Fraunhofer Institut
Experimentelles
Software Engineering

Towards Service-Oriented Application Development

Authors:
Joachim Bayer
Michael Eisenbarth
Dirk Muthig

Submitted for Publication to
GSEM'04

Partially supported by
Stiftung Rheinland Pfalz für Innovation (15202-38 62 62 / 666)
"Software Engineering und Workflow Management: Kompetenzzentrum für das Büro der Zukunft"
and PESOA BMBF (01 ISC 34 E)

IESE-Report No. 062.04/E
Version 1.0
May 11, 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)
Sauerwiesen 6
67661 Kaiserslautern

Abstract

Grid computing envisions a platform for negotiating between providers and consumers of computing resources. A grid relates providers of service-oriented applications on demand with consumers. Adaptive services grids extend grids towards not only providing pre-determined applications, but also providing applications that are newly assembled based on requested services. This paper proposes a realization of this vision of grids that is based on a combination of well-established software engineering techniques.

Table of Contents

1 Introduction

Grid computing envisions a platform for negotiating between providers and consumers of computing resources. The grid is a powerful paradigm for providing distributed resources on demand. The term grid was introduced to denote distributed computing infrastructures supporting advanced science and engineering [1]. Grids negotiate between providers and consumers of resources. Therefore, they can be used to share available resources among different collaborating parties. The dynamically changing set of resource providers and consumers is often referred to as virtual organization [2]. A virtual organization is a set of resource providers on one hand and a set of resource consumers on the other hand that use their shared resources for some purpose.

To negotiate among the provided and consumed resources in a grid, the notion of service is used. A service is a specific capability shared in a grid. A grid thus provides a forum to share applications that are determined by the services they provide between providers and consumers.

The current state are grids that provide to consumers means to access exactly those applications the application providers intended. The services of such grids are thus static and pre-determined by the service providers and by the grid's negotiation capabilities. Recent developments move towards adaptive services grids that enable the grid to not only provide pre-determined services but also services that have not been intended by the grid's service providers. In these cases, the grid then uses the requested services to compose, from resources available in the grid, new applications that provide exactly the requested services. The key asset of such an adaptive services grid is an architecture that enables the identification, classification, documentation, coordination, integration, evaluation, adaptation, and evolution of applications and application components based on the services they provide. We refer to such an architecture as service-oriented architecture. The applications derived from such an architecture are consequentially called service-oriented applications.

In this paper, we propose an approach for realizing the vision of adaptive service grids by combining well-established software engineering techniques, namely the V-Model, component-based software development, software product line engineering, and model-driven architectures. In this envisioned approach, an adapted version of the V-Model determines the principal overall process for service-oriented application development. Components and component-based approaches are used to package services. The services that are packaged as components can then be implemented using model-driven approaches. Product line engineering is finally used to support the reuse of ser-

vices in different applications by providing a predefined reuse context for the services.

Throughout the paper, we use a running example for illustration. The example services are taken from the domain of office workflow management.

The remainder of this paper is structured as follows. In Section 2, we present our vision of adaptive service grids and relate this vision to already existing software engineering practices. In section 3, we introduce how we combine existing, well-established software engineering techniques to realize this vision. In section 4 the elicitation, specification, and realization of services is discussed. Section 5 present service-oriented architectures. Section 6 finally summarizes and concludes the paper.

2 Adaptive Service Grids

Service grids are an envisioned forum for negotiating resources between resource providers and consumers. Adaptive service grids extend the resource sharing provided by service grids with the capability to develop new service-oriented applications satisfying requested services based on the resources available in the grid. The grid's service-oriented software architecture is the key asset that enables the establishment of adaptive service grids. In this section, we discuss the fundament that underlies service-oriented architectures and service-oriented applications. We also sketch the envisioned approach for using the service-oriented architecture to consume service-oriented applications.

2.1 Concepts

Grids have been introduced to denote distributed computing infrastructures supporting advanced science and engineering. Computational grids have been defined as hardware and software infrastructures that provide dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities [1]. The basic purpose of a grid was thus to provide on-demand access to computational resources. Later, the focus was moved towards the negotiation of shared resources among providers and consumers of these resources. The term virtual organization was introduced to capture the set of resource providers and resource consumers that share a pool of resources for some purpose [2].

