Models and Tools for SOA Governance

Patricia Derler[1] and Rainer Weinreich[2]

[1] Paris-Lodron University of Salzburg, Austria
[2] Johannes Kepler University of Linz, Austria
patricia.derler@sbg.ac.at, rainer.weinreich@jku.at

Abstract. Organizations are moving rapidly towards Service-Oriented Architectures (SOAs). Benefits include cost reduction through reuse, better integration through standardization, and new business opportunities through agility. The successful implementation of an SOA requires not only protocols and technologies like SOAP and WSDL but also support for the processes of creating, validating and managing services in an enterprise. Tools for SOA governance and management are evolving to be the heart of enterprise SOAs. We present an approach for supporting SOA governance activities. Notable aspects of our approach are an extensible model for describing service metadata of arbitrary service types (not only Web services), the concept of service proposals for the process of service specification and service creation, a service browser for service reuse, and support for service evolution through information about service versioning, service dependencies and service installations.

Keywords: Service-Oriented Architecture (SOA), SOA governance, service metadata, service repository, service life cycle.

1 Introduction

Organizations are adopting Service-Oriented Architecture (SOA) as the central principle for structuring their enterprise-wide software system architectures. SOA is both a business strategy and an architectural principle [1]. The business strategy aligns the software infrastructure with the business processes of the organization by modeling business processes as high-level services. From the architectural viewpoint, SOA is a means of partitioning the functionality of a software system into reusable and composable software components (services).

The frequently mentioned benefits of an SOA include cost reduction through cross-organizational reuse of services, agility through alignment with business processes, new business opportunities through agility, independent development through separation of concern, better scalability through isolation, and integration of legacy systems through the additional service layer [4]. The last aspect is especially attractive, since existing legacy systems can be wrapped and reused by services instead of being replaced.

Typically, the term SOA goes hand in hand with Web services [5]. Web service specifications such as SOAP [17], WSDL [18], UDDI [13], the WS-* family [3], and BPEL [19] are important prerequisites for business to business (B2B) and enterprise application integration (EAI).

For this reason, the focus of much work in this area has been (and still is) on Web services and related standards. However, standards for interoperability are not sufficient for realizing the benefits of SOA. In addition, activities for creating, validating, deploying and managing services need to be supported. Therefore, approaches for SOA governance and management are emerging [4].

We present an approach for supporting the specification, reuse, creation, deployment and evolution of enterprise services. The approach is based on an extensible model for service metadata and includes tools for describing services, creating service proposals, browsing and searching a service repository, and analyzing service dependencies. The approach is the result of analyzing the requirements of the IT infrastructures of cooperating financial institutions in Austria.

2 Overview

This paper is structured as follows: In Section 3 we describe the organizational processes and system structure for which we developed our approach; this includes a description of service-related artifacts and their relationships. In Section 4 we describe the process for identifying, specifying, creating, developing and deploying enterprise services; this section gives an impression of various activities and roles and presents a typical service life cycle. Section 5 contains an overview of our model and shows how the requirements and the processes described in the previous sections are represented in the model. In Section 6 the developed tools are described along with how selected activities and processes that are important in a typical service life cycle are supported. Section 7 contains a discussion of our approach. In Section 8 we comment on related work. Finally, the paper concludes with a summary and a description of the main points in Section 9.

3 Organizational Processes and System Structure

The presented models and tools were developed for a particular organization in the financial domain. The organization provides and manages the IT infrastructure of financial institutions (banks, insurance companies) and cooperates with other IT organizations and departments in this domain. For this purpose, the organization develops and deploys services.

As the following description shows, the system structure and requirements are generally applicable to many commercial organizations and the example is fairly typical for the application of a service-oriented approach.

To explain the processes involved in creating, deploying, hosting, and managing services, we need to elaborate on service development and service operation.

For a number of institutions, the organization develops and deploys applications and services which it operates in its own computing center. Those applications and services might use services operated by partner institutions, and services operated by the organization itself might be used by applications and services of partner institutions (B2B integration). In addition, services developed by the organization might also be operated by partner organizations and vice versa. This leads to rather complex application and service relationships that need to be managed. Important questions include the following:

(q1) Which services are available?
(q2) Where is a service deployed?
(q3) Which organization is the service provider?
(q4) Who is responsible for service development and evolution?
(q5) Which other services does a service need?
(q6) Which (external) applications and services depend on a service?

The first question (q1) focuses on service (re)use and is important during service specification and development. The questions about service dependencies (q5 and q6) are especially important for service evolution and release management. They help to determine how customers and partners are affected by a new version and which clients are affected (q3 and q4): a customer who simply uses a service might need to update his software (depending on compatibility issues); while a partner who operates a service needs to be informed that a new version is available (q2).

