Fraunhofer Institut
Experimentelles
Software Engineering

Asset Scoping: Identification of Reusable Software Components

Defining Service Components

Authors:
Joachim Bayer
Theresa Lehner
Dirk Muthig

PESOA
Process Family Engineering in Service-Oriented Applications

BMBF-Project

IESE-Report No. 125.06/E
Version 1.0
October 30, 2004

A publication by Fraunhofer IESE

Fraunhofer IESE is an institute of the Fraunhofer Gesellschaft.
The institute transfers innovative software development techniques, methods and tools into industrial practice, assists companies in building software competencies customized to their needs, and helps them to establish a competitive market position.

Fraunhofer IESE is directed by
Prof. Dr. Dieter Rombach (Executive Director)
Prof. Dr. Peter Liggesmeyer (Director)
Fraunhofer-Platz 1
67663 Kaiserslautern

PESOA-Report No. 12/2004

PESOA is a cooperative project supported by the federal ministry of education and research (BMBF). Its aim is the design and prototypical implementation of a process family engineering platform and its application in the areas of e-business and telematics.
The project partners are:

- DaimlerChrysler Inc.
- Delta Software Technology Ltd.
- Fraunhofer IESE
- Hasso-Plattner-Institute
- Intershop Communications Inc.
- University of Leipzig

PESOA is coordinated by
Prof. Dr. Mathias Weske
Prof.-Dr.-Helmert-Str. 2-3
D-14482 Potsdam

www.pesoa.org

Abstract

Software systems provide to their users a number of accessible services. For service-oriented applications the services an application provides to its users are the major driver for understanding the requirements on the application and also for developing it. A service-oriented application can then be specified by selecting required services. The application is realized by combining all required services using a service-oriented architecture that maps services to software components.

The goal of the PESOA project is to design and implement a platform for families of related service-oriented applications. The envisioned platform is used to manage process variants for families of service-oriented applications and to enable the process-based instantiation of such service-oriented application families.

The goal of asset scoping is to define techniques for the development and adaptation service components. Service components are components that can be deployed on a platform for families of related service-oriented applications and that provide a number of services. Using processes the different services that are provided by the service components deployed on the PESOA platform are combined.

Keywords: PESOA, Services, Service-oriented Applications, Service Components, Scoping.

Table of Contents

1	**Introduction**	**1**
1.1	Project Context	1
1.1.1	Goal	1
1.2	Outline	2
2	**Service-Oriented Applications**	**3**
2.1	Services	3
2.2	Service-Oriented Architecture	3
3	**Service Components**	**6**
3.1	Service Model	8
3.1.1	Service Specification	8
3.1.2	Service Realization	11
3.2	Scoping Service Components	12
3.3	Service Component Model	15
3.3.1	Service Component Specification	15
3.3.2	Service Component Realization	16
4	**Conclusion and Outlook**	**18**
References		**19**

1 Introduction

1.1 Project Context

PESOA is a cooperative project financed by the German federal ministry of education and research (BMBF). The goal of the PESOA project is to design and implement a platform for families of related service-oriented applications. The envisioned platform is used to manage process variants for families of service-oriented applications and to enable the process-based instantiation of such service-oriented application families. This goal is addressed by enhancing the approved technologies from the area of domain engineering, product line engineering, and software generation with new methods from the area of workflow management.

1.1.1 Goal

Software systems provide to their users a number of accessible services. For service-oriented applications the services an application provides to its users are the major driver for understanding the requirements on the application and also for developing it. A service-oriented application can then be specified by selecting required services. The application is realized by combining all required services using a service-oriented architecture that maps services to software components.

The goal of asset scoping is to define techniques for the development and adaptation service components. Service components are components that can be deployed on a platform for families of related service-oriented applications and that provide a number of services. Using processes the different services that are provided by the service components deployed on the PESOA platform are combined.

Services must be packaged as components, that is, in a form that they can be deployed and thus invoked. Each service could be packaged in a single component. This would, however, lead to unnecessary effort for developing, maintaining, and deploying the components. Additionally, the fact that services are similar could not be exploited using product line techniques. For these reasons service components are developed that provide a number of services. The problem addressed by asset scoping is the definition of service components that provide a number of services in a way that they provide a number of coherent and self-contained services.

In this report, we propose an approach to scope service components based on well-established existing practices.