Grids are characterized by the following features [3]:

- Grids coordinate resources that are not subject to centralized control
- Grids use standard, open, general-purpose and interfaces
- Grids deliver nontrivial qualities of services

That is, grids coordinate resources provided by the resource providers in a virtual organization. The coordinated resources are described in a way that enables requesting, discovering, and accessing them. To the resource consumers of a virtual organization, the grid provides resources and combinations of resources in a coordinated way.

Grids negotiate among resource providers and consumers by managing the resources that are provided and delivering them to the consumers. They do, however, only provide applications and application components that have been envisioned by the providers of the respective applications or components. Even

though combinations of applications and components can be provided, there is no way to create novel applications in case a request cannot be satisfied directly by the grid.

Adaptive service grids extend the grid idea to also provide applications that have not been envisioned by the resource providers. The key idea behind adaptive service grids is to use the managed resources to create novel applications in case a request cannot be satisfied directly by the available resources. A generic service-oriented architecture is the central asset of an adaptive service grid. The service-orientation of such an architecture stems from the fact that services are used to describe the capabilities and characteristics of the applications derived from the architecture. A service is some capability an application provides. Services are determined by the protocol by which they can be accessed and by their behavior [2]. In a service-oriented architecture, services are used to document, identify, classify, coordinate, evaluate, adapt, evolve, and integrate the applications and application components provided by the adaptive service grid. Service-oriented architectures are generic architectures that encompass a number of applications and that can be used to derive specific applications from it. They are, therefore, related to architectures of software product families. The relationships between these two types of architectures are discussed in the following section. The similarities can be exploited by transferring ideas and results from the domain of software product line engineering to the domain of adaptive service grids.

2.2 Service Grids and Software Product Lines

Product line engineering is an approach to improve the efficiency of development and maintenance of software systems [4][5]. This is done by exploiting commonalities among similar systems that have some variable characteristics. Such a set of systems is called a product line. The underlying idea of product line engineering is to build a reuse (or product line) infrastructure covering a product line. Such a reuse infrastructure supports the location, the evaluation, and the adaptation of reusable assets, as well as a more accurate planning of development projects. The single products are assembled from generic reusable assets, making software development more a matter of instantiation and composition of pre-built assets than a matter of building everything from scratch. The maintenance of the products derived from a product line can be centralized by maintaining the reusable assets in the product line infrastructure and then re-deriving the products. The product line members can therefore be developed and maintained more efficiently.

The central asset of a product line infrastructure is the product line architecture, a generic architecture that encompasses the architectures of the different product line members. The genericity of a product line architecture is documented by identifying points in the architecture that vary for different members

of the product line (e.g., a component that is present for only a subset of the members).

The genericity is a common characteristic of product line architectures and service-oriented architectures. The main differences are the instantiation process and the binding times. In product line engineering, an application engineering process is used to derive a specific product line member. Based on a decision model that contains all points of variation in the infrastructure and that relates them to features of the resulting applications, a specific product line member is derived from the infrastructure. The result is an application that can be deployed.

For adaptive service grids the situation is different. The idea here is provide service-oriented applications on demand. This especially means that, in contrast to product line engineering, there are no people-based negotiations possible to understand the requirements for a specific application and how they can be satisfied using the available assets. Rather, the requested services are the sole means for the adaptive grid to provide an application fulfilling these requests. The other difference is the binding time. Product lines are usually instantiated, that is specific products are built, before run-time. For service-oriented applications the binding time is at run-time requiring concepts to handle such dynamic variabilities.

There are, however, techniques developed for product line software engineering that can support service grids, for example the creation of novel services base on service requests can be supported by decision models. To satisfy a given request, decision models can be used that are resolved as in application engineering performed in product line engineering. In case the request can be satisfied with the present service-oriented architecture, the service can be delivered. In case the request can not be satisfied directly, the instantiation provides an incomplete architecture and the decision model identifies the services that could not be satisfied with the present architecture. This information is valuable input for the subsequent negotiation process.

2.3 Adaptive Service Grids for Application Development

Once an adaptive service grid is established, it provides consumers with service-oriented applications satisfying their requests for services. Figure 1 shows the situation and three possible scenarios for providing service-oriented applications. On the left hand side of the figures there is a virtual organization that requests certain services from an adaptive grid shown on the left hand side of the figures. If the service can be provided directly by the grid, the request can be fulfilled by delivering the requested service, that is, by delivering an application that provides that service (shown in part (a) in Figure 1).