The system structure of the organization contains typical elements of an SOA. Business logic and processes are distilled into services that typically integrate legacy systems. In this context, a huge part of the organization's business rules are implemented on a mainframe and can be used via Customer Information Control System (CICS) [6] transactions, which are encapsulated resource adaptors that are used via transaction codes (TRACOS). The mapping from services to CICS transactions is an important aspect during service definition and design and is supported by our approach.

From a development and release management perspective, it is interesting to describe how products are structured and how services are packaged. Products consist of clients and services and are the units of release planning. Service modules are the units of deployment and versioning. Services are typically implemented using J2EE. Some of the services are published as Web services. Aside from typical Web service clients using WSDL and SOAP, clients can also be implemented in Java accessing the services via IIOP; this is typically the case for Java web clients.

4 Service Life Cycle

This section describes part of a service life cycle, in particular activities and roles from service identification to deployment. Three roles that participate in the life cycle are the product manager, the service developer, and the administrator.

The product manager determines customer requirements, specifies a service for the business logic needed and is responsible for associating the service with a product. The developer is responsible for implementing the service and decides how the service is structured on a technical level, which might include the

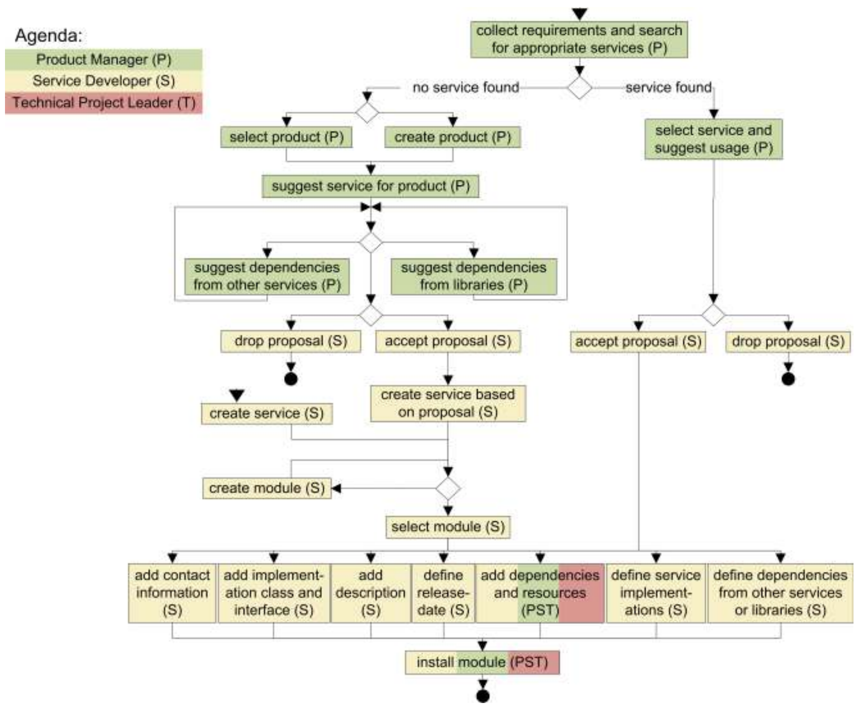
Agenda:
Product Manager (P)
Service Developer (S)
Technical Project Leader (T)
collect requirements and search for appropriate services (P)
no service found
service found
select product (P)
create product (P)
select service and suggest usage (P)
suggest service for product (P)
suggest dependencies from other services (P)
suggest dependencies from libraries (P)
drop proposal (S)
accept proposal (S)
accept proposal (S)
drop proposal (S)
create service (S)
create service based on proposal (S)
create module (S)
select module (S)
add contact information (S)
add implementation class and interface (S)
add description (S)
define release-date (S)
add dependencies and resources (PST)
define service implementations (S)
define dependencies from other services or libraries (S)
install module (PST)

Fig. 1. Service identification, creation and deployment

assignment of a service to a service module. After the development, a service enters a multistage testing and release process.

Figure 1 shows the process in more detail. First, the requirements for the new software are collected by the product manager (1), who organizes the required functionality into services and clients. During specification, the product manager checks whether a service with the required functionality already exists. If an appropriate service is found, this task is complete (3). If no appropriate service is found, the product manager creates a service proposal and assigns it to an existing product (2). The product manager might also propose already existing services that could or must be used for implementing the new functionality.