1.2 Outline

The remainder of this report is structured as follows. In chapter 2, the context of this work is presented in detail. To this end, we first discuss services and service-oriented architectures to understand the requirements posed on service components. In chapter 3, we present a technique to develop and document services, propose an approach for scoping service components, and, finally, present our approach to develop and document service components.

2 Service-Oriented Applications

2.1 Services

There are numerous definitions of the term service, a widely used one is the following: an act or performance offered by one party to another. Although the process may be tied to a physical product, the performance is essentially intangible and does not normally result in ownership of any of the factors of production [Lovelock96].

To us, a service is some functionality that can be invoked using a well-defined interface. A service can be situated anywhere in the world. In order to be visible and accessible from everywhere, service providers generally enroll their services at a service broker that registers and publishes the services. Web services are a typical form of this type of service provision, where web-based protocols and mechanisms are used by software elements to enable a standardized communication with these services.

2.2 Service-Oriented Architecture

Figure 1 shows the typical pattern for service-oriented architectures. It consists of three components. A service provider offers a number of services. These services are published at a service broker component that collects services from a number of service providers, registers and publishes them in a service registry. The service registry can be used by a service consumer to find an appropriate service. If the consumer finds a service, a direct connection between the service consumer and the service provider is established and the requested service is bound to a provided service.

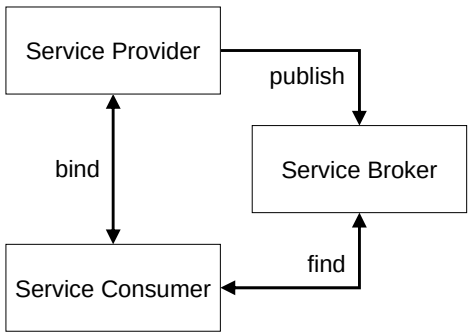

Figure 1: Service-Oriented Architecture

For process-based service-oriented applications, multiple services are combined to processes at the service consumer side. An interaction control component thereby coordinates the execution and combination of the different services. Workflow or process execution engines can for example be used as interaction control components. Additionally, context information can be taken into account in the interaction control component. This context information includes information about the user's location for mobile applications or information about the user's device and the device's capabilities. The interaction control component is also responsible for providing a user interface.

Service providers usually provide multiple services that are packaged as components to bring together and provide at once a number of related services. A prominent reason for doing this is to reduce communication overhead since services that are located in the same (non-distributed) component require less communication overhead than services that are located in different components. Another reason for packaging services is that the development, maintenance, and deployment of the services are simplified when several services are handled together. Product line engineering approaches can be used to exploit the commonalities among related service and to reuse large portions of the developed, maintained, and deployed artifacts.