If, however, the service cannot be directly provided by the grid, there are two possibilities to still provide the requesting consumer with an application that fulfills the respective request. Both possibilities include a negotiation between the consumer of the service and the grid acting a representative for the different service providers. The negotiation is based on functional and on non-functional, quality of service, aspects of the services. The result of the negotiations can be the adjustment of the request to a request that can be directly fulfilled by the grid (part (b) in Figure 1). The other possibility is the creation of a novel application that fulfills the original request (part (c) in Figure 1).

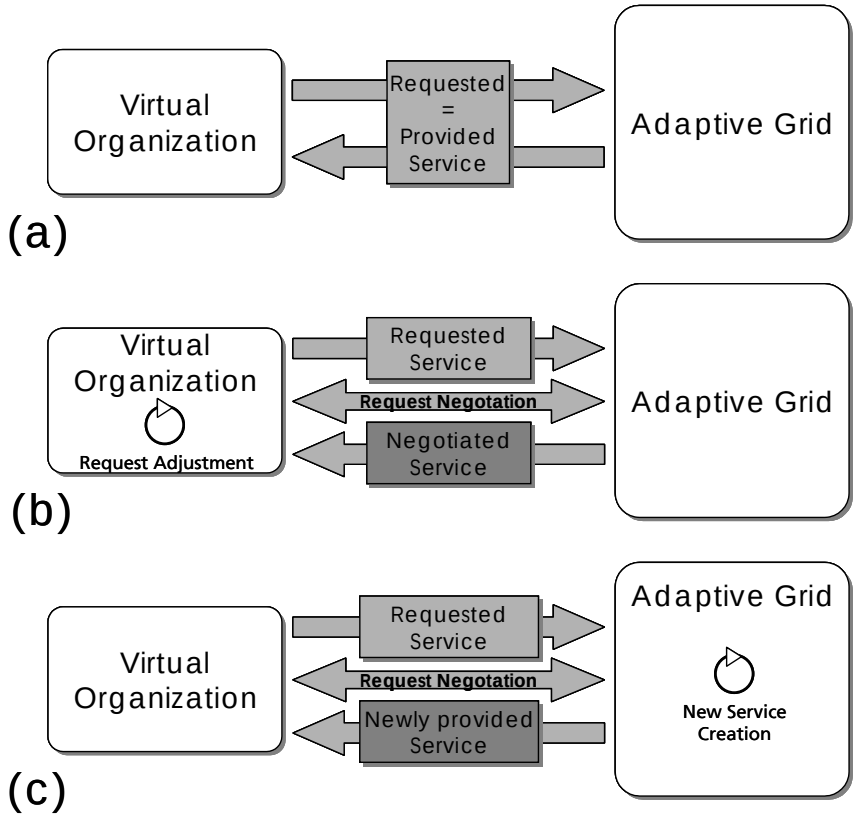

Figure 1. Service Provision in Adaptive Service Grids

To support such negotiations, services must be defined and documented in a way that they are comparable and that enables automated reasoning about the extent to which a service satisfies a given request. The service description should encompass functional, as well as non-functional characteristics and constraints, like quality of service or costs. The service documentation should also describe the parameters within which both the service provider and consumer will negotiate [6], as it is supported by decision models for example.

In order to enable the creation of novel applications from existing assets, services should be packaged in components [7]. This enables the usage of composition techniques from component-based approaches to realize adaptive service grids.

The realization of these requirements on services is discussed in the following section. In the following discussion, we will concentrate on the combination of component-based software development and service-orientation by showing how the notion of service can be introduced in component-based software development. The starting point for this integration is the V-Model. The other two ingredients of our approach, MDA and software product line engineering are addressed only marginally in this paper.

3 Product Model for Service-Oriented Application Development

Numerous organizations use the V-Model as a process model for developing software [8]. 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 [9]. Certain (functional) requirements are specified and designed as services independent of an architecture and component structure. The services are then grouped together and thus yield components of the system architecture. These components can then again be implemented according to the design of the grouped services. The effects of the service-orientation on the V-Model are shown in Figure 2.