The created service proposals are forwarded to the development team for the specified product. The development team decides whether to reject, accept or modify a proposal (4). If a proposal is accepted (5), the service enters the development process. The process of service creation can also start with the service developer creating a service without a service proposal (5). This is usually the case for general services that are used in multiple products.

After development has started, the service can be assigned to a service module (6) and details of a service are defined. Examples include contact information, details of a particular service implementation, the service interface, an informal description, the planned release date, dependencies on other services or libraries, properties, and required resources (7). After the development of the service is finished and all information about a service is stored, the service can be installed as part of a service module (8).

A service module is first installed in a test zone by the service administrator. If the tests are successful, the service module is rolled out to the service operators, where it is tested with production data in the integration zone.

Finally, the service module and its services are deployed to the production zone. Since an operator might host services for different customers, a service module is installed and tested in an integration zone for each customer before it moves to the production zone.

5 Model

Figures 2-5 show simplified parts of the model for service metadata. The model consists of eight main areas. The central elements describing proposals, products, services, service implementations, libraries and policies are depicted in Figure 2. Elements for describing properties, resources, dependencies and installations can be found in Figure 3 and Figure 5; Figure 4 shows an example for properties.

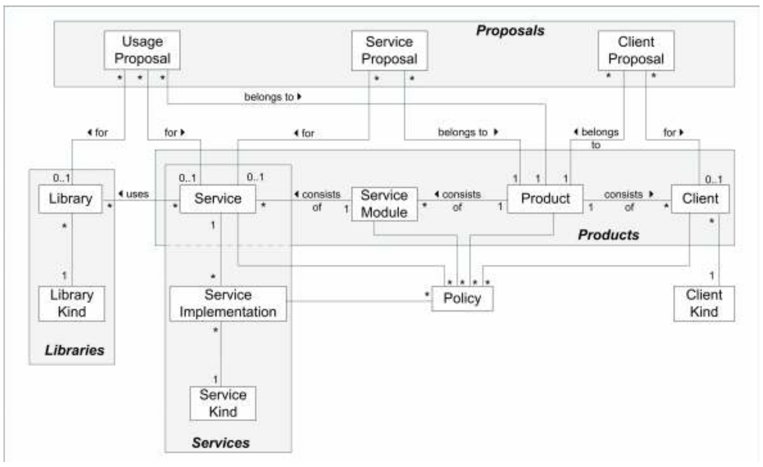

Fig. 2. Proposals, Products, Services, and Libraries

Proposals. Proposals can be created for services and clients (*ServiceProposal*, *ClientProposal*) or for the usage of services or libraries (*UsageProposal*). A proposal contains information about the proposed client, service, or library and status information that indicates whether the proposal has been accepted, realized or rejected.

Products. A product consists of clients and service modules. A service module contains one or more services. The version of a service module defines the version of all contained services. Different versions of a service are part of different service modules, which indicates that the service module is the smallest unit of deployment. If a service needs to be deployed and versioned separately, it can be packaged into a separate service module. A service module has an owner attribute that defines the organization responsible for service development and maintenance. The attribute *deactivationDate* shows how long the service module and its services are supported by the service producer. A service element might have one or more associated tags, which can be used for browsing and searching services in the repository.

Libraries. The *Library* element is used to model the relationship of services to other software artifacts that are not services. Libraries are software components that are not reusable via a network connection like services; instead, they are typically bound to the service (or service module) that uses the library. A library might represent an adapter to an external resource like a CICS transaction. The *LibraryKind* defines the properties and resources that are required to define a specific library. For example, a special instance of a *LibraryKind* describes CICS transactions.

Service implementations. The service implementation element describes how a particular service is implemented. In our case, a service can be implemented as a Web service, an EJB service or a JMS service. A service can have zero or more service implementations.

Policies. Policies define the access level and quality of service indicators for an SOA resource. A user name defines the person, user group or company and a role indicates the access or quality-of-service level. Policies can be defined on the level of products, service modules, clients or services.

Properties, resources, and contacts. Services and service implementations have various properties and resources. For example, a service property could describe the security mechanism for accessing the service. The documentation for a service is a typical service related resource while the EJB home interface of an EJB Service is a resource of the service implementation.