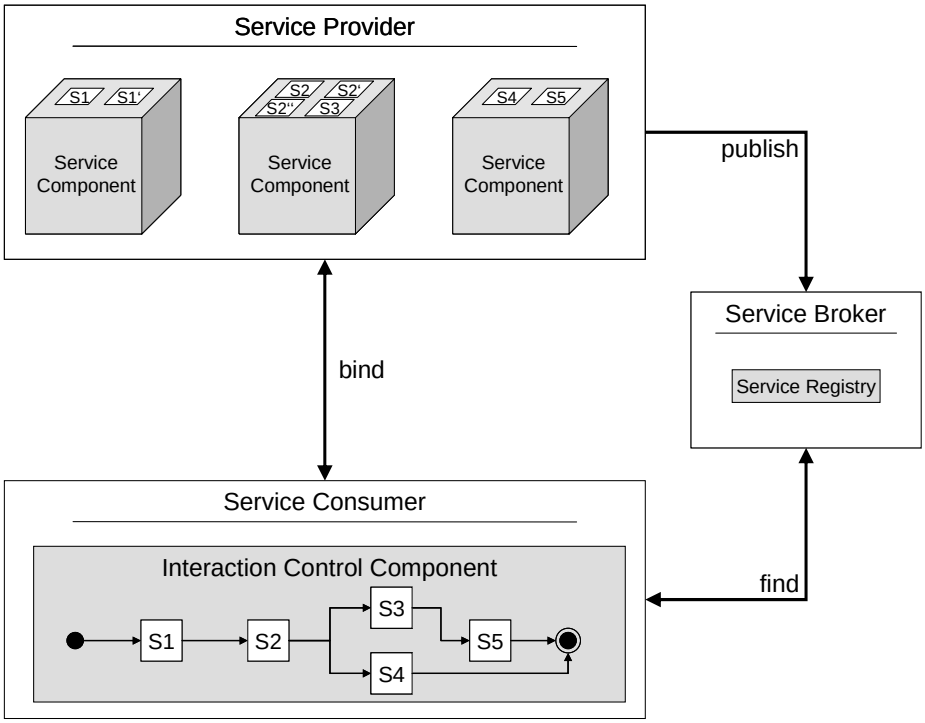

Figure 2: Service-Oriented Architecture for Process-Based Service-Oriented Applications

The next chapter presents how service components that provide a number of services in a service-oriented architecture can be developed.

3 Service Components

To develop service components, the software development strategy must be adapted to reflect service-orientation. A service component can be defined as a component that provides a number of services via standard interfaces and that is deployable in a component infrastructure. An approach for developing service components should be an extension of existing practices. In the following, we adapt the widely used V-Model [Broehl95] towards service-orientation to yield a principal development strategy and combine it with the KobrA method [Atkinson01], a component based software development approach, to develop service components. The resulting approach is described in the following.

According to the V-Model, software systems are described at different levels of abstraction: at the highest level of abstraction, the problem is described, followed by the user and developer requirements, respectively. The next level of abstraction is the system design describing the solution by breaking down the functional requirements into sub systems and components. For each of the components, a design is given, as well as an implementation using some implementation technology.

Service-orientation changes the product model of the V-Model. The service-oriented V-Model is depicted in Figure 3. The first two levels of adapted, service-oriented V-Model are, like in the original V-Model, also problem description and user requirements. At these levels, the processes, as well as the service-oriented applications to be supported are elicited and documented as problem description and user requirements. These are used to validate the used and the usable service-oriented application, respectively.

The next two levels of abstraction incorporate services into the model. Services are described at two levels of abstraction, service specification and service realization. These terms stem from the KobrA method. The KobrA method is a component-based method for product line development that we use as a basis to capture service components and consequently also for services. The service specification captures the externally visible characteristics of a service, that is, especially the functionality the service provides and the interface to invoke the service. The service realization captures the internal characteristics of a service, for example internal data structures or other sub-services that are used to provide a service. The service specification is used to validate the usable service that is bound to a requested service.

