Modeling of Services and Service Collaboration in UML 2.0

Petr Weiss and Jaroslav Zendulka

Dept. of Information Systems, Faculty of Information Technology, Brno University of Technology, Božetěchova 1, 612 66 Brno, Czech Republic
{weiss, zendulka}@fit.vutbr.cz

Abstract. One of many definitions of Service-Oriented Architecture (SOA) says that SOA is an architectural style for building next-generation distributed information systems. If we want to get a reliable and good working system, it must be well designed first. This paper deals with Service-Oriented Architecture Design (SOAD), especially with modeling of services. Furthermore, abstraction layers of SOA are introduced and possible using of object oriented approach on each layer is discussed in this paper. Besides, three types of service collaboration are presented. The main objective of the paper is to demonstrate how these types of collaboration can be described in UML 2.0.

Keywords: Service-Oriented Architecture, Service-Oriented Architecture Design, Service Co-operation, service, component.

1 Introduction

Although service-oriented architecture is not a new concept in the area of distributed software architectures, it comes to the fore in recent years thanks to modern technical solutions, e.g. well-defined communication networks and modeling disciplines. SOA implementation rarely starts on the green field. That means creating a SOA solution is almost based on integrating existing systems by decomposing them into services, business processes, and business rules. As a consequence, the SOAD is composition of well-established practices such as Object-Oriented Analyse and Design (OOAD), Enterprise Architecture Design (EAD), Business Process Modeling (BPM) and some other innovative elements. The ideas presented in this paper are a part of a greater project, which deals with a methodology for SOA systems modeling. More precisely, the methodology is aimed at building models of inner-enterprise services and models of collaborations of such services . Inputs for the methodology are elements generated by decomposition of legacy (nonSOA) systems. For the decomposition an appropriate method (e.g. SOMA [1]) is used. Another goal of the methodology is to find how to make the inner-enterprise services accessible for consumers who stay outside of a given enterprise.

This paper describes which object-oriented (OO) features and techniques can be used in SOAD and how UML can be used for modeling of components, assembling components into services, services and a service collaboration.

2 Background

There is no 'standard' definition of SOA. In [5] SOA is presented as an approach for building distributed systems that deliver application functionality as services to either end-user applications or other services.

An abstract view depicts SOA as a partially layered architecture. In this view, SOA consists of three layers: component layer, service layer and business process layer. There are two sections in parallel with these layers. They provide tools for integrating components to services and services to business processes and tools for monitoring and maintaining security and QoS of SOA applications.

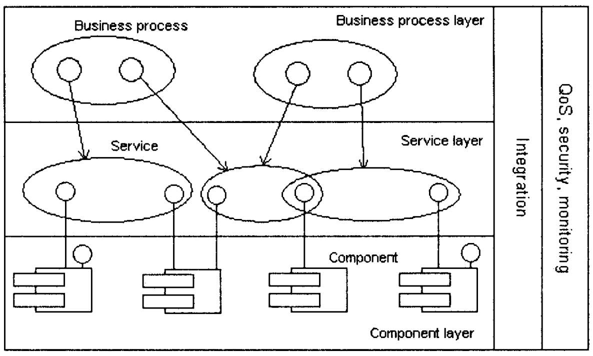

Fig. 1. SOA as a partially layered architecture.

It is evident that the key element of SOA is a *service*. Each service is a composition of collaborating *components*. Components are responsible for providing service's functionality and maintaining the quality of service.

Services are loosely coupled software entities with well-defined, published *interfaces*. The service interface separates provided functionality from its implementation (services are implementation-independent) and forms so-called service description. The service description is available for searching, dynamic binding and invocation of a given service by a service consumer. The communication between services is based on *message sending*.

On a higher abstraction level, services can be assembled into a *business processes*. In SOA terms, a business process consists of a series of operations which are executed in an ordered sequence according to a set of business rules. The process of sequencing, selection, and execution of business rules is referred to as *service choreography*. Typically, choreographed services are invoked in order to respond to business events.