We chose a generic mechanism to store properties and resources of model classes. The model classes *Property* and *Resource* describe kinds of properties and resources (see Figure 3). Concrete values for properties and resources are stored in *PropertyInstances* and *ResourceInstances*. Properties and resources are connected to many classes in the model. The attribute *datatype* for a property

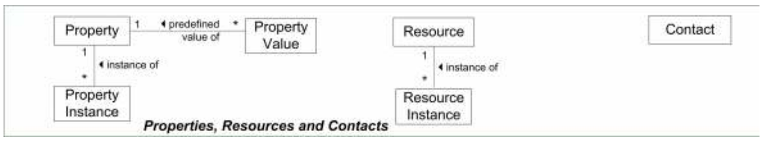

Fig. 3. Properties, Resources and Contacts

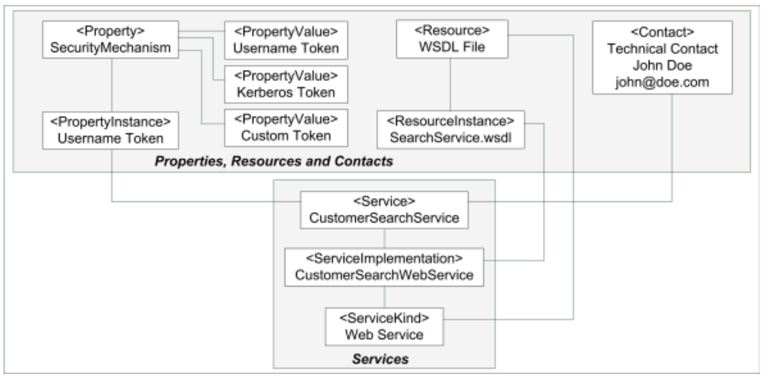

Fig. 4. Property Example

and the attribute *resourcetype* for a resource enable the validation of resources and properties. The attributes *min_occurence* and *max_occurence* define how many instances of a property or resource are required and allowed. *Property-Values* define possible values for the property. The contact element is used for describing persons responsible for specific products, clients and services.

An example for a property is the security mechanism which is used for the authentication of a service and the authorization of the usage of a service (see Figure 4). Possible values for the security mechanism are the Username Token [24] which authenticates the service consumer with a username and optionally a password to the service provider, the Kerberos Token [25], a generated token for authentication which is used by Windows, or custom binary security tokens. Those security mechanisms are stored as *PropertyValues* for the *Property* security mechanism. To show which security mechanism a service uses, a PropertyInstance with the chosen security mechanism is associated with the service. An example for a resource is the *WSDL-File* which is usually part of the description of a web service. In our example, the *CustomerSearchService* is implemented as a web service and this web service (*CustomerSearchWebService*) is associated with a resource instance containing a link to the actual WSDL file.

Instances (installations). Service modules and clients can be installed for different customers and operating zones. The operating zone can be test, integration or production. Installing a service means creating an instance in our model. Instances are also created for other elements, like clients, service modules, and service implementations (see Figure 5). Instance elements contain installation-specific information like the JNDI name of an EJB service.

The status of a service is derived from the installation location of the service module. If the service module is installed in a test zone, the service status changes to *test*; if the module is installed in the production zone, the status changes to *production*; if a new version of an existing service is created, the status of the old version is changed to *deprecated*.

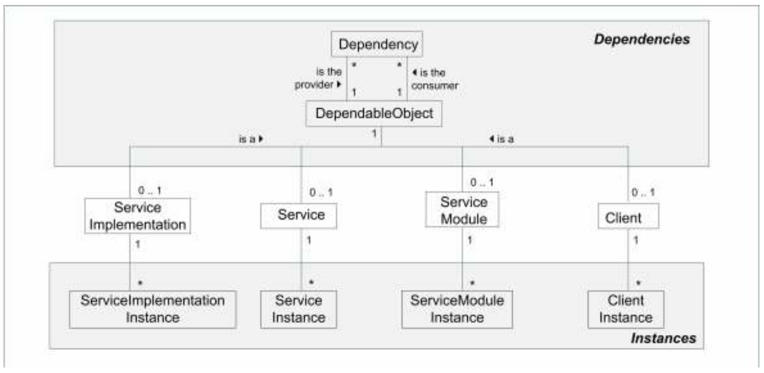

Fig. 5. Dependencies and Instances

Dependencies. Services can be used by other services and by clients. A *Dependency* is defined between two elements where one element is used by another one; one element provides information, the other one consumes this information (see Figure 5). On a more detailed level, dependencies can be defined for service implementations; on a less detailed level, dependencies can be defined for service modules. The model imposes no restrictions on the direction of the dependencies and allows incorrect dependencies such as a service using a client. Currently, such constraints have to be checked at the application level.

6 Tools

We provide two main tools, the Service Repository Console and the Service Browser, that are based on the model described in Section 5 and support the life cycle activities described in Section 4. Both tools were implemented on the basis of the Eclipse platform. This means that they can be used as standalone applications but can also be integrated into a software development environment.