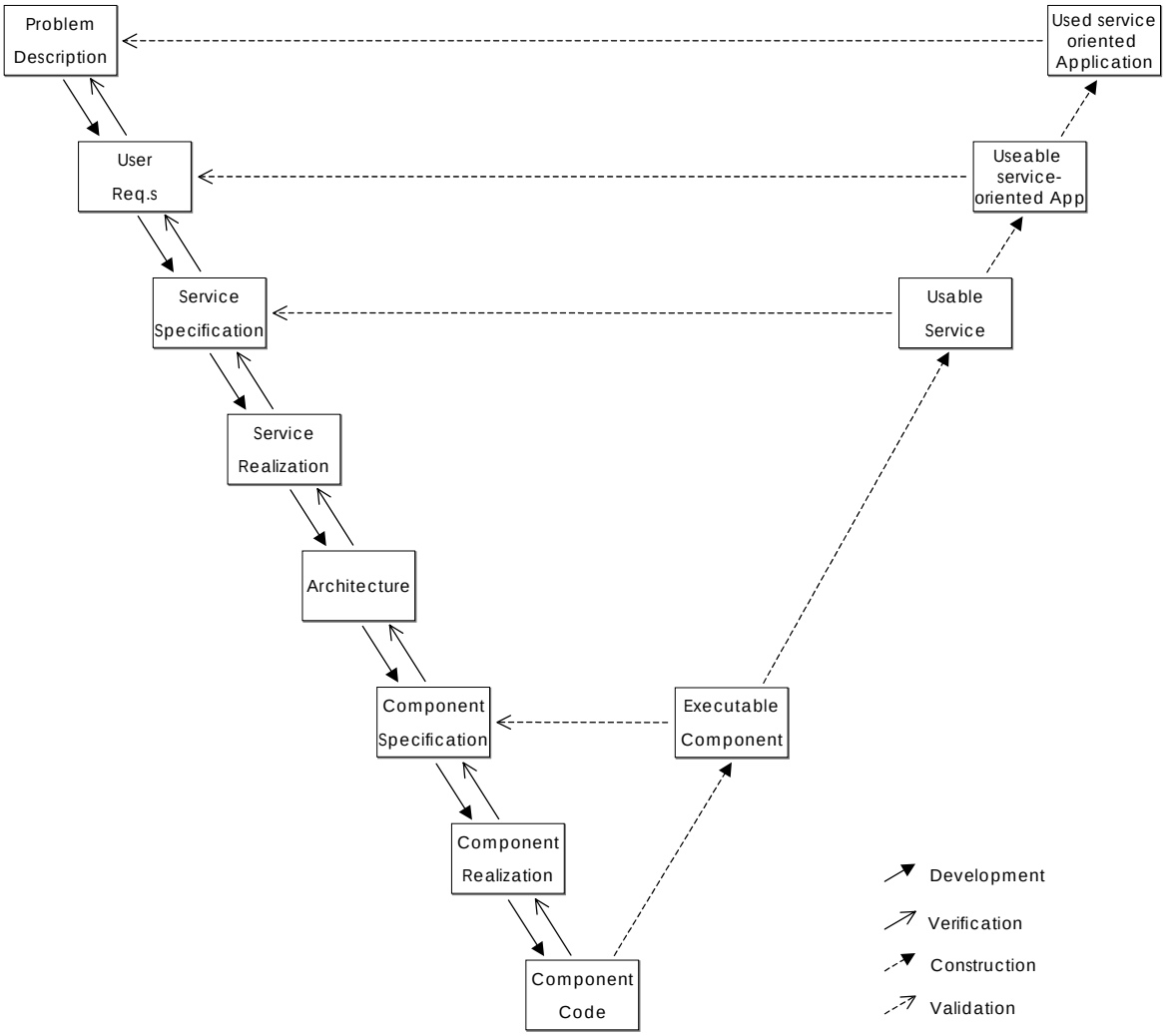

Figure 2. Service-Oriented V-Model

The first two levels of adapted, service-oriented V-Model are, like in the original V-Model, also problem description and user requirements. These are validated against the used and the usable service-oriented application, respectively. The next two levels of abstraction incorporate services into the model. Services are in turn described at two levels of abstraction, service specification and service realization. These terms stem from the KobrA method [10]. The KobrA method is a component-based method for product line development that we use as a basis to capture components and consequentially also for services. The architecture describes how the services are distributed onto subsystems and components. These components are then specified, realized, and captured as component code. Since the KobrA method supports the model-driven architecture approach, the component code can be implemented using MDA techniques [11].

