Establishing Service Management in SOA

Carsten Kleiner, Hannover University of Applied Sciences and Arts, Germany

Jürgen Dunkel, Hannover University of Applied Sciences and Arts, Germany

ABSTRACT

In service-oriented architectures the management of services is a crucial task during all stages of IT operations. Based on a case study performed for a group of finance companies the different aspects of service management are presented. First, the paper discusses how services must be described for management purposes. In particular, a special emphasis is placed on the integration of legacy/non web services. Secondly, the service lifecycle that underlies service management is presented. Especially, the relation to SOA governance and an appropriate tool support by registry repositories is outlined.

Keywords: *Service Lifecycle, Service Management, Service Monitoring, Service Registry, Service Repository, Service Semantics*

INTRODUCTION

Moving to service-oriented architecture (SOA) introduces a completely new structure of enterprise IT (Krafzig et al., 2004). For each application in the enterprise reusable services must be defined encapsulating the implementation platforms and technologies used. Subsequently, these services can be orchestrated by process languages as BPEL (OASIS, 2007) or BPMN (OMG, 2011) to compose complex and application spanning business processes that realize the different business strategies (Figure 1).

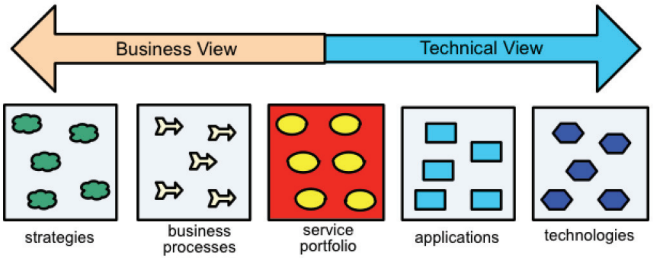
Introducing SOA causes a paradigm shift: In SOA, reusable services are the crucial components of IT infrastructure. A service portfolio that is under continual development serves as the integration platform for enterprise-wide business processes. Managing the service portfolio is an essential task during IT operations. From a general perspective, moving to SOA let an enterprise substitute application and technology management by service management.

As one major goal SOA fosters the reuse of already existing services. Particularly, service management has to support all aspects of service-reuse. Especially, it has to deal with the following issues:

- *Finding a service*: Software developers and business process designers need decent support to decide whether the service portfolio contains an appropriate re-usable service. Otherwise, a new enterprise-wide usable service must be commissioned.

DOI: 10.4018/jeei.2012010101

Figure 1. Portfolio of services in a service-oriented architecture



- *Selecting a service version*: If a suitable service has been found, the appropriate version of the service must be selected. Services will usually change over time so that different versions will co-exist. Version control regarding dependencies and incompatibilities between different service versions is indispensable for managing huge service portfolios. Therefore, change management is a significant aspect of service management.
- *Providing service access*: After the adequate version of a service has been chosen, the software developer must be provided with all information necessary to invoke the service, e.g., service signature, transport protocols and service endpoints.
- To accomplish these objectives, service management has to overcome different challenges.
- *Service semantics*: The prerequisite of service-reuse is a complete understanding of the services provided. Firstly, semantics of each service must be precisely defined so that every service requester can understand all the effects and impacts of a service invocation. Furthermore, all non-functional aspects of a service such as quality of service (QoS) and service level agreements (SLAs) must be specified by formal contracts preventing differing interpretations. For instance, the security properties or performance characteristics of a service must be known in advance.
- *Heterogeneous technologies*: Big enterprises, especially in the financial sector are based on a heterogeneous IT infrastructure, including legacy systems, e.g., CICS transaction monitors or packaged systems like SAP. In a service-oriented architecture the services of the portfolio are implemented in different technologies running on different platforms. In real-world scenarios, a service-oriented architecture cannot only rely on Web services but must integrate legacy services based on proprietary protocols and technologies.
- *Tool support*: Big enterprises might have to administer several hundreds of services with a few thousands of operations. Due to the huge number of services, an appropriate tool support is indispensable. A service registry-repository must provide all necessary artifacts to find, understand and manage the services.

In summary, the real challenge in SOA-based enterprise IT is not developing a single service, but managing a huge amount of continuously changing services.

In this paper we want to present the results of a project we realized together with five enterprises from the financial sector. The main goal was to study and evaluate different concepts for service management. We laid the emphasis on a pragmatic approach, i.e., the objective was a service management process that could be easily established in the different

enterprises. Our pragmatic approach should be based on technological standards as far as possible. Usually, an extensive tool support exists for well-established standards, so that the costs of developing own solutions can be significantly reduced. Therefore, we will discuss in particular in what extend Web services standards like WSDL, UDDI and WS* can be used in management of legacy services. Finally, we investigated the particular requirements on an adequate tool support for service management.

The paper is organized as follows: In the next section we discuss how services in a service-oriented architecture can be described. In particular, different standards and formalisms for service description are discussed. We outline the different aspects of service management related to the service life cycle. The subsequent section presents the requirements for SOA registry-repositories in some more detail. Finally, we summarize our results and provide an outlook on future lines of research and development.

Service Description

Many different aspects must be considered for using a service that is not self-implemented. First, the service description must specify the service interface providing the technical basis for service invocation. Furthermore, non-functional aspects such as quality of service (QoS) and service level agreements (SLAs) (Lee et al., 2003; Wada et al., 2006) must be formally defined. For instance, details about the security mechanisms in place must be known before passing confidential data to a service. Other management aspects support life cycle management. For instance, version information with regard to service lifecycle has to be provided. Finally, the semantics of a service must be specified to decide if a certain service can fulfill the intended tasks. For Web services decent formalisms exist for most of these aspects of service descriptions. In this section, we will discuss how they can be applied for legacy services, which are services implemented by proprietary legacy technology.