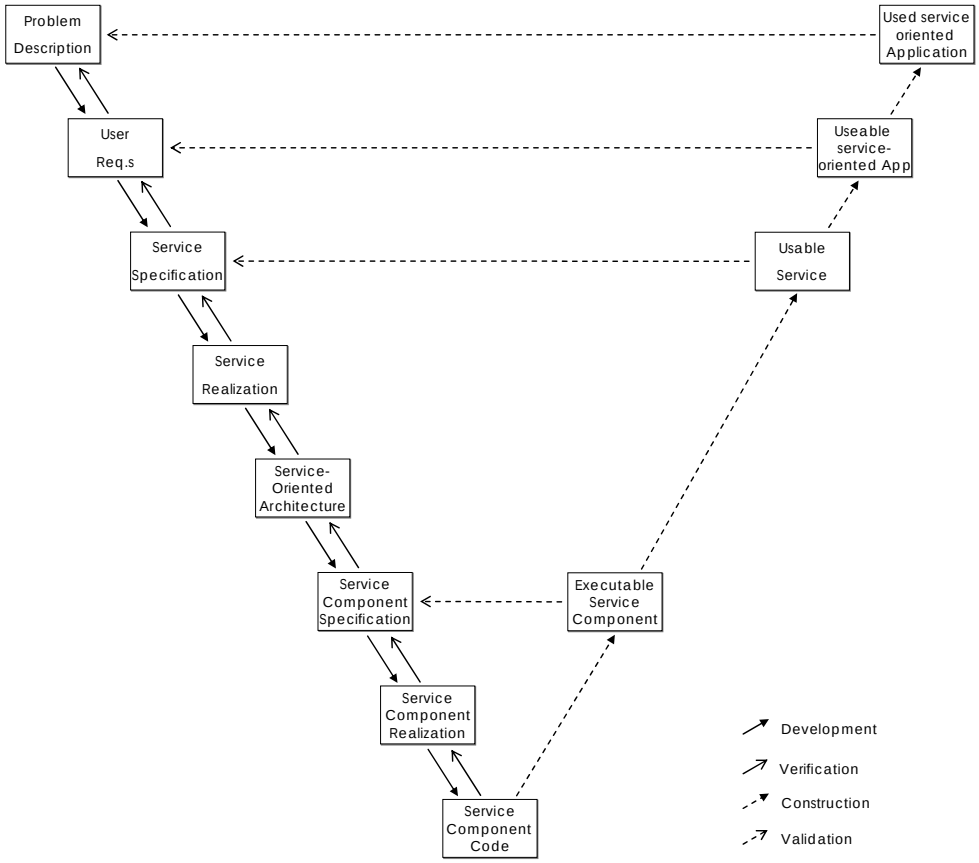

Figure 3: Service-Oriented V-Modell

The next level of abstraction in the service-oriented V-Model is the service-oriented architecture that describes how the services are used in processes at the service consumer side and how the services are distributed onto service components at the service provider side.

The identified service components are then specified, realized, and implemented as component code. We use a slightly adapted version of the KobrA component model to capture service components. Since the KobrA method supports the model-driven architecture approach, the component code can be developed using MDA techniques [MDA]. The service component specification is used to validate executable service components.

The focus of this report is on the packaging of services as service components. That is, the focus is on addressing the following problem: once the services to be provided are elicited, specified, and realized, which services should be packaged together in service components? To address this question, the following three sections present the service model to capture services, the scoping of ser-

vice components, and the transformation of the scoped logical components to technical components, respectively.

3.1 Service Model

A service is some functionality that can be invoked using a well-defined interface. Service can be either atomic, simple services or they can be composed of other services. Composed services are services as well and thus simple and composed services are modeled similarly. As described above we use component-based techniques to develop service-oriented applications. Therefore, component specification techniques are used to specify services. To this end, a service is considered a component with exactly one method. This approach makes services concrete software development entities and also enables the usage of component-based software engineering approaches to specify and design them. Additionally, the composition and integration of services can be handled by component-based techniques. We use an adapted version of the KobrA method to specify and design services [Atkinson01]. The KobrA method supports the principle of encapsulation by modeling components in terms of a specification, describing the requirements that the component is expected to fulfill and the expectations that the components places on its environment, and a realization that describes the design by which a component fulfills these requirements. The specification and realization of a KobrA component is composed of several inter-related models, describing complementary aspects of the component. For the specification, the separately modeled aspects are the component's functionality, its structure and behavior, for the realization aspects are the components internal structure, as well as its dynamic aspects.

Following the approach the KobrA method takes for modeling components, we describe services at two levels of abstraction as well. The two levels are the service specification that captures the externally visible characteristics of services and the service realization that captures the internal characteristics of a service. The service specification and the service realization are presented in more detail in the following two subsections.

3.1.1 Service Specification

The service specification describes the functionality a service provides and the interface by which the service's functionality can be invoked. Consequently, a service is also specified by means of at least two related models. A third model is optional and not always necessary. An interface model captures the interface of the service and the respective signature. We use simplified UML class diagrams as interface models. For simple services, the class diagrams contain only one class that in turn only contains one method (i.e., the service). A stereotype <<service>> is used to depict a service in the class diagrams containing the in-

terface model. The reason for using UML class diagrams for describing simple services is that services composed from multiple simple services and also service components can be described similarly.

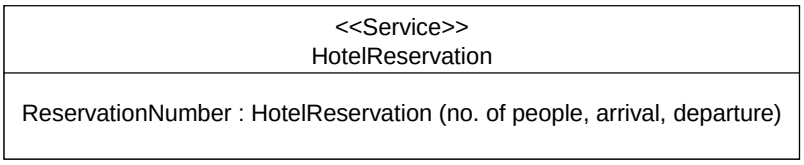

Figure 4: Example Service Hotel Reservation

Figure 4 presents an example. It specifies the interface of the HotelReservation service that takes the number of people, the arrival date, and the departure date as input uses and returns a ReservationNumber.

The second model used to specify services is the functional model that describes the functionality by means of describing the externally visible effects of using the respective service. In the KobrA method, operation schemata are used to capture functional models. Applied to services, operation schemata capture the pre conditions and post conditions of a service, as well as manipulated data. The following table shows the attributes for service operation schemata.