4 Service-Oriented Software Development

In this section, we will address the impact of service orientation on software development. We do this by traversing the service-oriented V-Model given above to show how services are elicited, specified, and designed. But first, we will clarify our understanding of what services are.

Users of an adaptive service grid request certain services as described above. Such a service request will then be compared to the services available in the service grid. The grid holds the necessary information about all services and tries to find a service or a group of services that fit to the user request. As mentioned above, decision models as introduced in software product line engineering provide possible parameters ranges for services and captures relationships between different services in order to retrieve suitable services from the grid base in our approach.

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 [12].

To us, a service is a method, collaboration, or functionality that can be invoked using well-defined interfaces and signatures. Especially in the context of an adaptive service grid with dynamically changing sets of resource providers and consumers, a service can be situated anywhere in the world. In order to be visible and accessible from everywhere in the grid, service providers generally publish a service as a service interface providing service methods. 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.

Service providers usually provide multiple services that are packaged as components. Consumers of services often use more than one service to perform a specific task. The different services need, however, not be provided by the same service provider, but may come from different sources. In these cases, the integration and coordination of services takes place at the service consumer's site.

In the following sections, we will present how services are elicited, specified, and designed.

4.1 Use Case Driven Service Elicitation

Consumers in a virtual organization request services. Thus, it is important to describe services from a consumer-oriented point of view. Use cases are a useful technique to elicit and to specify functional requirements. As services are visible and publicly exposed functionality, we decided to use use cases to capture user requirements for service-oriented applications. The use case descriptions and diagrams are capable to describe, at a high level of abstraction, which tasks the consumers of a service grid perform that need to be supported by services.

Figure 3 shows an example use case diagram from the office domain that shows a user that requires the possibility to send emails and to make phone calls.

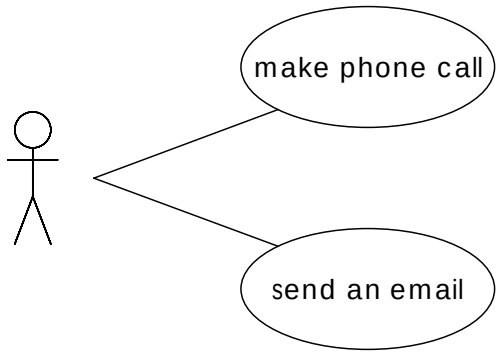

Figure 3. Partial Use Case Diagram for the Office Workflow Support

From the use cases, we elicit the required services using scenarios. Scenarios describe the flow of interaction between a user and an application for a given use case. Each invocation of the application by a user thereby represents a service provided the application to the user.

Figure 4 shows the scenario for the use case “send an email” documented as a sequence diagram. The scenario shows that there are five services related to the use case. One of them, “send_email”, will be specified and realized in the following.

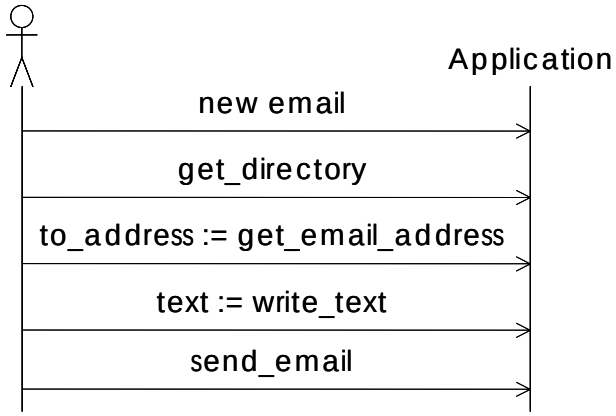

Figure 4. Scenario for Use Case "send an email"

4.2 Service Specification

As described above we use component-based techniques to develop service-oriented applications. Therefore, component specification techniques can be 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 [10]. 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 of a KobrA component is composed of several inter-related models, describing complementary aspects of the component. The separately modeled aspects are the component's functionality, its structure and behavior.

Consequently, a service is also specified by means of three related models. The structural model captures the service and required data structures. A stereotype service is used to depict a service in the UML class diagrams containing the structural specification model.