Service Interface

Different middleware approaches provide well-established formalisms for describing service interfaces, e.g., Interface Definition Language (IDL) in Corba (OMG, 1997) and Web Service Description Language (WSDL) for Web services (Booth et al., 2007). Especially WSDL allows a technology-independent description of all relevant technical aspects of a service. In WSDL four different parts are distinguished:

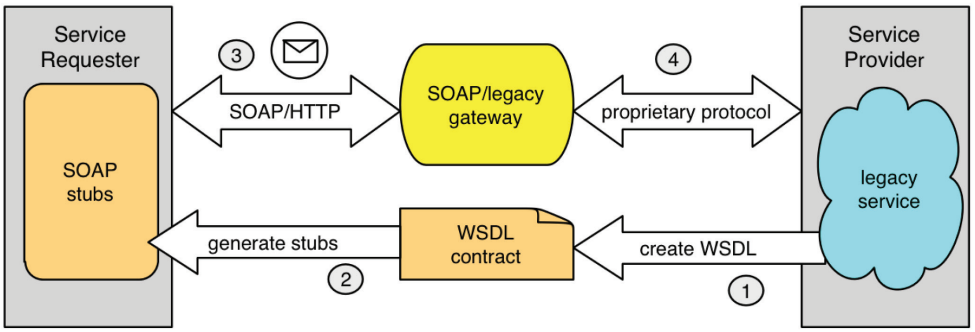
- The format of input and output messages that must be sent and received to interact with a service is defined by XML schema.
- All operations of a service are specified by the incoming and outgoing messages and the corresponding message exchange patterns (MEP) (Booth et al., 2007).
- The message format and the transport protocols are determined in the service bindings, e.g., message format (such as SOAP) and transport protocol (e.g., HTTP).
- Finally, the service endpoints describe the locations where the services reside.

A service description specifying these aspects is technically complete in the sense that the stubs classes responsible for service marshaling and demarshaling can be generated automatically.

For Web services, WSDL is the standard formalism for service description with a comprehensive tool support. However, for legacy services WSDL cannot be used directly. Nevertheless, WSDL is a good starting point for describing legacy services also, since WSDL is extensible and also because legacy system providers offer different tools to generate WSDL descriptions from legacy code.

- In legacy systems, the service operations with the corresponding input and output messages are described by proprietary data structures, e.g., by Cobol copybooks or 3270 screen buffers. To allow the usage of WSDL these legacy data structures must

Figure 2. Using SOAP/legacy gateways for integrating legacy services



be defined by XML schema as in the type part of WSDL. Fortunately, there exist tools to convert legacy data to XML, e.g., CB2XML (Thomas, 2011) or DFHLS2WS (Williams et al., 2007).

- Furthermore, the specification of the bindings and service endpoints must be adjusted to the technical characteristics of the legacy systems. The extensibility mechanisms of WSDL (Booth et al., 2007; IBM, 2011) allow to incorporate arbitrary XML code in WSDL, e.g., to define legacy specific elements and attributes. For instance, to specify a Websphere MQ service endpoint, a technology-specific XML vocabulary determining queue names and delivery mechanisms can be defined.

Integration of legacy services can be accomplished by exchanging SOAP messages (Lawrence, 2007). Figure 2 shows how a SOAP/legacy gateway helps to integrate a legacy service in a service-oriented architecture.

1. A WSDL description is generated from the legacy service.
2. Stub classes are generated from the WSDL description.
3. The service requester invokes the legacy service using the stub classes that send the input data via SOAP messages to the gateway.
4. The gateway converts the SOAP message to legacy data (e.g., a Cobol copybook) and sends it via a proprietary protocol to the legacy system (e.g., to a Websphere MQ message queue).

Return values are sent back on exactly the opposite way to the service requester.

Non-Functional Aspects

The usefulness of a particular service depends strongly on its non-functional properties (Wada et al., 2006). Generally, we can distinguish constraints from service capabilities.

Constraints are conditions that the service requester has to fulfill. In general, the messages used for service invocation must be adapted according to the given service constraints. For instance, it might be necessary that the input message for a service contains a certain type of security token (e.g., a X.509 certificate). In contrast, *capabilities* describe a particular behavior that the service provider guarantees (under the condition that the requester satisfies all the constraints). For instance, it can be asserted that the response time is smaller than a threshold or that all data is encrypted according to specified mechanisms. Service capabilities can provide decision criteria for selecting an appropriate service.

According to Lee et al. (2003) different types of quality-of-service (QoS) properties can be distinguished:

- *Quantitative* QoS properties are described by a numerical value. Typically, quantitative measures characterize service capabilities like performance, reliability or availability measurements (e.g., response time or throughput). Estimating such numerical values is very challenging. Usually performance or reliability measures cannot be calculated by formal mechanisms, but they are influenced by many different dynamically changing factors such as workload, utilization, and network traffic or hardware problems. A pragmatic approach that most enterprises use is estimating quantitative QoS properties by analyzing monitored data. Note that this approach doesn't yield any guarantees for the service requester. There are always certain circumstances under which a service cannot meet the promised quality.
- *Qualitative* QoS properties define the essential requirements and characteristics of a service that can be described by a Boolean expression. For instance, the assumed security mechanisms or the transaction behavior of a service are specified by qualitative QoS properties (e.g., the use encryption algorithms and the type of required security tokens). Usually, qualitative QoS properties must be completely fulfilled for a successful service invocation.

The dependencies between service constraints and capabilities can be defined in form of so-called Service Level Agreements (SLA). An SLA can be understood as a contract between a service provider and the service requesters about their rights and obligations. An indispensable prerequisite for establishing Service Level Agreements in an enterprise is that they are provable (Keller et al., 2003). In particular, violations of SLAs must be detected for allowing the enforcement of corresponding penalties, i.e., what happens when the service provider fails to offer the pre-agreed quality. Therefore, two prerequisites must be fulfilled.

- Service Level Agreements must be precisely specified in a formal language that is machine-readable and leaves no room for interpretation. It is crucial that non-functional aspects are described in a formal way for allowing service requester and server provider to rely on well-defined rules of service usage.
- All QoS properties need to be measurable and must be continuously monitored during the provisioning of the service for detecting SLA violations. In general monitoring is a prerequisite for contract enforcement. Each SLA might also contain some penalty clauses that specify the consequences of a specific SLA violation (Rana et al., 2007). We will discuss service monitoring in some more detail.