The Service Repository Console is used for creating service proposals and service descriptions, for specifying service relationships, and for defining service installations. The Service Repository Console is depicted in Figure 6. The tool offers multiple views showing service proposals, services and service installations. The main area (see (2) in Figure 6) shows detailed information about the element selected in a view and can also be used for editing the element.

The Service Browser can be used for searching and browsing the service repository and for investigating service details, service relationships and service status. The browser, shown in Figure 7, offers a view for browsing products and services, a search view for searching services according to various criteria, a view for browsing the structure of a particular product, and a view for browsing client-to-service and service-to-service dependencies.

In the following, we illustrate the usage of the tools in the service life cycle presented in Section 4.

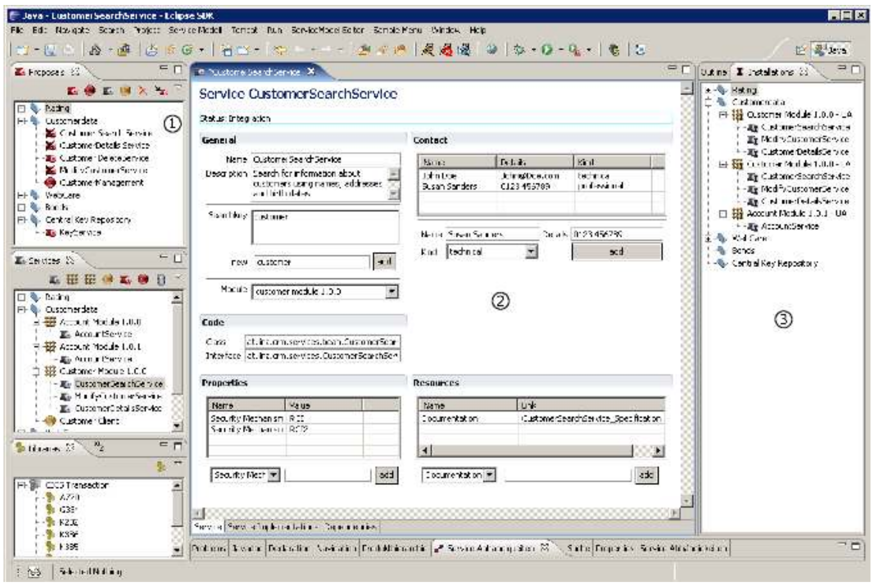
Java - CustomerSearchService - Eclipse SDK
Service CustomerSearchService
①
②
③
General
Contact
Code
Properties
Resources

Fig. 6. Service Repository Console

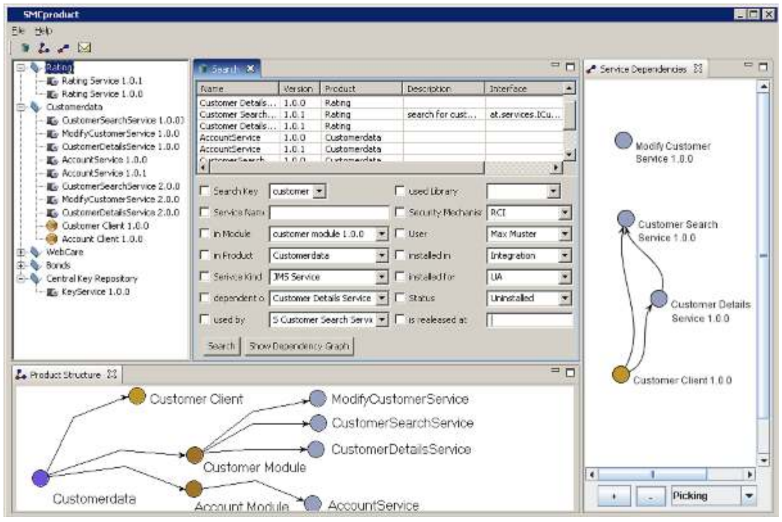

Fig. 7. Service Browser

The process is typically started by the product manager, who is responsible for defining a product and for determining product requirements, product development and other product related tasks through the complete product life cycle. The product manager can define products, clients and services using the repository console, specify new services and new clients, and propose their realization by creating service and client proposals (see (1) in Figure 6). The product manager can also define dependencies on existing services and thus propose the services that should or might be used by the new service. The icons of the displayed service proposals show whether the proposal is a client or a service proposal and whether it has been processed, accepted or rejected. The product manager can create a proposal from an existing, similar service or client. This activity is supported by the repository console. To find appropriate services, the product manager can use the service browser depicted in Figure 7. He can also use the service browser to study the existing services and their dependencies in more detail.