Figure 5 shows the structural model for the "send_email" service as UML class diagram.

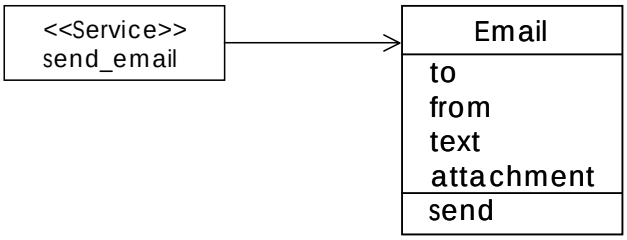

Figure 5. Specification Structural Model for the Service "send_email"

The second model used to specify services is the functional model that describes the externally visible effects of using the respective service. In the KobrA method, operation schemata are used to capture functional models. Operation schemata capture the pre- and post-conditions of service, as well as manipulated data.

Figure 6 shows the operation schema for the "send_email" service.

Service	send_email
Description	Sends an email text with attachment from a sender to a recipient.
Receives	-
Returns	-
Reads	emailAddress to emailAddress from String text File attachment
Changes	-
Assumes	to is a valid email address from is a valid email address text and attachment are available
Result	An email with text and attachment has been sent to the recipient with email address to.

Figure 6. Operation Schema for the Service "send_email"

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

4.3 Service Realization

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 specification structural model by adding information on internal data structures and on services that are acquired to provide the service under consideration.

Figure 7 contains the realization structural model for the "send_email" service. As the figure shows, the service acquires a "sendmail" service from a mail gateway to delegate the actual sending of an email.

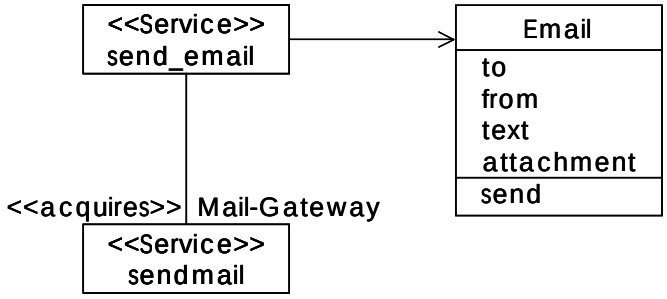

Figure 7. Realization Structural Model for the Service "send_email"

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 8 shows the interaction model for the "send_email" service that captures the delegation of email sending to the mail gateway.

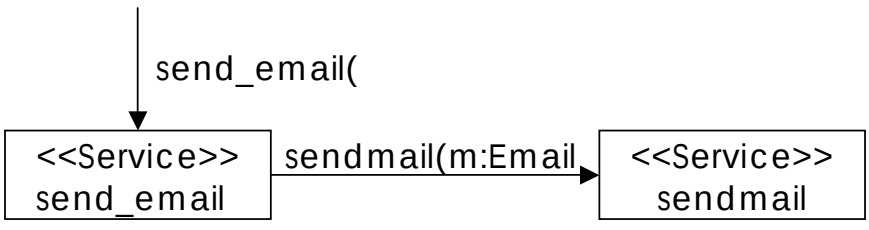

Figure 8. Interaction Model for the Service "send_email"

5 Service-Oriented Architecture

The services that have been elicited, specified, and realized are in the next step mapped to components. These components can then be further developed using existing component-based software engineering methods. Services are packaged together in components at the service provider side to bring together and provide at once a number of related services.

Services are also combined at the service consumer side. Service consumers often use a number of services that need not be taken from a single service provider and combine them to perform some task. The different service must then be integrated and coordinated at the service consumers' site.

The service-oriented architectures we use to package services into components is described in the following. We first introduce the conceptual service-oriented architecture, and then we propose a new architectural view to document service-oriented architectures properly. Finally, we briefly sketch how the resulting components are specified, realized, and implemented.

5.1 Conceptual Architecture

The conceptual architecture for service-oriented applications is depicted in Figure 9. It shows that a client usually interacts with a number of clients (e.g., applications or devices). These clients consist of a user interface and, if necessary, local data caches. Additionally, there is a component that controls the interaction between the user and the different clients, the so-called interaction control component. The interaction control component can use information on the user's context for initialization, which is especially interesting and useful for mobile applications.