There are already some standards in the Web services stack that address the description of non-functional aspects. The WS Policy (Vedamuthu et al., 2007) and the WS Policy Attachment (Vedamuthu et al., 2007) offer a rather general framework for specifying non-functional service aspects for Web services such as QoS properties. WS Policy provides just formalisms for combing arbitrary assertions by using Boolean expression. However, formulating a specific assertion in a particular domain requires a domain-specific vocabulary. In some fields such a vocabulary has been already defined in corresponding WS* specifications, e.g., WS Security Policy (OASIS, 2007) defines the vocabulary for security issues, and WS RM Policy Assertion (OASIS, 2004, 2007) that for reliability aspects. However, for many areas an own enterprise- or domain-specific vocabulary must be defined, e.g., for performance characteristics such as response time or throughput. Another example for missing standards is an assertion vocabulary for cost or billing aspects. Overall, even Web services are still lacking comprehensive standards for defining policies.

Legacy services can exploit WS Policy as a description language for non-functional properties without any problem, because WS Policy is completely technology-independent. However, the assertion-specific vocabulary defined in some WS* specs cannot be easily transferred to legacy services. Instead, legacy-specific vocabularies must be set up. One example is determining the transactional behavior of a service or defining the parameters for reliable message transport both of which might be technology-dependent.

When a SLA has been formally specified, the service provider as well as the service requester can monitor SLA violations. Different types of SLA violation can be considered: defective performance, i.e., the service provides lower quality (for instance longer execution times); late performance, i.e., a service provides the promised quality with some delays; the service doesn't provide at all. Certain penalties can define the consequences of a SLA violation. Often, financial sanctions reduce the costs of using the failed service (or of subsequent service usage). Another possibility is a decrease of the service providers' public reputation.

Web services Agreement Specification (WS-Agreement) (Andrieux et al., 2007) is a standard from the Global Grid Forum that can be used as a protocol for establishing SLA contracts between service provider and consumer. This standard is XML-based and technology-independent and therefore also well-suited for legacy services.

Management Aspects

Management aspects cover all the necessary information for supporting lifecycle management. In particular, the following aspects are of interest.

- *Lifecycle state*: For each service its actual lifecycle state must be known: for instance, it is crucial, whether a service is just planned or already in operation. Note that a well-defined service lifecycle model is a prerequisite for distinguishing lifecycle states.
- *Versioning*: Because services are subject to further development and maintenance, different versions of the same service might co-exist. Therefore, each service description must contain appropriate version and release information. Furthermore, for each service version a corresponding lease time should specify when it is deprecated.
- *Dependencies*: Usually, services mutually rely on each other, especially if they are orchestrated using process modeling languages like BPEL (OASIS, 2007) or BPMN (OMG, 2011). Another type of dependency exists between all services that belong to a certain release. Overall, dependencies between services must be comprehensively defined.
- *Access Rights*: Furthermore, each service must be related to the organizational structure of an enterprise. The usage of a particular service might be restricted to a specific group of persons. (For instance, a service for increasing the salary might only be accessible by people from the personnel office.) Therefore, for each service its appropriate access rights must be specified. Because SOA services are provided by different applications with their own user accounts, an enterprise-wide integration of user accounts in form of an *identity federation* is required.

Administrative information must be accessible for each service. For describing management aspects we can also make use of the WS Policy framework. Specific XML code can define state, version and dependency information. Unfortunately, there are no standards in this field yet, so that an enterprise-specific XML format has to be developed. Only for specifying authorizations a standard formalism already exists: the Security Assertions Markup Language (SAML) (OASIS, 2005) allows the detailed specification of access rights for particular users in form of assertions.

Service Semantics

Service-oriented Architectures foster the reuse of services that are not self-implemented. Using a service without knowing its implementation details requires a complete understanding of the service´s semantics. In particular, services can be reused, that works with inputs and outputs that users can provide/accept and that provides the required functionality. For example, if a service requester is looking for a booking service for accommodations, he can also use reservation services for hotels, hostels or boarding houses. This means that he has to understand that the domain concepts 'hotel', 'hostel' and boarding house' are sub-concepts or specializations of the concept 'accommodation'. Also service parameters often require some semantic understanding. For instance, a service consumer has to know, if a parameter 'length' requires inch or cm as the appropriate measurement unit. In summary, a service requester needs service semantics for using a service appropriate; as well as for service discovery, selection, composition and replacement. Furthermore, the service users must know all about its specific restrictions and specialties.

There are a lot of different ambitious approaches for describing semantics, e.g., using complex formalisms. Those are therefore mostly used in research projects. Instead, enterprises are looking for a pragmatic way to define service semantics. In general, a range of the following formalisms can be applied:

- *Informal textual descriptions*: Actually, only few enterprises use precise formalisms for service semantics. In our case study, all companies specified service semantics only informally by unstructured text documents. But they don't feel comfortable with this approach, because textual descriptions are imprecise and leave room for interpretations.
- *Design-by-contract*: One aspect of the semantic description of a service is a precise specification of the service interface. Design-by-contract uses pre and post conditions to describe the obligations of a service caller and the corresponding guarantees of the service provider. The Object Constraint Language (OCL) of UML (OMG, 2006) allows formulating Boolean and temporal expressions for defining restrictions on operation parameters and return values. There are several tools (Demuth, 2011) that can use OCL constraints for generating code that checks pre- and post-conditions. Actually, OCL is rather established in industry and most of the enterprises are planning to enrich the service description with OCL in the near future.
- *Ontologies*: An ontology is a semantic model to describe domain concepts including their properties, relations and dependencies. Independent of the applied formalism service provider and consumer must have a common understanding of the concepts defined in the semantic model. There are many different ontology languages. Well-known are e.g., RDF (Manola & Miller, 2004), RDF-S (Brickley & Guha, 2004) and OWL (W3C OWL Working Group, 2009). Ontologies can support the intelligent search of appropriate services using reasoning and inference mechanisms.
- *Ontology languages for service descriptions*: Meanwhile, ontology languages specialized on the semantic description of (Web) services have been introduced: the most popular are OWL-S (Martin et al., 2004) and WSMO (de Bruijn et al., 2005; Lara et al., 2005) that can support semantic service matchmaking (Paolucci et al., 2002) and automatic service composition (Yang et al., 2004). Semantic matching tries to match service requests and service advertisements that both are described by using existing ontologies. This type of matching problems should be decided automatically and should yield services that provide the requested functionality. Automatic service composition generates composite services based on a high level specification of the

desired composition and a set of composability rules that compare syntactical and semantic features of the services (Medjahed et al., 2003).