Service	**Name of the service**
Description	Informal description of the functionality the service provides
Receives	Information input to the service
Returns	Information returned by the service
Reads	Information accessed by the service
Changes	Information changed by the service
Assumes	Weakest pre-condition on the state of the service and on the inputs that must be true to guarantee the post-condition
Results	Strongest post-condition that becomes true after service execution with true assumes clause

The second table presents the operation schema for the example service Hotel Reservation. In this example we use an informal textual description to describe the attributes. Other description languages, like WSML as used in WSMO [WSMO] or XML as used in WSDL can be used as well [WSDL].

Service	Hotel Reservation
Description	Reserve a hotel room for a given number of people and period of time
Receives	no. of people arrival date departure date hotel name max price
Returns	Reservation number
Reads	-
Changes	-
Assumes	- no of people > 0 - departure date > arrival date - max price > 0
Results	If there is a room available in hotel hotel name that is free from arrival date to end date and that costs less than max price then a reservation number is sent back to the service requester. Otherwise a message "no such room available" is sent back to the service requester.

There is sometimes a third optional model used in the specification of services. This third model is a behavioral model that captures externally visible state changes that occur when a service is consumed. Since single services are usually stateless, the behavioral model is not always necessary. For services that have states, state chart diagrams are used to capture the respective behavioral model.

Figure 5 presents the behavioral model using state chart diagrams of a Travel Booking service that reserves the fight, the hotel, as well as a rental car; thereby it traverses the states Flight Booked, Hotel Booked and Rental Car Booked.

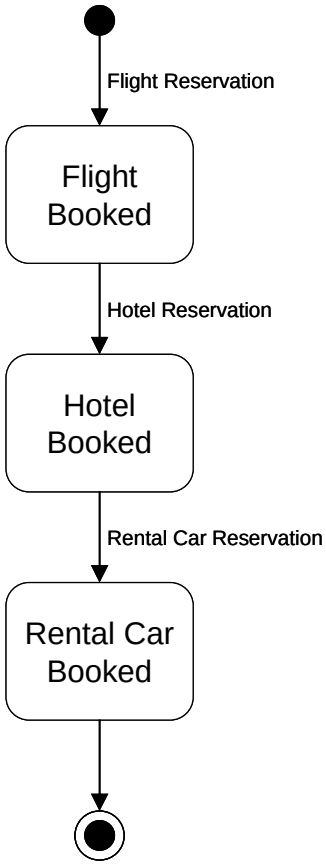

Figure 5: Example State chart diagram for Service Travel Booking

3.1.2 Service Realization

The service realization is required to develop deployable components that implement and this way provide the possibility to invoke the service. The service realization describes the internal design of a service and thus captures how the service fulfills its specification and what other services it requires. Again, we use the models used in the KobrA method to realize components. Therefore, a service is realized by means of three different models.

The realization structural model refines the interface model by adding information on internal data structures and on services that are acquired to provide the service under consideration in the case of a composed service. Being a refinement of the interface model, the realization structural model is a UML class diagram as well.

As shown in Figure 6 the example service Hotel Reservation uses the data structure Reservation to perform the reservation.

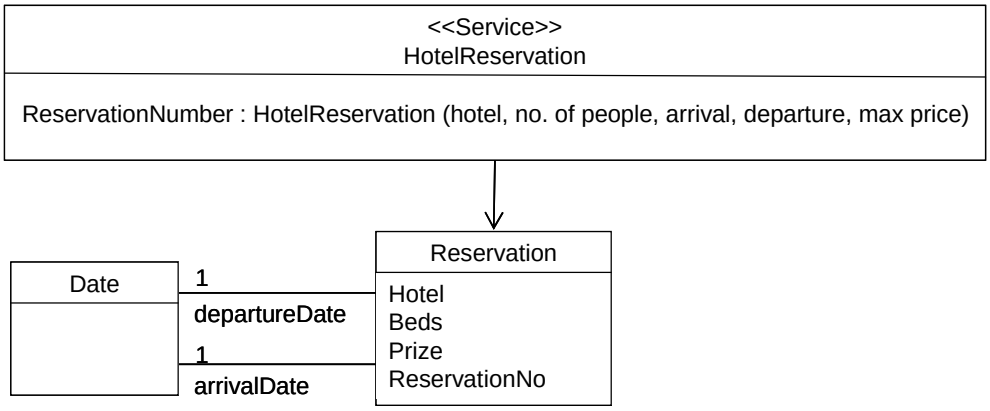

Figure 6: Example Service Realization Structural Model

The two other models used to realize services both describe the dynamic aspects of the service from different perspectives. The activity model uses activity diagrams to capture the sequence of activities performed during service provision. The interaction model on the other hand uses collaboration diagrams to capture the interaction between a service and its acquired services. Usually, it is sufficient to use only one of the two models.