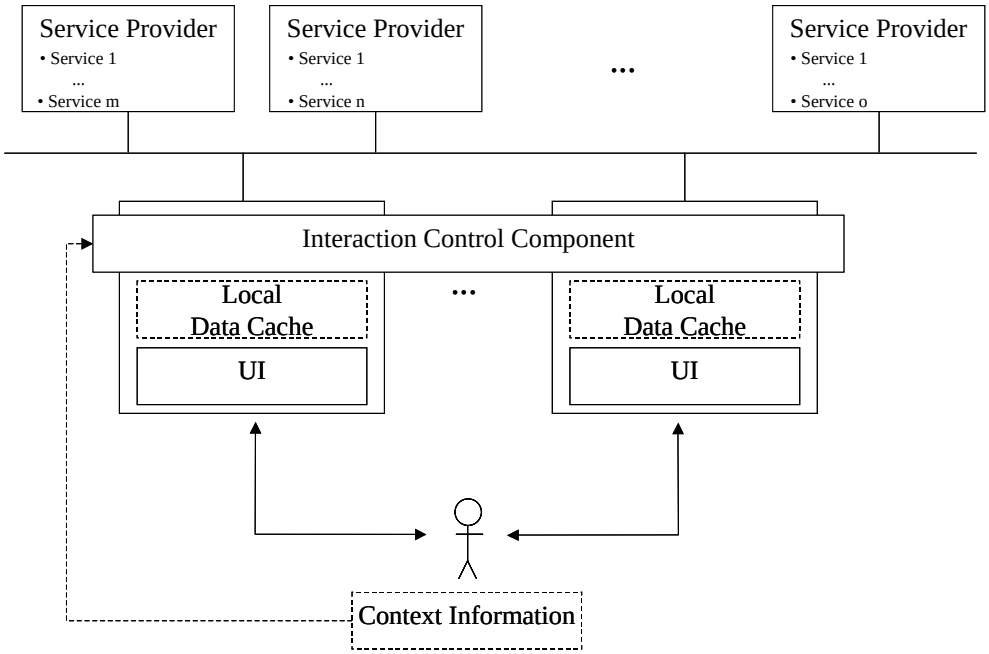

Figure 9. Conceptual Architecture for Service-Oriented Applications

The interaction control component executes use cases involving services from providers and coordinates the communication using the different user interfaces. In the ideal case the user interfaces of the different applications are combined.

5.2 Service Architectural View

To document the architecture of service-oriented application, we introduce a novel architectural viewpoint. An architectural viewpoint establishes the conventions by which specific architectural views are created, depicted, and analyzed [13].

The service architectural view addresses the concerns that are important for service-oriented applications and enables their explicit documentation.

At the service provision side, the view enables the description of services and how they are packaged as components. At the service consumption side, the service architectural view describes the different applications, the interaction control component that coordinates the applications, as well as the necessary data structures.

Finally, the service architectural view makes explicit the communication between service providers on one side and service consumers on the other.

Figure 10 shows an example service architecture view. In this view, a personal information management (PIM) component plays the role of the interaction control component on the service consumers' side. The PIM component provides to its users an email client, as well as a telephone. The services that are consumed by the PIM component are taken from two different service providers that in turn delegate certain services to other service providers that are not visible from the PIM component.