For most enterprises approaches such as OWL-S or WSMO are too complex, difficult to understand and therefore too expensive (Lara et al., 2005). A more pragmatic approach is the employment of Semantic Annotations for WSDL (SAWSDL) (Farrell & Lausen, 2007), which is a W3C standard allowing the annotation of WSDL language elements. In particular, the WSDL message types can contain pointers to an arbitrary conceptual model, which for instance is described in RDF-S, OWL or another ontology language.

In summary, it is challenging to introduce semantic service descriptions in an enterprise-wide service portfolio. But our case study has shown that an incremental process is promising. First, textual descriptions can be enriched by OCL pre and post conditions. Secondly, semantic annotations based on SAWSDL can be introduced, which presume that a conceptual domain model already exists. Finally, service ontologies could be developed. They allow advanced options such as using the RDF query language SPARQL (Prud'hommeaux & Seaborne, 2008), or exploiting inference mechanisms for intelligent reasoning.

Service Management During the Service Lifecycle

In a real-world enterprise IT adequate SOA governance must be in place making use of the service descriptions in all facets as described in the previous section. It is important to note that not only technical aspects are important for governance but the domain-specific business aspects alike. This can be inferred from the fact that SOA by its very nature is supposed to be accessible by both business as well as technology experts. Naturally, SOA should also be supported by appropriate software tools.

Prerequisite: SOA Governance

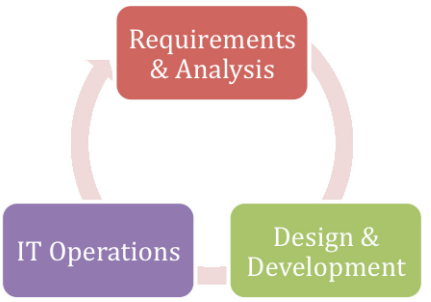
SOA governance is typically understood as the set of guidelines, rules and regulations within an organization on how the IT is structured and managed internally. It also consists of a set of processes that should be in place in order to enforce the aforementioned specifications.

Since SOA is both a technological as well as a business oriented pattern at least the first two steps in the development process differ significantly from traditional software development processes. This is due to the fact that the business departments are much more involved in the development process than before. Thus software tools used for service management have to be accessible by both technical as well as business experts. Ideally they are not only accessible but even improve and simplify the communication between those parties in the SOA process. In that way the software tools can be used to support all aspects of the process.

SOA governance does have to take all phases of the service lifecycle into account. The well-known (cf. Durvasula et al., 2008) simple lifecycle model which is sufficient for analyzing the governance aspects consists of three stages (Requirements & Analysis, Design & Development, IT Operations) which are depicted in Figure 3. It is important to stress that opposite to widespread belief, data from operations is also required in order to enable real management of a SOA. That is because services once in operation have to be monitored, their usage has to be controlled and other operations related information has to be analyzed. This analysis is often carried out by managing personnel. Thus software tools for SOA management have to provide an interface to managing personnel and also have to be integrated with IT operational systems. In fact, SOA specific extensions of those systems will be required in order to facilitate business activity monitoring.

Roles are often used within an organization in order to define responsibilities independent of particular beings. In SOA apart from the

Figure 3. Simple model of service lifecycle (cf. Durvasula et al., 2008)



classical IT roles such as system analyst, application developer and IT administrator there is the need for defining an additional set of roles. In our case studies we identified the roles of domain owner, SOA architect, service developer (not to be confounded with application developer!) and metadata administrator. These roles seem to be of sufficient general importance even though an enterprise specific extension or adjustment may be required. In any case should software tools that support service management be able to deal with such extended and flexible roles in a SOA. They should also support these roles and moreover be supportive of optimizing the cooperation between them.

Services Lifecycle

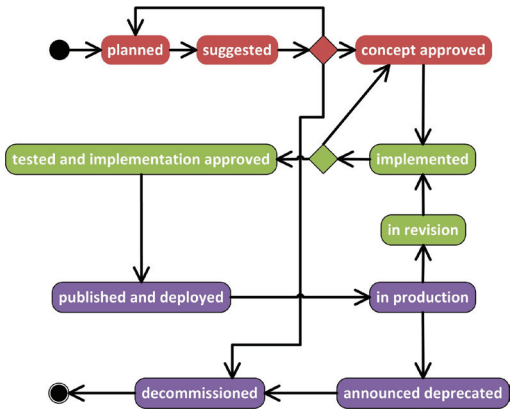
In order to be able to define the requirements for service management more precisely, it is necessary to look in more detail into the service lifecycle. We did that in our case study in cooperation with a wide range of companies, mainly from the financial sector. Though we received our empirical results solely in financial companies we feel that the foundation of our study is wide enough to lead to easily transferable results for other fields.

The more detailed lifecycle model that has been developed in our study is shown in Figure 4. Initially a service reaches the status *planned* after the idea is raised to offer a specific functionality as service. Note that the functionality does not have to be a new one; this model also applies to legacy functionality that is lifted to a service as well as external functionality to be integrated. The potential service is described in a concept paper which is then submitted to the local SOA managing board or SOA architect (depending on the particular organization), reaching the state *suggested*. The managing instance will then decide on the future development regarding this service:

- The concept may be approved if the service seems beneficial, is subject to implementation in the near future and thus entered into the development pipeline achieving the status *concept approved*.
- The service may seem beneficial in general, but the concept is not yet sufficiently detailed, completed or contains inconsistencies; in this case the service is returned to the suggesting party for an extended planning stage.
- The service is considered not relevant or already existing; in this case the service will not be considered for implementation and may be treated as *decommissioned*.