The proposals and their states can be viewed by the development team. If the team decides to implement a service, they must create a new service entity in the repository. Services can be created from a proposal or directly without a proposal. Services without proposals are typically created if the product has been restructured or if internal services are needed. If a service is created on the basis of a service proposal, the state of the proposal is automatically set to *accepted*. The proposal already contains suggested elements and dependencies on the new service. This information is automatically transferred to the service metadata and can be modified and extended by the service developer. At this stage, the service is already part of the repository and assumes the state *development*. Using this feature, other product managers and developers can keep track of planned services and release dates. The developer can also define service modules as the units of versioning and deployment and assign services to service modules, as shown in (2) in Figure 6. During development, the metadata of a service is continually extended. In particular, the actual dependencies, properties and resources of the chosen service implementation are provided. If a service is completely described, it can be released for installation in the test zone.

The Service Repository Console is also used for managing information about service installations (see (3) in Figure 6). The installations are organized by products and show the installed service modules along with version, location (operator), and installation zone. Installations can also be queried using the Service Browser. Together with the information about service dependencies, this information can be used to check which clients and services are affected when updating a service or when releasing a new version of a service.

7 Discussion and Further Work

We decided to create a very generic model. Properties and resources are stored as instances of the Property and Resource model elements; this facilitates adding and removing properties and resources without changing the model; only the data must be changed. Dependencies have also been modeled in a generic way. Dependencies can exist pairwise between services, clients, service modules and service implementations. The advantage of this approach is flexibility. The drawback is that certain constraints are not enforced by the model; for example, incorrect dependencies, such as a service using a client, must be prevented by the tools. Other approaches would be to specify such constraints using the Object Constraint Language (OCL) [20] in the model or to split the model into a metamodel and a model. Classes like *Property* and *Resource* could be elements of the metamodel and the concrete instances of a property such as the security mechanism could be elements of the model (i.e., the metamodel instance). This would lead to a richer but also heavier and more complex model with the advantage that more semantic information would be expressed in the model and the disadvantage that the model must be changed when data changes or new properties are needed. In our approach, all special model elements and constraints were implemented by the tools on the basis of the generic model.

Currently, the tools provide support for various activities in a service life cycle. For example: Services can be specified with service proposals; service proposals can be accepted or rejected; services can be added to products; dependencies on other services can be described and analyzed; services can be installed only if they are completely described. This means that the tools provide passive support for service governance activities by providing and validating useful information and by offering context-dependent functionality for the various roles and activities. The tools currently provide no active support for modeling a specific process or notifying specific users in case of process- or state-specific events. The latter could be realized through a notification mechanism (publish/subscribe) that informs users in case of changes. Additional useful and partly necessary enhancements include role-based authentication and authorization.

Since the tools are implemented as plug-ins, they can be integrated in an Eclipse-based workplace or development environment. Additional integration with development tools would be desirable. For instance, deployment information could be generated out of service metadata and parts of the service metadata (e.g., the class name of a service implementation) could be derived automatically from the development environment.

In Section 3, we posed 6 important questions that need to be answered for a successful SOA implementation. In the following, we outline how these questions are answered by our approach. Available services (q1) are described in the service element of the model and the Service Repository Console and the Service Browser offer lists with filter and grouping options to browse those services. The *Instance* concept of the model describes deployed services (q2) and attributes of those service instances hold information about the location of a service. The Service Repository Console provides a view containing a list of deployed services for each product. The person responsible for service development and evolution (q4) is described in the owner attribute of the service module element in the model. This information can be viewed and edited in the Service Repository Console. Dependencies between internal and external services and applications (q5 and q6) are modeled by the *Dependency* concept of the model. The Service Repository Console offers a list of services, dependent services and clients and required services. The Service Browser provides a graphical representation of the dependencies. The role of a service in a relationship (q3) is modeled in the role attribute in the *Dependency* concept. When creating a dependency between two elements in the Service Repository Console, this role attribute must be defined.

8 Related Work

Web Service Distributed Management (WSDM) [11] by OASIS comprises two sets of specifications: management using Web services (MUWS) and management of Web services (MOWS). The first specification addresses the use of Web services as the foundation of a distributed management framework [21]. The use of Web services for creating a governance or management infrastructure is not the focus of our approach. The second WSDM specification, MOWS, concentrates on managing Web services themselves, which usually means controlling and monitoring Web services as resources. This also differs from our approach, since we focus on controlling and governing the process of establishing a service-oriented architecture, which includes the process of developing a service, but not controlling the service itself. In fact, a service management approach is complimentary to governing service life cycle activities and might provide valuable information for governance activities. Both approaches are needed to successfully install an SOA governance framework.