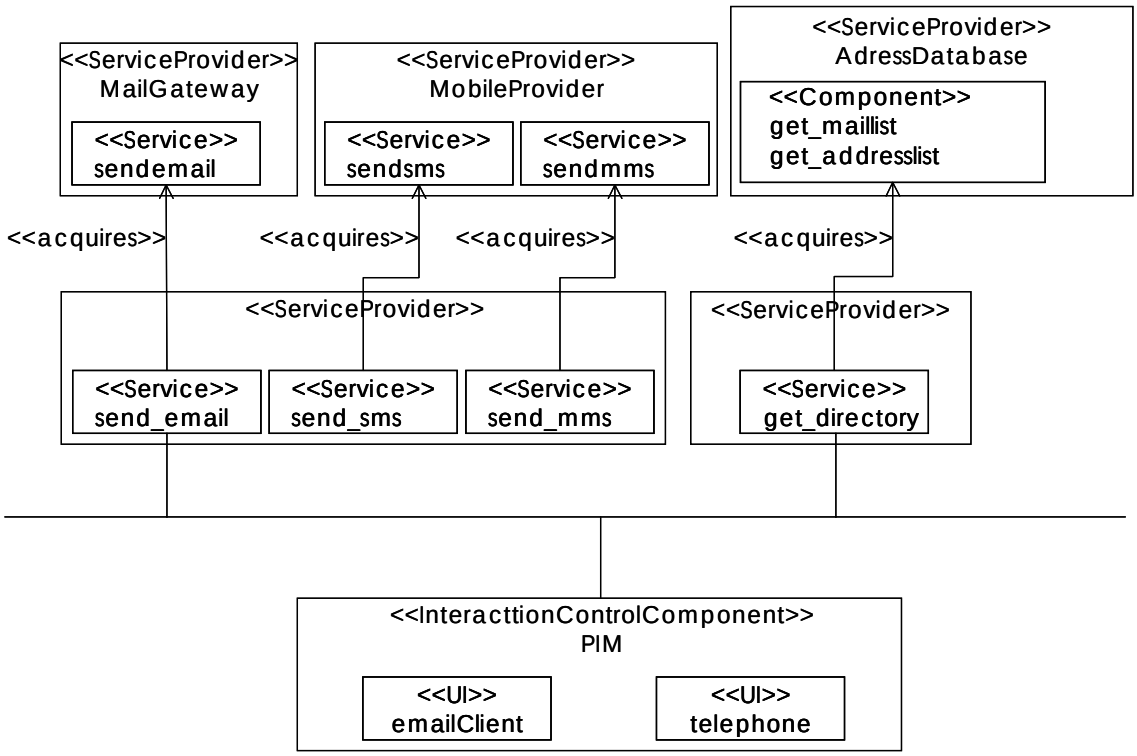

Figure 10. Example Service Architecture View

5.3 Service Components

Service components (like the address database component in Figure 10) combine a number of services in a component and provide the services through their interface. Service components are specified, realized, and implemented using existing component-based techniques. Again, we use the KobrA method to do that. This means that service components are specified and realized using a number of models and implemented using refinement and translation patterns, following the MDA approach.

6 Conclusions

In this paper, we propose an initial approach to realize the vision of adaptive service grids by means of well-established software engineering techniques. The V-Model is used as general process model, in which service-orientation and component-based techniques are integrated. The use of component techniques enables pattern-based implementation of the resulting service components.

Product line engineering is closely related to service-oriented application development. Even though there are similarities, there are also clear differences between the two approaches that must be addressed. The different configurations in which services and service components are used imply the idea of product line of service-oriented applications. The conceptual architecture thereby represents a reference architecture of the potential applications. There are, however, some issues that need to be addressed to use product line engineering techniques to support service-oriented application development. The often used proactive analysis of an application domain that tries to find all possible applications in that domain is not suitable for service-oriented application development due to the dynamic nature of service-oriented applications. Other issues are the automatic instantiation process and the late binding time that requires dynamic variability. We will address these issues in the future based on the approach for service-oriented application development presented in this paper.

References

[1] I. Foster and C. Kesselman (eds.). The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 1999.

[2] I. Foster, C. Kesselman, S.Tuecke. The Anatomy of the Grid. In International Journal of High Performance Computing Applications, 15 (3), pp. 200-222, 2001.

[3] I. Foster. What is the Grid? A Three Point Checklist. In GRIDToday, July 2002.

[4] P. Donohoe (ed.). Software Product Lines - Experience and Research Directions. Proceedings of the First International Software Product Lines Conference (SPLC1), Denver, Colorado, USA, August 2000.

[5] G. J. Chastek (ed). Software Product Lines. Proceedings of the Second International Conference (SPLC2), San Diego, California, USA, August 2002.

[6] M. Turner, D. Budgen, and P. Brereton. Turning Software into a Service. In IEEE Computer, October 2003.

[7] 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.

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

[9] J. Bayer and D. Muthig. Ein Produktmodell zur Service-Orientierten Anwendungsentwicklung. 2004. Submitted for Publication.

[10] 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.

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

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

[13] IEEE Architecture Working Group. IEEE Recommended Practice for Architectural Description, 1999. http://www.pithecanthropus.com/~awg/

Document Information

Title:	Towards Service-Oriented Application Development
Date:	May 11, 2004
Report:	IESE-062.04/E
Status:	Final
Distribution:	Public