In the design and development phase a service will be implemented and thereafter is subject to testing and validation similar to traditional software components. It may either be considered ready for production moving to the *tested and implementation approved* status. Or the service still contains errors or deviates from the original concept in which case it is returned to the developers and moves back to the *concept approved* state.

Figure 4. Detailed model of service lifecycle



After successful testing a service is deployed and thus published to the IT operations stage. After operations has made the service available on a production system it reaches the *in production* status. This is the status where service monitoring takes place as described in the previous section, generating all necessary usage information for the services. This information should be stored in a central location ideally the same location which has been used for managing services throughout the previous stages of the lifecycle. Other developers and especially the SOA governing instance in the company may use such data for planning the future development of a service. There might be the need for certain adjustments which leads to the service going into the *in revision* state. In this state the next version of the service is planned and implemented while the current version may remain in production.

It might also be detected that a certain service is (no longer) required or used in which case it can be prepared for removing from the production system. This is expressed by the *announced deprecated* state. After a certain period of time the service is physically removed from the production system. The particular time interval depends on the service and company in question. This is expressed by the *decommissioned* state. At this time the service is no longer available but all meta-information assembled about the service is still available. This may be important for future services and SOA planning in general.

The increased complexity of the lifecycle coupled with the governance requirements from the previous section causes strong demand for advanced software support for service management. Tools that provide such functionality are typically called service registries and/or service repositories. In the next section we will describe requirements for these tools based on the discussion of service management in this section.

Using Registry Repositories for Service Management

Support During Service Lifecycle

As already discussed in the previous section a central software tool for managing all information about services in all different stages of the lifecycle is extremely beneficial. Such service registry-repositories (SOA-RR) are an important component of a SOA and support service management as well as governance throughout the full service lifecycle.

In particular a SOA-RR may be used for the following important tasks:

- In the requirements and analysis stage it may be used as a well-defined instance between the different parties involved in this part of the lifecycle (e.g., SOA board and

service designers). It may assist in securing a redundancy-free SOA as well as conformance to the technological guidelines within the company. Moreover it facilitates a valid judgment of the business case for any given service by the managing board. Finally it may be used for enforcing a proper process model as well as conformance to the defined business strategy.

- During design and development it may increase reuse of basic services (cf. Krafzig et al., 2004) by the developers and may assist them in observing version-based dependencies between services. A SOA-RR supports the compliance with defined SLAs both on the basic service level as well as on the business service and process levels. Finally, it assists SOA board and lead developers in gaining an overview of the current implementation status of the company's SOA at any given time.
- In the operations stage information in the SOA-RR may be used for discovering rarely used services and to prepare decommissioning of services by e.g., identifying dependent services. It may also be used as a single point of storage for change requests to a given service. Finally, the information in the SOA-RR could also be used for accounting information in case the company internally or externally uses a charge model for the services.

Information for this final stage is in many cases based on data that is generated while services are in production. Monitoring life services and feeding the information back into the SOA-RR is a very important feature and is thus discussed in more detail.

Detailed Requirements for SOA-RR

In our study we also tried to evaluate existing commercial offerings of SOA-RR products. In order to do so we analyzed the requirements for SOA-RR described in the previous section. This leads to a functionality-oriented list of detailed requirements for SOA-RR in order to perfectly support service management.

We structured the necessary functions into four categories, namely basic functions, static functions, dynamic functions and technical functions which will be explained in more detail in the sequel.

Among the *basic functions* is the storage and management of all artifacts that are important for a SOA, namely services themselves, descriptions of interfaces, message formats, business processes and policies. All enterprises regardless their internal structures and processes require storage and management of full-text documents and arbitrary binary files. Of course all required metadata associated with any of these artifacts should also be managed by the SOA-RR. Finally it is not only required to store this information but also to be able to search for and visualize information. Note that services are not restricted to Web services in the context of most enterprises: other types of services integrated into the SOA such as legacy services should be equally manageable by the SOA-RR.

With *static functions* we denote all functionality that is not inherently related to operation of services in the production environment. Static functions support service developers, architects and the SOA board in the earlier stages of the lifecycle. Static functions include service discovery in order to increase the level of service reuse as well as dependency management between services and/or versions of services. These are already required during analysis and design in order to be able to improve the design of services. Moreover management of connections between different artifacts in a SOA such as versioning and classification of services is also necessary in the analysis and design stages already. It should again be noted that integration of legacy services (cf. IBM, 2011) is already very important in this set of functions as the SOA architects need a complete view on all services in the IT infrastructure in order to achieve the best possible overall design.

The IT operations part of the service lifecycle also has to be supported by a SOA-RR. Such *dynamic* or *runtime functions* may be of a very technical nature such as user,

roles and rights management. They may also be of interest for the managing instances of a company considering e.g., support for change management or accounting of service usage. If required in a particular SOA the support for dynamic service discovery and binding also originates from dynamic functions supported by the SOA-RR. In summary, the requirements in the area of dynamic functions can be roughly structured into the following groups:

1. Lifecycle management
2. User and rights management
3. Change management
4. Logging
5. Monitoring/Accounting
6. Governance support

Starting with these groups the requirements for a SOA-RR can be further detailed into specific micro requirements which can then be evaluated against products in question. The groups for the dynamic functions which have been named lead to the following micro requirements:

1a. Service lifecycle can be managed.
1b. User-defined service lifecycle can be defined and will be used.
1c. States in a service lifecycle can be assigned automated actions to be performed upon entry/presence/exit of a given state.
1d. User and rights management of SOA-RR is specific to certain states of a lifecycle.
2a. User and rights management is SOA-RR specific.
2b. User model for SOA-RR is based on roles.
2c. SOA-RR can be configured to use external rights management software (e.g., LDAP).
3a. Changes to a service are managed by SOA-RR.
3b. Automated notifications are sent by SOA-RR upon changes to a service to pre-configured users.
3c. Users can manually register to receive notifications upon changes to a service.
3d. Automated registration of users to receive notifications upon changes to a service are possible based on certain conditions; conditions may include information from previous states of the service lifecycle, e.g., authors of a service are automatically notified when a new version of the service has been published by a different author.