More information about SOA principles can be found in [5] and [4].

Object-oriented analysis and design (OOAD) is an approach in which a system is modeled as a collection of classes. The system behavior is achieved by collaboration of instances (objects) of these classes, and the state of the system is the combined state of all the objects in it. Collaboration among objects is achieved by sending messages. Since SOA is a new paradigm and services and objects are two different concepts, the question is "Could OOAD or a part of OOAD be used in SOAD?" The following paragraph discusses which OO features and techniques [8] can be used in SOAD. As mentioned in Introduction, this paper deals only with components and services. Consequently, using OO at the business process layer is not described.

- *Information hiding*: SOA services are for service consumers black boxes with well-defined interfaces. The goal is to separate what service does (its interface) from how it does it (its implementation). From the services point of view, components are also black boxes. Services have only information about components interfaces and route requests (for a service) from a consumer to subordinate components.
- *Messaging* is the fundamental communication model for components and services in SOA.
- *Inheritance* can only be considered on a specification level of component and service modeling.
- *Polymorphism* describes the situation where the result (of a behavior) depends on the class of an object the behavior of which is invoked. In other words, two or more classes accept the same message, but respond on it differently. The possible use of polymorphism in SOAD is similar to inheritance.
- *Classes and instances*: Classes are templates for creating instances (objects). From SOA point of view, this concept can be applied to both components and services.
- *Encapsulation*: A service encapsulates the state and behavior of a number of components.

It results from the previous paragraph that except inheritance and polymorphism the underlying OO features can be used in SOA for modeling of components or services. It should be pointed out that all OO features including inheritance and polymorphism can be used for modeling the inner structure of components.

3 Three Types of Collaboration among Services

To model collaborations of services inside an enterprise, it is useful to distinguish several categories of collaborations. Each category is defined by a set of communication rules (a protocol) and by the purpose of the collaboration. The categories are following:

- The $Service\ Co-operation$ describes a collaboration of services, in which one service has to use another service(s) to fulfil incoming requests.
- The *Service Aggregation* is a set of rules which defines how to create a new service from two (or more) existing services. The new service provides combined functionality of its building services.

- The *Service Choreography* is such collaboration of services that supports a business process.

The basic concept of service co-operation modeling is explained latter in this paper. More about service choreography can be found in [5], [4] and [2], the service aggregation is object of future research.

4 Using UML in SOAD

As mentioned before, object-oriented technology and languages are great ways to design and implement components. Unified Modeling Language (UML) is a specification language for object modeling. Since there exists a relationship between OO and SOAD (described in section 2), UML can be used in SOAD. UML provides extension mechanisms (stereotypes, tagged values, constraints) which enable to model services. This chapter describes the use of UML 2.0 [10] for modeling components, services and service collaboration. Furthermore, stereotypes for modeling SOA components and services are presented.

4.1 Models of Components and Services

In this paper, we use a simple example of co-operating services to demonstrate the use of the proposed modeling techniques. We assume two collaborating services: `Smath` and `Splus`. `Smath` offers calculation of some mathematical operations, but in fact it does not perform the operation. It collaborates with other services. We consider only one operation here the sum. The service that really performs it is referred to as `Splus`. A part of this example is shown in Figure 2. The figure depicts a model of `Smath`. The structure of this model is explained in the following paragraph.

A service is modeled as a stereotype `service` , which is derived from the element `component` of the UML. There are two possible abstract views of a service - an external and internal one. The internal view (or "white-box" view) shows how the external behavior is realized internally. In this case, the inner structure depicts how component instances are interconnected to provide the required functionality. An example of this view is shown in Figure 2 . This level is used only for components interconnection modeling inside a service. The other abstract view, the external view (or "black-box" view), is used to model services in service collaboration. It hides the inner structure (implementation) of services.

The mapping between the internal and external view is by means of a `delegation connector` . It is represented by a `port` and an UML predefined-stereotype of dependency `delegate` . The port is shown as a small square symbol on the boundary of the symbol denoting a service.

