantically Enabled Service-Oriented Architectures - A Manifesto and a Paradigm Shift in Computer Science, Technical Report TR-2005-12-26, http://www.deri.at/fileadmin/documents/DERI-TR-2005-12-26.pdf, 2005.

[6] Bumpus, W.; Schweitzer, J. W.; Thompson, P.: Common Information Model John Wiley & Sons Ltd, New York, 2000.

[7] C. Crawford and A. Dan, "eModel: Addressing the Need for a Flexible Modeling Framework in Autonomic Computing, Proceedings of the IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS 2002), IEEE, New York, 2002.

[8] Karl Czajkowski, Asit Dan, John Rofrano, Steven Tuecke, Ming Xu: Agreement-based Service Management (WS-Agreement), 2005.

[9] A. Dan, D. Davis, R. Kearney, A. Keller, R. King, D. Kübler, H. Ludwig, M. Polan, M. Spreitzer, A. Youssef: Web services on demand: WSLA-driven automated management. IBM Systems Journal, Vol. 43 (1), 2004.

[10] Distributed Management Task Force (DMTF): Common Information Model (CIM) Specification – Version 2.2, DMTF, http://www.dmtf.org/standards/cim/cim_spec_v22, 1999.

[11] Distributed Management Task Force (DMTF): WS-Management CIM Binding Specification, DMTF, http://www.dmtf.org/standards/published_documents/DSP0227.pdf, 2006.

[12] Distributed Management Task Force (DMTF): Metrics Model, DMTF, http://www.dmtf.org/standards/documents/CIM/DSP0141.pdf, 2003.

[13] Emig, Christian; Weisser, Jochen; Abeck, Sebastian: Development of SOA-Based Software Systems – an Evolutionary Programming Approach. In: International Conference on Internet and Web Applications and Services ICIW'06, Guadeloupe, 2006.

[14] Global Grid Forum: Open Grid Services Architecture - Glossary of Terms, http://www.ggf.org/documents/GFD.44.pdf, 2005.

[15] Roy Grønmo, Michael C. Jaeger: Model-Driven Methodology for Building QoS-Optimised Web Service Compositions, The 5th IFIP Intl. Conf. on Distributed Applications and Interoperable Systems (DAIS 2005), Athens, Greece, LNCS 3543, pp. 68-82, DOI:10.1007/b137217, June 2005.

[16] Hegering, Abeck, Neumaier: Integrated Management of Networked Systems, Morgan Kaufmann Publishers, 1999

[17] Hollingsworth, D. The Workflow Reference Model. Workflow Management Coalition, TC00-1003, http://www.wfmc.org/standards/docs/tc003v11.pdf, 1995

[18] International Business Machines Corporation (IBM): An architectural blueprint for autonomic computing, IBM, http://www-03.ibm.com/autonomic/pdfs/ ACBP2_2004-10-04.pdf, 2004.

[19] Jeng, J.-J.; Schiefer, J.; Chang, H.: An Agent-based Architecture for Analyzing Business Processes of Real-Time Enterprises. In: Proceedings Seventh IEEE International Enterprise Distributed Object Computing Conference (EDOC'03). 2003.

[20] Katchabaw, M. J.; Howard, S. L.; Lutfiyya, H. L.; Marshall, A. D.; Bauer, M. A.: Making Distributed Applications Manageable Through Instrumentation, Journal of Systems and Software, Elsevier Science Inc., New York, 1999.

[21] Alexander Keller, Heiko Ludwig: The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services, Journal of Network and Systems Management, Seite 57 - 81, Springer, 2003.

[22] Steffen Lamparter, Björn Schnizler: Trading Services in Ontology-driven Market, In Proceedings of the 2006 ACM symposium on Applied computing, Dijon, France, 2006.

[23] Frank Leymann: Web Services - Distributed Applications without Limits, Business, Technology and Web, Leipzig, 2003.

[24] Heiko Ludwig, Asit Dan, Robert Kearney: Cremona: An Architecture and Library for Creation and Monitoring of WS-Agreements, ICSOC'04, New York, 2004.

[25] Heiko Ludwig, Alexander Keller, Asit Dan, Richard P. King, Richard Franck: Web Service Level Agreement (WSLA) Language Specification, IBM Corporation, 2003.

[26] McGregor, Carolyn: A Method to Extend BPEL4WS to Enable Business Performance Measurement., http://www.cit.uws.edu.au/research/reports/paper/paper03/TR-CIT-15-2003.pdf, 2003.

[27] McGregor, Carolyn; Schiefer, Josef: A Web-Service based framework for analyzing and measuring business performance. In: Information Systems and e-Business Management. Springer-Verlag,, P. 89-110, 2004.

[28] Sahai, A., Machiraju, V., Sayal, M., van Moorsel, A., Casati, F.: Automated SLA Monitoring for Web Services. In Proc. of´the 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2002), Montreal, Canada, Lecture Notes in Computer Science (LNCS), No. 2506. Springer-Verlag (2002) 28-41, 2002.

[29] Sheth, A.; J. Cardoso, J. Miller, J. Arnold, Modelling Quality of Service for Workflows and Web Service Processes; VLDB Journal, 2002.

[30] Tosic,V., Esfandiari, B.: Towards a System for Web Service Composition Management, In Proc. of the 2005 IEEE International Conference on Web Services (ICWS 2005), Orlando, USA, 2005.

[31] Tosic, V., Ma, W., Pagurek, B., Esfandiari, B.: Web Services Offerings Infrastructure (WSOI) - A Management Infrastructure for XML Web Services, In Proc. of NOMS (IEEE/IFIP Network Operations and Management Symposium) 2004, Seoul, South Korea, 2004.

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

[33] White, S. A.: Business Process Modeling Notation. BPMN 1.0, http://www.bpmn.org/Documents/BPMN%20V1-0%20May%203%202004.pdf, 2004.

[34] Andrea Westerinen, Jim Davis: WBEM Standards, DMTF 2002 Developers' Conference, http://www.dmtf.org/data/presentations/devcon02/AndreaWesterinen-WBEMStandardsTutorial.pdf, 2002.

[35] World Wide Web Consortium (W3C): Simple Object Access Protocol (SOAP) 1.1, http://www.w3.org/TR/soap/, May 2000.

[36] World Wide Web Consortium (W3C): Web Services Description Language (WSDL), Version 1.1, http://www.w3.org/TR/wsdl, Mai 2005.

[37] World Wide Web Consortium (W3C): XSL Transformations (XSLT) Version 1.0, W3C Recommendation, http://www.w3.org/TR/xslt, 16 November 1999

[38] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, Quan Z. Sheng: Quality Driven Web Services Composition, WWW2003, May 20–24, 2003, Budapest, Hungary, 2005.