In a similar manner the other groups of dynamic requirements can be further detailed as well as these exemplary groups. This is omitted here due to space constraints.

Among the *technical functionality* to be provided by a SOA-RR are traditional database management system functions such as reliability, backup, recovery features and the option to operate it in a distributed environment. Also the potential for easy integration into the given IT infrastructure of the company is extremely important leading to requirements such as extensibility, support of well-known standards and publicly available API.

All these clusters of requirements can be detailed in the same two-step process as has been shown exemplarily for the dynamic functionality cluster and the groups of lifecycle management, user and rights management and change management. All the micro requirements extracted from this process together form a huge set of potential functions that almost no product will be able to fulfill without customized extensions. Therefore the complete list of micro requirements should then be assembled in a spreadsheet, prioritized and weighted (by each potential using company individually) in order to achieve a customized set of the most important functions in a given enterprise environment. Even if several of the listed functions do not seem immediately required for a given company they should be kept in mind, because it is very likely that they are required later on with increasing level of SOA maturity.

In order to assess the usability of a certain SOA-RR product and/or compare it with other products one requires an additional set of practi-

cal requirements. These are extremely company and specific policy dependent. They include issues like quality of documentation, complexity of installation, configuration and operation as well as ergonomic user interface. In addition issues like extensibility, price, licensing options and reference customers may be used as micro requirements in the non-functional area as well. They also have to be included into an overall judgment of a specific SOA-RR tool.

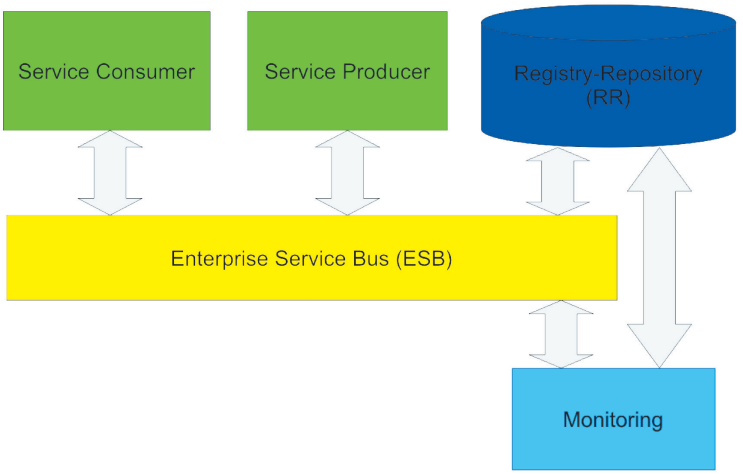
In our cooperation project we performed a market study to narrow the list of potential candidate products for our cooperating companies to a short list in a first step by analyzing system documentation. Note that parts of the individual priorities have already been used to determine the products on the short list. The products on that short list have been: Software AG CentraSite, IBM WebSphere Service Registry & Repository, Bea AquaLogic Service Registry (now: Oracle Service Registry), Systinet Registry (now part of: HP SOA Systinet). Finally for each cooperating company we used the individual priorities and weights to compare the products from the short list in detail in a second step. This comparison was based on concrete installations of the products and application to real-world services from the partners. The results strongly depend on the individual situation and thus no generally valid recommendation can be made.

Service Monitoring

Of particular importance for SOA management and governance is monitoring the current production operation of an enterprise's SOA. This is already obvious from the discussion of non-functional service aspects as well as from the description of SOA governance tasks. For service monitoring should provide an understanding of the quantitative aspects the entire SOA. A main goal is to guarantee Quality of Service, i.e., stability and trust in a dynamically changing IT infrastructure. Recently, Assurance Networks have been introduced discussing adequate mechanisms for achieving service quality (Dunkel, 2011; Kakuda & Malek, 2011). In particular the following issues can be distinguished:

- *Bottleneck analysis*: An important aspect of the technical SOA management is the detection of bottlenecks in the enterprise IT allowing performance-tuning activities. To be able to do that information about the execution times of and average response time of services and processes as well as error rates and call frequencies is required. This information can be obtained while monitoring the operation of the services within a SOA.
- *Network diagnosis*: Another capability of system monitoring is a network diagnosis in real-time. The system manager are interested in detecting threads like network intrusion, hardware or network failures as early as possible to take appropriate actions.
- *Capacity planning*: IT operation's management requires detailed monitoring information as the basis for capacity planning and infrastructure optimization.
- *Managing SLAs*: Furthermore, service monitoring is a necessary infrastructure for establishing Service Level Agreements in an enterprise. Only observed performance measurements allow a rather precise estimation of QoS properties. And service monitoring is the tool for guaranteeing the compliance with defined service level agreements, i.e., for detecting SLA violations during system operations.
- *SOA governance*: Based on the monitored atomic data further information can be derived, which is important for the general SOA governance. Among that information may be billing and accounting for commercial services and the conformance to defined security policies. This may go as far as monitoring call statistics for certain services and advertising these. Also business statistics might be generated from process monitoring information depending on the SOA maturity level of the organization.

Figure 5. Architectural model for SOA/service monitoring



The companies in our case study also considered service monitoring to be so important that a follow-up project had been started.

Depending on the particular goals of service monitoring in an organization and the tools employed, several different architectural options are possible. In the simplest case each service would report the desired data to a central monitoring component in fixed intervals. This may be sufficient for simple monitoring information such as call frequency but is not really satisfactory as foundation for SOA governance.

Another option might be to define a fixed monitoring interface which every service has to implement. This interface would then be called by the monitoring component in certain intervals. The benefit over the first option is that a central monitoring component owns the monitoring process. Thus service errors or unfulfilled SLAs can be detected. The major drawback is that the service implementation itself has to be concerned with monitoring by implementing the interface. To overcome this, a loosely coupled architecture for monitoring should be preferred as shown in Figure 5.