A port delegates to a set of subordinate components (and vice versa). At execution time, signals will be delivered to the appropriate instance. In the cases where multiple target instances support the handling of the same signal, the signal will be delivered to all these subordinate instances.

Component modeling is easy, because UML provides a modeling element `component` (a subclass of `Class` in the UML metamodel). A `component` is a self contained unit that is able to interact with its environment through its provided and required `interfaces` (`Classifiers` in the UML metamodel). A component can be replaced at run-time by a component that offers equivalent functionality based on interface compatibility. Components are modeled as "black-boxes".

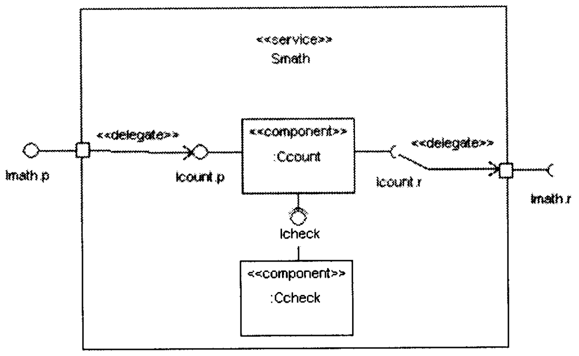

Fig. 2. A Service diagram - internal view of `Smath`. Notation: Imath.p is the provided interface of the service `Smath`, Imath.r is the required interface of `Smath`.

4.2 Co-operating Services

Co-operating services have following properties:

- service can receive infinite number of incoming messages
- incomming requests are queued up into the input queue
- request processing and communication are processed in parallel
- the service description of each co-operating service is known

Following two sections introduce fundamentals of static and dynamic modeling of service co-operation. Since we focus on modeling services inside an enterprise, we assume services with fine-grained interfaces.

4.3 Static Models of Co-operating Services

In general, each service can provide its functionality to other services and it can also require some functionality from other services. The former role of the service is referred to as a "service provider" and the latter one as a "service consumer". Because each service can play both roles, it should realize a provided interface of the service provider and has a required interface of the service provider. In addition, since the communication of the services is asynchronous, our model is based on the Observer pattern [6].

Figure 3 depicts a static model of co-operating services `Smath` and `Splus`. Here, the service `Smath` plays the role of a service consumer and the service `Splus` plays the role of a service provider. Both services realize interfaces: `IProvider` and `IConsumer`. `IProvider` contains a method `attach(service)`. Using this method, a service consumer registers at the service provider. The registration also includes a service request, which is represented as a `service` parameter of the `attach()` operation in the model. Such requests are queued. It is modeled by means of aggregation with the service provider being an aggregate. As soon as the service provider completes the request processing, a method `notifyConsumer()` sends a message with the result to the service consumer. The corresponding operation in the model is `setResult(serviceRslt)`.

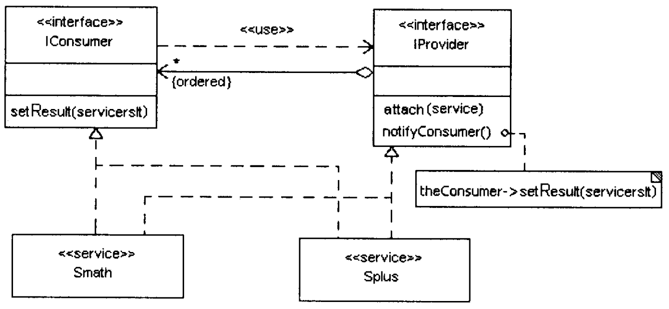

Fig. 3. A model of co-operating services (Service diagram - external view).

4.4 Dynamic Models of Co-operating Services

If we want to model service-to-service connection during service co-operation, we can not use the approach we used to model components composition inside a service (section 4.1). Components are assembled into services in advance in design time. On the other hand, services can be discovered and are bound dynamically. Consequently, we can model service co-operation only by means of UML behavioral models, namely state machine and interaction diagrams. We have chosen a sequence diagram to model communication of services. Since the modeled entities are services, the diagram have to fulfill the following restrictions:

- Lifelines are services.
- All messages among services are asynchronous.