The Common Information Model (CIM) [12] by the Distributed Management Task Force (DMTF) describes managed elements across an enterprise, including systems, networks and applications [21]. CIM contains a metamodel and generically managed object classes. The purpose of CIM is to integrate different management approaches and to allow the exchange of management information between systems throughout the network. DMTF does not intend to support the reuse of services and the governance of service life cycle activities.

The MNM Service Model [9][10] is a reference model for service management. The model is intended to describe a typical service life cycle. The life cycle phases described include design, negotiation, provisioning, usage and deinstallation. The model includes typical roles participating in the service life cycle, such as user, customer and client. It also identifies typical interactions and life cycle activities, such as contract management, problem management, security management, usage, customer care and change management. The interactions and activities are not described in detail, however; their refinement is stated as an open research issue. The MNM Service Model is intended to provide a conceptual model for describing the relationship of artifacts, roles and activities in the service life cycle. It can be used to analyze service-oriented solutions and architectures and as a conceptual model for implementing governance and management processes. It is not intended for storing actual service-related information or as a basis for governance and management tools.

UDDI [13] is a specification for (Web) service registries. It provides a standardized API for publishing and discovering services and defines registry entries. UDDI supports especially Web services through tModel registry entries. UDDI can be used to implement private and public registries. A public registry is a global directory for business services and is intended to support global service reuse. A private registry supports service reuse within one organization and the establishment of business-to-business transactions with selected partners. The main goal of UDDI is service advertisement and discovery. Its primary aim is not governance of service life cycle aspects such as service definition, implementation and evolution, though it could be used to store certain information for such activities. Contrary to service repositories described below, a UDDI registry stores only information that is needed for service (re)use, not for service development and maintenance.

Manolescu and Lublinsky [14] describe service repositories as a solution for the problem of finding service information and for supporting design, implementation, testing and reuse of services. They identify roles using the repository, such as service developer, service designer, architects and infrastructure specialists. The repository is intended to support the inception, design, implementation, deployment, enhancements, versions and discovery of services with the goal of reuse. Dependency management, versioning and change notification are mentioned as functionalities that should be supported by a service repository. This is an indication that service repositories are seen as a central element for SOA governance and reuse.

A number of commercial registries/repositories exist. Most of them are based on UDDI, some on ebXML. Examples include Systinet 2 [7], Centrasite [8] by Software AG, X-Registry by Infravio [16] and the SUN Service Registry by SUN Microsystems [23]. The functionality typically provided by such commercial approaches includes service discovery, dependency management, change notification, authentication and identity management, policy management, and federation with other repositories. In some areas the functionality clearly exceeds the functionality provided by our approach, including features such as security (authentication) and change notification. Some features are similar, like support for service description, service discovery and dependency management. Some registries like the Sun Service Registry support phased deployment to different zones, which is also supported by our approach. Some products support additional features such as impact analysis and support for tracking and analyzing processes. Competing standards for repository interoperability are emerging. Examples include the Governance Interoperability Framework (GIF) [2] and SOALink [22]. Noteworthy features of our approach are support for product management, versioning and evolution, IDE and workplace integration, a rich user interface, different kinds of services, and implicit state tracking of services. The last feature implicitly derives the life cycle status of a service (development, test, active, deprecated, etc.) from the metadata provided. The status of a service cannot be changed if not the all elements of required information have been provided to advance the service to the next state in the life cycle.

9 Conclusion

We have presented an approach for supporting service reuse and service life cycle activities. The approach is based on a generic model for describing SOA artifacts in the various stages of the service life cycle. The model contains elements for describing client and service proposals initiated by the product manager. These proposals are eventually transformed to client and service descriptions and instances. The model also contains elements for products and service modules, which are used for representing the units of product management and the units of versioning and deployment, respectively. Two tools are presented that use a service repository that is based on the model presented. The Service Repository Console is used for creating service proposals and service descriptions, for specifying service relationships, and for defining service installations. The Service Browser is used for searching and browsing the service repository and for investigating service details, service relationships and service status. The tools can be used for service reuse and for coordinating and governing the activities of product managers, developers and administrators. Together the repository and the tools represent an important step towards SOA governance and management.

Acknowledgments. The presented approach was developed as part of a joint project of the Software Competence Center Hagenberg (SCCH) and the GRZ-IT Center Linz. Both authors were members of the SCCH project team. We wish to thank Hermann Lischka, Thomas Kriechbaum and Johannes Mayr from the GRZ IT Center Linz and Thomas Ziebermayr from the SCCH for their cooperation and support.