In this architecture the monitoring component is independent of the services. It operates directly on the ESB to collect important execution information. It is also connected to the SOA-RR for two reasons: firstly the monitoring component operates based on the RR to obtain meta information required for proper operation. Secondly it assembles important monitoring information and stores it in the SOA-RR. This is required because the SOA-RR is the foundation for SOA governance, i.e., the instance where the monitoring information will be evaluated.

SUMMARY

In this paper we have discussed important aspects of managing services in a service-oriented architecture. The discussion and findings are based on a case study that we have performed in cooperation with several major companies from the financial sector in the Hannover region. Of specific interest for these companies is the integration of legacy/non-web services within their SOA.

In particular we have discussed which aspects of services need to be described for service management. For each of these aspects we have given example languages that can be used. Thereafter we have shown that service management in an enterprise environment requires SOA governance. Such governance has to be based on the specific service lifecycle in

a SOA and should be supported by appropriate software tools. A specific lifecycle model which has been developed in our project has been presented and the SOA registry-repositories (SOA-RR) have been identified as the central piece in SOA governance. Also we have discussed why service monitoring and the feedback of results of this monitoring into the SOA-RR are very important for mature service management; this is still not standard in many products nowadays.

Consequently, SOA-RR tools have to support all the XML based languages used for service descriptions as well as all the requirements arising from supporting the service lifecycle. We have assembled all these requirements and shown that they can roughly be classified as basic, static, dynamic or technical. Advanced software tools are on the market and the best choice depends on the individual priorities of the particular enterprise. For the companies in our project we have developed a balanced scorecard. Based on this, each company is able to define its individual weights for the complete list of requirements. After defining such weights the best individual choice of a SOA-RR product at the time of writing can be concluded.

Future Work

Some parts of our research are subject to constant changes while others seem pretty stable. The general aspects of service description are not expected to change much whereas the particular languages used to describe them may change significantly in the near future. In particular the area of service semantics (maybe in a given domain context) might be subject to rapid improvement. Consequently, the requirements to support semantic service descriptions might increase.

The validity of the service lifecycle modeled in our project should be evaluated in real-world scenarios in other business sectors as financial. While the general stages seem pretty reasonable for every sector there might be some specific parts which are domain specific. Requirements for SOA-RR would have to be adjusted accordingly.

The tool evaluation results are undergoing constant change as the tools are improving constantly. Thus the partner companies of our project will need to monitor the market closely in order to be able to derive the most recent results at any time. Currently the companies are at different stages in the process of establishing a service management as described in this paper. Thus the time when a SOA-RR is required to establish a proper service management differs from immediate to medium-term.

Apart from service semantics we see the most potential for improvements in the near future in the area of service and business process monitoring. We are currently working on a project with the same companies in order to establish a service and process monitoring within their SOA. Particular focus in this project is on generation of monitoring information and the feedback of such data into the SOA-RR. This is required to establish a business process and service controlling based on monitoring.

ACKNOWLEDGMENT

The authors would like to thank the partner companies from the competence center for information technology and management (CC_ITM) for funding of this research project. In particular they would like to thank the companies for practical support and domain-specific cooperation during the project.

REFERENCES

Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., et al. (2007). *Grid Resource Allocation Agreement Protocol (GRAAP) WG: Web Services Agreement Specification (WS-Agreement)*. Retrieved October 24, 2011, from http://www.ogf.org/documents/GFD.107.pdf

Booth, D., & Liu, C. K. (2007). *Web Services Description Language (WSDL) Version 2.0 Part 0: Primer*. Retrieved October 24, 2011, from http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626

Brickley, D., & Guha, R. V. (2004). *RDF Vocabulary Description Language 1.0*. Retrieved October 24, 2011, from http://www.w3.org/TR/rdf-schema/

de Bruijn, J., Bussler, C., Domingue, J., Fensel, D., Hepp, M., Keller, U., et al. (2005). *Web Service Modeling Ontology WSMO*. Retrieved October 24, 2011, from http://www.w3.org/Submission/WSMO/

Demuth, B. (2011). *Dresden OCL*. Retrieved October 24, 2011, from http://www.dresden-ocl.org/index.php/DresdenOCL

Dunkel, J. (2011). Assurance networks from a software perspective. In *Proceedings of the 10th International IEEE Symposium on Autonomous Decentralized Systems*, Tokyo, Japan (pp. 441-448).

Durvasula, S., Guttmann, M., Kumar, A., et al. (2008). *SOA Practitioners' Guide Part 3: Introduction to Services Lifecycle*. Retrieved October 24, 2011, from http://www.soablueprint.com/practitioners_guide

Farrell, J., & Lausen, H. (2007). *Semantic Annotations for WSDL and XML Schema*. Retrieved October 24, 2011, from http://www.w3.org/TR/sawsdl/

IBM. (2011). *Write the WSDL extensions*. Retrieved October 24, 2011, from http://publib.boulder.ibm.com/infocenter/iseries/v5r3/index.jsp?topic=/rzatz/51/webserv/wswsifattwsdl.htm

Kakuda, Y., & Malek, M. (2011). A unified design model for assurance networks and its applications to mobile adhoc networks. In *Proceedings of the 10th IEEE International Symposium on Autonomous Decentralized Systems*, Tokyo, Japan (pp. 637-644).

Keller, A., & Ludwig, H. (2003). The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *Journal of Network and Systems Management*, *11*(1), 57–81. doi:10.1023/A:1022445108617

Krafzig, D., Banke, K., & Slama, D. (2004). *Enterprise SOA: Service-Oriented Architecture Best Practices*. Upper Saddle River, NJ: Prentice Hall.