The second restriction ensures that a sender does not wait for a replay (after a request was sent) and the sender can accept another incoming messages or process stored requests (replies). Figure 4 shows communication of services `Smath` and `Splus` from Figure 3.

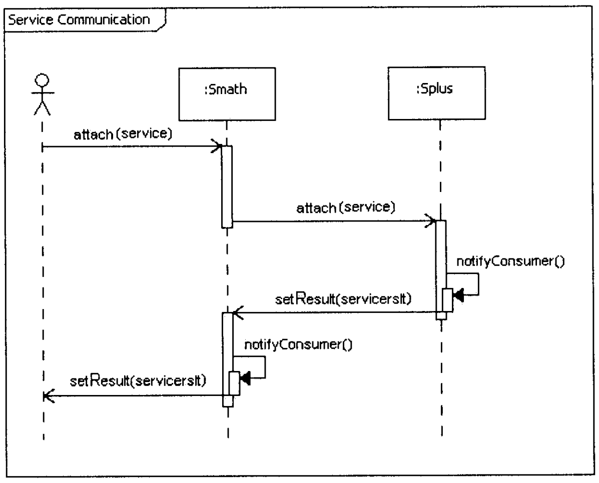

Fig. 4. A Sequence diagram. This diagram shows details of communication between `Smath` and `Splus`.

5 Conclusion and Future Work

This paper briefly discusses the use of object-oriented approach in SOAD. It is shown that information hiding, messaging, classes and instances, and encapsulation are concepts that can be used in SOAD. In addition, they are supported by UML. Therefore, UML was chosen as a modeling language for components, services and service co-operation. UML provides well-defined diagrams for modeling composition of components inside a service and for modeling collaborations of services including their communication. The future work will focus on improvement and extension of the approach mentioned above, especially on creating a meta-model of service co-operation and service aggregation that will enable to develop more advanced and sophisticated models. In addition, it would be useful to create some verification methods for created models .

This research has been supported by the Grant Agency of Czech Republic grants No. 102/05/0723 "A Framework for Formal Specifications and Prototyping of

Information System's Network Applications" and by the Research Plan No. MSM 0021630528 "Security-Oriented Research in Information Technology".

References

1. Arsanjani, A.: Service-oriented modeling and architecture. Document is available on URL http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design1/
2. Benatallah, B., Dijkman, R., Dumas, M. and Maamar, Z.: Service Composition: Concepts, Techniques, Tools, and Trends. In Stojanovic, Z. and Dahanayake, A. (Eds): Service-Oriented Software System Engineering. Idea Group (2005), ISBN 1-59140-426-6.
3. Booch, G., Rumbaugh, J., Jacobson, I.: Unified Modeling Language User Guide. Addison-Wesley (1999), ISBN 0-201-57168-1
4. Endrei, M., et al.: Patterns: Service-Oriented Architecture and Web Services. IBM Redbooks (2004), ISBN 0-738-45317-X. Also available at URL: `http://www.redbooks.ibm.com/redbooks/pdfs/sg246303.pdf` (February 2006)
5. Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR (2005), ISBN 0-13-185858-0
6. Gamma, E. et al.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional Computing Series (1995), ISBN-10: 0201633612
7. Heckel, R., Lohmann M., Thone, S.: Towards a UML Profile for Service-Oriented Architectures. Document is available on URL `ttp://citeseer.ist.psu.edu/heckel03towards.html` (April 2006)
8. Jacobson, I.: Object-oriented software engineering, A Use-Case Driven Approach. ACM Press (1992), ISBN 0-201-54435-0
9. Johnston, S.: UML 2.0 Profile for Software Services. Document is available on URL `http://www-128.ibm.com/developerworks/rational/library/05/419_soa/` (February 2006)
10. Unified Modeling Language - UML 2 Superstructure. Document is available on URL `http://www.omg.org/technology/documents/formal/uml.htm` (February 2006)