Figure 7 specifies the interaction of the example service Travel Booking using a collaboration diagram. Travel Booking acquires the Flight Booking service, the Hotel Reservation service and the RentACar service.

3.2 Scoping Service Components

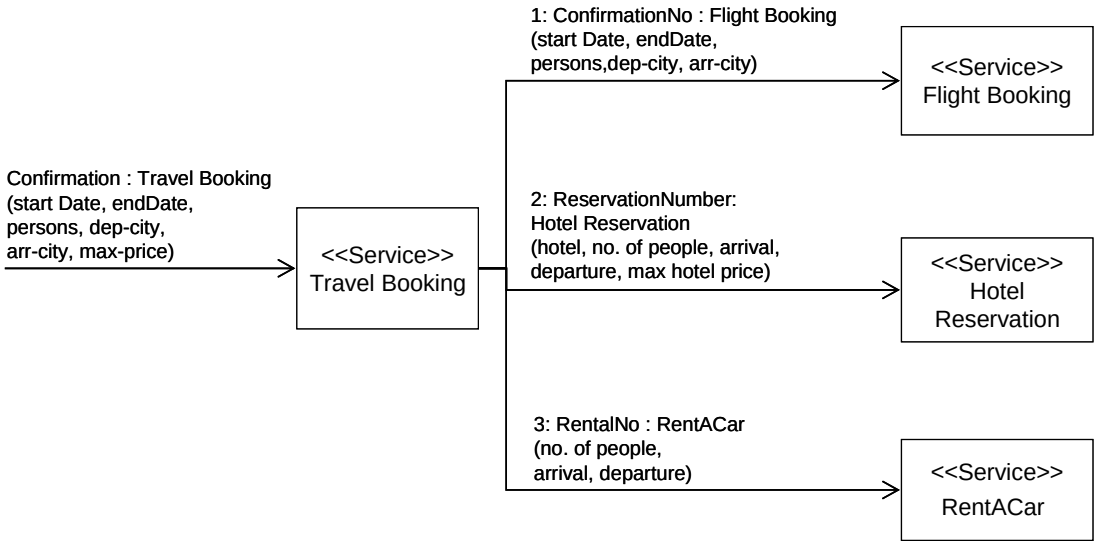

Figure 7: Example Service interaction/activity model

Once services are specified and realized, the service-oriented architecture determines how the services are used in processes at the service consumer side and how the services are distributed onto service components at the service provider side. In this report, we concentrate on the service provider side and, therefore, address the issue of defining service components that provide multiple services. Service providers package a number of services as components to bring together and provide at once a number of related services. A prominent reason for doing this is to reduce communication overhead since services that are located in the same (non-distributed) component require less communication overhead than services that are located in different components. Another reason is a simplification in the development, maintenance, and deployment of the services that is achieved since several services are handled together. Often service providers provide a number of similar services, for example similar, related services that have different prices and functionality. In these cases, product line engineering approaches can be used to exploit the commonalities among related service and to reuse large portions of the developed, maintained, and deployed artifacts.

Service components provide a number of services. In order to benefit from the common handling of multiple services in one component and from reuse, the components must be defined in a way that the services fit well together. The obvious question is how to achieve that. We propose to base the decision on whether a number of services should be packaged in a service component on coupling and cohesion. This means that the goal of scoping service components is to find for a number of specified and realizes services the set of service components for which the cohesion within the different service components is as high as possible and the coupling among the service components is as low as possible. The coupling among services can be measured by measuring the interaction between services (i.e., the invocation of one service by another) and the common usage of data structures by multiple services.