Lara, R., Polleres, A., Lausen, H., Roman, D., de Bruijn, J., & Fensel, D. (2005). *A Conceptual Comparison between WSMO and OWL-S*. Retrieved October 24, 2011, from http://www.wsmo.org/2004/d4/d4.1/v0.1/20050106/d4.1v0.1_20050106.pdf

Lawrence, C. (2007). *Adapting legacy systems for SOA*. Retrieved October 24, 2011, from http://www.ibm.com/developerworks/webservices/library/ws-soa-adaptleg/

Lee, K., Jeon, J., Lee, W., Jeong, S.-H., & Park, S.-W. (2003). *QoS for Web Services: Requirements and Possible Approaches*. Retrieved October 24, 2011, from http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/

Manola, F., & Miller, E. (2004). *RDF Primer.* Retrieved October 24, 2011, from http://www.w3.org/TR/rdf-primer/

Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIllraith, S., et al. (2004). *OWL-S Semantic Markup Web Services*. Retrieved October 24, 2011, from http://www.w3.org/Submission/OWL-S/

Medjahed, B., Bouguettaya, A., & Elmagarmid, A. (2003). Composing Web services on the Semantic Web. *The Very Large Data Base Journal*, *12*(4), 333–351. doi:10.1007/s00778-003-0101-5

OASIS. (2004). *Web Services Reliable Messaging TC WS-Reliability 1.1*. Retrieved October 24, 2011, from http://docs.oasis-open.org/wsrm/ws-reliability/v1.1/wsrm-ws_reliability-1.1-spec-os.pdf

OASIS. (2005). *SAML V2.0 Executive Overview*. Retrieved October 24, 2011, from http://www.oasis-open.org/committees/download.php/13525/sstc-saml-exec-overview-2.0-cd-01-2col.pdf

OASIS. (2007). *Web Services Business Process Execution Language Version 2.0*. Retrieved October 24, 2011, from http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

OASIS. (2007). *Web Services Reliable Messaging Policy Assertion (WS-RM Policy) Version 1.1*. Retrieved October 24, 2011, from http://docs.oasis-open.org/ws-rx/wsrmp/200702/wsrmp-1.1-spec-os-01.pdf

OASIS. (2007). *WS-SecurityPolicy 1.2*. Retrieved October 24, 2011, from http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.2/ws-securitypolicy.html

Object Management Group (OMG). (1997). *Common object request broker architecture*. Retrieved October 24, 2011, from http://www.corba.org

Object Management Group (OMG). (2006). *Object Constraint Language Version 2.0*. Retrieved October 24, 2011, from http://www.omg.org/spec/OCL/2.0/

Object Management Group (OMG). (2011). *Business Model and Notation (BPMN) Version 2.0*. Retrieved October 24, 2011, from http://www.omg.org/spec/BPMN/2.0/

Paolucci, M., Kawamura, T., Payne, T., & Sycara, K. (2002). Semantic matching of web services capabilities. In I. Horrocks & J. Hendler (Eds.), *Proceedings of the First International Semantic Web Conference* (LNCS 2342, pp. 333-347).

Prud'hommeaux, E., & Seaborne, A. (Eds.). (2008). *SPARQL – Query Language for RDF*. Retrieved October 24, 2011, from http://www.w3.org/TR/rdf-sparql-query/

Rana, O., Warnier, M., Quillinan, T., & Brazier, F. (2007). Managing violations in service level agreements. In *Proceedings of the Usage of Service Level Agreements in Grids Workshop, alongside ACM/IEEE*, Austin, TX.

Thomas, P. T. R. (2011). *CB2XML: Cobol copybook to XML converter*. Retrieved October 24, 2011, from http://sourceforge.net/projects/cb2xml/

Vedamuthu, A. S., Orchard, D., Hirsch, F., Hondo, M., Yendluri, P., Boubez, T., & Yalçinalp, Ü. (2007). *Web Services Policy 1.5 – Attachment*. Retrieved October 24, 2011, from http://www.w3.org/TR/ws-policy-attach/

Vedamuthu, A. S., Orchard, D., Hirsch, F., Hondo, M., Yendluri, P., Boubez, T., & Yalçinalp, Ü. (2007). *Web Services Policy 1.5 – Framework*. Retrieved October 24, 2011, from http://www.w3.org/TR/ws-policy/

W3C OWL Working Group (Ed.). (2009). *OWL 2 Web Ontology Language*. Retrieved October 24, 2011, from http://www.w3.org/TR/owl2-overview/

Wada, H., Suzuki, J., & Oba, K. (2006). Modeling non-functional aspects in service oriented architecture. In *Proceedings of the IEEE International Conference on Services Computing* (pp. 222-229).

Williams, N., Herrman, R., Lopez, L., & Ebbers, M. (2007). *Implementing CICS Web Services*. Retrieved October 24, 2011, from http://www.redbooks.ibm.com/redbooks/pdfs/sg247206.pdf

Yang, J., & Papazoglou, M. (2004). Service components for managing the life-cycle of service compositions. *Information Systems*, *29*(2), 97–125. doi:10.1016/S0306-4379(03)00051-6

Carsten Kleiner received a MS in computer science from Purdue University in 1996, a diploma degree in mathematics and computer science from the University of Hannover, Germany, in 1997 as well as a doctoral degree (Dr rer nat) from the University of Hannover in 2003. Thereafter he worked for several years as a technical consultant and project manager in the software industry. Since 2004 he is a professor for secure information systems in the computer science department of the University of Applied Sciences and Arts Hannover, Germany. His research interests include database and information systems and their applications, as well as information systems for mobile devices. His focus in this area is on security aspects and web services for mobile platforms. He is also interested in computer science education, specifically for capstone courses. Carsten Kleiner is a member of the German Computer Science Society (GI) as well as the Association for Computing Machinery (ACM).

Jürgen Dunkel received a Diploma degree in computer science from University of Dortmund in 1984 and a Doctoral degree from University of Hagen (Germany) in 1989. Afterwards, he worked for several years as a software architect and project manager in software industry. Since 1998 he is a professor of computer science at the Hochschule Hannover (University of Applied Sciences and Arts). His research interests include software architecture, event-driven architecture, semantic models, and model-driven software development. Currently he is working in several projects applying complex event processing in sensor networks. Jürgen Dunkel is a member of the German Computer Science Society.