References

1. Borges, B., Holley, K., Arsanjani, A.: Service-Oriented Architecture. WebServices Journal 4(9) (2004)
2. Systinet: Governance Interoperability Framework (2006), Retrieved June 29, 2006, from `http://www.systinet.com/products/gif/overview`
3. Motahari Nezhad, H.R., Benatallah, B., Casati, F., Toumani, F.: Web service interoperability specifications. IEEE Computer 39(5), 24–32 (2006)
4. McGovern, J., Ambler, S.W., Stevens, M., Linn, J., Sharan, V., Jo, E.K.: A Practical Guide to Enterprise Architecture. Prentice Hall PTR, Englewood Cliffs (2003)
5. Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice-Hall, Englewood Cliffs (2005)
6. IBM: Customer Information Control System (2006), Retrieved June 26, 2006, from `http://www-306.ibm.com/software/htp/cics/`
7. Systinet: Systinet 2 Overview. (2006), Retrieved June 26, 2006, from `http://www.systinet.com/products/systinet_2`
8. Software AG: Software AG brings governance to SOA (2006), Retrieved June 26, 2006, from `http://www.softwareag.com/Corporate/Solutions/soa_governance/default.asp`
9. Garschhammer, M., Hauck, R., Kempter, B., Radisic, I., Roelle, H., Schmidt, H., Hegering, H.-G., Langer, M., Nerb, M.: Towards Generic Mananagement Concepts: a Service Model Based Approach. In: 7th IEEE/IFIP Symposium on Integrated Network Management, Seattel, Washington, USA (2001)

10. Garschhammer, M., Hauck, R., Kempter, B., Radisic, I., Roelle, H., Schmidt, H.: The MNM Service Model - Refined Views on Generic Service Management. Journal of Communications and Networks 3(4) (2001)
11. OASIS: OASIS Web Service Distributed Management (WSDM) (2006), Retrieved June 26, 2006, from `http://www.oasis-open.org/`
12. DTMF: Common Information Model (CIM) Standards (2006), Retrieved June 26, 2006, from `http://www.dmtf.org/standards/cim/`
13. OASIS: OASIS Universal Description, Discovery and Integration (2006), Retrieved June 26, 2006, from `http://www.uddi.org/`
14. Manolescu, D., Lublinsky, B.: SOA Enterprise Patterns - Services, Orchestration and Beyond (DRAFT). To by published by Morgan-Kaufman Publishers in 2007 (2006), Retrieved 29, 2006, from `http://orchestrationpatterns.com/`
15. Brauer, B., Kline, S.: SOA Governance: A Key Ingredient of the Adaptive Enterprise. HP & Systinet White Paper (2005), Retrieved June 20, 2006, from `http://www.systinet.com/resources/white_papers/`
16. Infravio: X-Registry Platform Overview (2006), Retrieved June 26, 2006, from `http://www.infravio.com/products/`
17. W3C: SOAP Version 1.2 Part 1: Messaging Framework. W3C Recommendation 24 June 2003 (2006), Retrieved June 28, 2006, from `http://www.w3.org/TR/soap12-part1/`
18. W3C: Web Services Description Language (WSDL) 1.1. W3C Note 15 March 2001(2006), Retrieved June 28, 2006, from `http://www.w3.org/TR/wsdl/`
19. OASIS: Web Services Business Process Execution Language, Version 2.0. Committee Draft, 17th May, 2006 (2006), Retrieved June 28, 2006 from `http://www.oasis-open.org/`
20. OMG: Object Constraint Language (2006), Retrieved June 28, 2006, from `www.omg.org/docs/ptc/05-06-06.pdf`
21. Papazoglou, M.P., van den Heuvel, W.-J.: Web Services Management: A Survey. IEEE Internet Computing (2005), (November/December 2005)
22. Soa Link Organization: SoaLink (2006) Retrieved June 29, 2006, from `http://www.soalink.com/`
23. Sun Microsystems: Effective SOA Deployment using an SOA Registry Repository, A Practical Guide (2005), Retrieved June 29, 2006, from `http://www.sun.com/products/soa/registry/`
24. OASIS: Web Services Security Username Token Profile 1.1. OASIS Standard Specification, 1st February, 2006 (2006), Retrieved October 28, 2006, from `http://www.oasis-open.org/committees/download.php/16782/wss-v1.1-spec-os-UsernameTokenProfile.pdf`
25. OASIS: Web Services Security Kerberos Token Profile 1.1. OASIS Standard Specification, 1st February, 2006 (2006), Retrieved October 28, 2006, from `http://www.oasis-open.org/committees/download.php/16788/wss-v1.1-spec-os-KerberosTokenProfile.pdf`