As described in the last section, services are specified in isolation, that is, the functionality a service provides, as well as its interface are defined without taking into account other services. The realization of a service covers three different aspects the behavior of the service, its internal structure, and the interaction of the service with other services. The behavior of the service describes the algorithm by which a service provides its functionality. The internal structure describes the data structures the service requires and the interaction of the service with other services determines the sub-services a service requires. Both are given in an isolated way meaning that required services and data structures are defined for each service in isolation without taking into account the requirements of other services. The key idea of our approach for scoping service components is to take the isolated information from a number of services and investigate it in a global way.

Service components are scoped by optimizing the cohesion within a service component and the reducing the coupling among the different service components of one service provider. The starting point for scoping service components is a number of services that are specified and realized as described in the previous section. This especially means that for each service the data structures, as well as the interaction with other services are documented. The scoping of service components then encompasses the following two activities:

- Structural optimization: Services use data structures internally to store and retrieve information manipulated by the services. In the service realization, the data structures are documented in the structural realization model. The data structures are, however, given for each service in isolation.
  The goal of the structural optimization is to consolidate the data structures for a number of services in order to make an informed decision on how to distribute the service onto service components. The underlying idea here is to put together services that work with the same data to reduce the need to send data between different service components.
  The result of the structural optimization is a distribution of services onto service components that collect services that work on the same data structures.

- Interaction optimization: Services often delegate certain aspects of their task to other services. In the service realization, these delegations are documented, again only in an isolated way.
  The goal of the interaction optimization is, therefore, to collect services in service components in a way that reduces the service interaction among components and thus keeps the service interaction within a service component. This reduces communication overhead.

The two activities are not meant to be conducted in a sequential way but rather in a way that they provide input to the other since the structure of the components as well as their interaction need to be optimized at the same time and the two influence each other.

It is important to note that for providing services it is not necessary to distribute the service to components in a way that each service is put in exactly one component. Rather replication of services is no problem since it is possible to have one service in multiple components.

3.3 Service Component Model

The scoped components are modeled using the KobrA component model [Atkinson01]. The KobrA method is an approach for developing component-based applications. In the KobrA component model components are described at two levels of abstraction, a specification, which defines the component's externally visible properties and behaviors, and thus serves to capture the contract that the component fulfils and a realization, which describes how the component fulfils this contract in terms of contracts with other, possibly lower level components. Figure 8 shows the KobrA component model. In the following, the specification and realization of components is described in more detail.

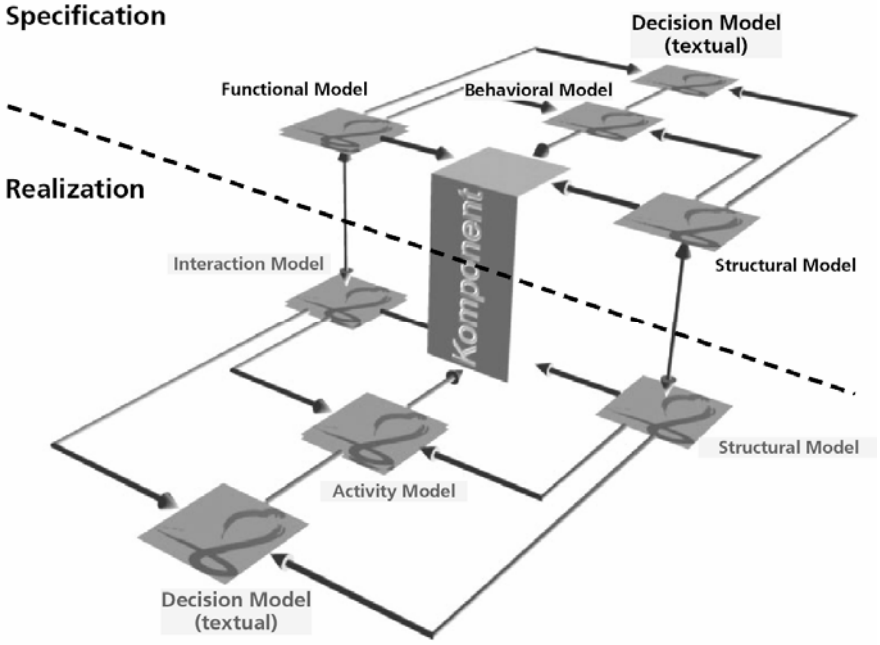

Figure 8: KobrA Component Model

3.3.1 Service Component Specification

A service component collects a number of services and provides them. Therefore, the different services are modeled as operations in the respective service component specification. The service component specification comprises a set of models that collectively describe the externally visible properties of a service component. As such, the specification can be viewed as defining the interface of a service component and describing the services a component provides. The specification of a service component is comprised of the following four models:

- Structural Model: Captures the nature of the classes and relationships by which a component interacts with its environment, as well as any structure of the component that is visible at its interface. The structural model consists of a number of UML class diagrams that captures the externally visible structural elements a subject component interacts with and a number of UML object diagrams that capture the externally visible parts of run-time configurations of a component and the components it acquires.
- Behavioral Model: describes the reaction of the component to external stimuli using UML statechart diagrams. The behavioral model is optional for service components.
- Functional Model: addresses the functionality of a component by describing the externally visible effects of the services provided by the component. The behavioral model contains an operation schema for each of the services a service component provides.
- Decision Model: The structural, behavioral and functional models constitute the specification models for a component. If the component is a generic product line component, an optional decision model contains information about how the models change for the different instances of the product line component.

3.3.2 Service Component Realization

The goal of component realization is to create a set of models that collectively describe the private design of a service component. As with all design, the basic requirement is that the realization must realize the service component's specification. A service component’s realization is comprised of the following four models:

- Structural model: captures the classes and relationships from which the component is realized, as well as its architecture. The realization structural model is a refinement of the specification structural model. It consists of a number of UML class diagrams that capture the structural elements a service component interacts with and a number of UML object diagrams that capture the run-time configurations of the service component and the services components it interacts with.
- Activity model: covers the realization of the functional aspects by describing the algorithms by which the services of the component are realized using UML activity diagrams.
- Interaction model: provides different aspects on the algorithms used to realize operations, from the perspective of instance interactions rather than flow control (as in the execution model). UML collaboration diagrams are used in the interaction model.

- Decision model: as in the specification, the optional decision model describes the model changes for the different instances of a product line component.

4 Conclusion and Outlook

This report presented one part of a development approach for creating service-oriented applications. This part of development is the scoping that supports the packaging of services in deployable service components. Our scoping approach is based on the reduction of communication overhead by optimizing the common use of data structures and by reducing the interaction among service components.

We also presented techniques for modeling the input of scoping, that is, the services and the resulting service components. These techniques are based on a well-established approach to component-based application development, the KobrA method.

As mentioned above, scoping is a step in the development of service-oriented applications and needs to be integrated in a complete software development life-cycle. The life-cycle model we propose is based on the V-Model. The service-oriented customization of the V-Model has been presented as well.

There are, however, a number of issues that need to be addressed. The proposed scoping technique needs to be integrated in a method for developing process families of service-oriented applications. Especially, the interplay between service components and process families must be investigated in this context.

Additionally, case studies should be conducted to empirically validate the benefits and drawbacks of the proposed scoping strategy. In this context also appropriate measures for analyzing the coupling and cohesion between service components should be developed.

References

[Atkinson01] C. Atkinson, J. Bayer, C. Bunse, E.Kamsties, O. Laitenberger, R. Laqua, D. Muthig, B. Paech, J. Wüst, and J. Zettel. Component-based Product Line Engineering with UML. Component Software Series. Addison-Wesley, 2001.

[Bennett00] K. Bennett, P. Layzell, D. Budgen, P. Brereton, L. Macaulay, and M. Munro. Service-Based Software: The Future of Flexible Software. In Proceedings of the Asia-Pacific Software Engineering Conference, Singapore, December 2000.

[Broehl95] A.-P. Broehl and W. Droeschel (eds.). Das V-Modell. Der Standard fuer die Softwareentwicklung mit Praxisleitfaden (2nd ed.), Oldenbourg Verlag, 1995.

[Lovelock96] C. Lovelock, S. Vandermerwe, and B. Lewis. Services Marketing. Prentice Hall, 1996.

[MDA] Model-Driven Architecture (MDA) Homepage, Object Management Group (OMG). http://www.omg.org/mda/

[WSDL] Web Services Description Language (WSDL) Homepage. http://www.w3.org/TR/wsdl

[WSMO] The Web Service Modeling Ontology (WSMO) Homepage. http://www.wsmo.org

Document Information

Title:	Asset Scoping: Identification of Reusable Software Components Defining Service Components
Date:	October 30, 2004
Report:	IESE-125.06/E
Status:	Final
Distribution:	Public