Approaches to Service Interface Design

Martin Henkel[1], Jelena Zdrakovic[2]

[1] Royal Institute of Technology and Stockholm University,
Forum 100, 164 40 Kista, Sweden
martinh@dsv.su.se

[2] Royal Institute of Technology and University of Gävle,
Forum 100, 164 40 Kista, Sweden
jzc@dsv.su.se

Abstract. Services can be seen as an extension of the component and object oriented paradigms. Just like objects and components services need well defined, well documented interfaces. However, unlike most components and objects, services must be designed to work in a wide range of computing environments. Service design must cope with problems ranging from the integration of remote-procedure based legacy systems to the use state-of-art message based orchestration servers. In this paper we provide an overview of three fundamental approaches to service interface design; method centric, constrained and message centric interface design. Furthermore, we briefly discuss how each approach impact the use of web service standards and conclude with future research directions.

1 Introduction

Many view web services as a next step of evolution from components and object-oriented development. This view is not surprising since components are often described as providing services via their interfaces [1]. Components separation of interface from implementation [2] is also one aspect that corresponds with web services separation of interface description (WSDL) from its implementation. However, even if components, objects and services do share some basic concepts many authors consider it to be a mistake to design web services in the same way as components and objects [1],[3],[4].

Web services as a technology have been in use a couple of years now. Knowledge on how to (and how not to) design web services is starting to be documented. One specific skill that a web service architect need to posses is the ability to design the interfaces to a web service. This task includes the selection of a service's operations, and their input and output parameters. To most experienced architects this might be a routine task, while novices struggles with the various "best practice" design descriptions in literature and as found on the World wide web.

The design of a web service interface (it's operations and their parameters) can vary widely depending on the fundamental design principle that is applied. The problem of selecting an approach is further complicated by that some of the design principles found in the literature are widely agreed upon as being "best practice", while other principles are popular, but controversial.

This paper gives a high-level overview of the fundamental approaches to service interface design. We also provide a brief discussion on how the use of each described design approach influences the use of XML schema, and the WSDL and SOAP web service standards.

2 Designing Interfaces

From the beginning the basic web service protocol SOAP was designed to be a simple way to do remote procedure calls (RPC) over the Internet. The similarity between SOAP and earlier distributed communication protocols such as CORBA and DCOM made SOAP easier to understand and implement. This similarity also affected the design of web service interfaces, services where designed with "RPC style" or "method centric" interfaces, with clearly separated operations and well-defined parameters [5]. However method centric interface design is not the only design approach suggested as a "good" way to design interfaces. Other "message centric" approaches suggest that the design should focus more on the design of the messages in the system, and that the interfaces should contain a comparable small set of operations [6]. This discussion about method versus message centric design has caused some controversy. Some authors argue that the method centric design should not be applied when designing web services [7]. Others argue that a mixture between method and message centric design is appropriate, this third approach is called a constrained design [8].

In the following sections we briefly describe the three basic approaches to service design.

2.1 Method Centric Interfaces

As mentioned earlier, method centric, or RPC style design, is the common way to design interfaces in distributed systems based on components or distributed object technology. RPC style design results in relatively large set of operations for each service interface, each operation performing a certain function. Figure 1 shows a simple example of a method centric interface to a library system. The simple example interface can be utilized to get information about a book (the getBookDetails operation), and to register that a book is loaned (the regBookLoan operation).

The RPC style design of interfaces has several drawbacks when applied in an environment where several separated applications need to communicate [9]. The RPC style design can cause tightly coupled interfaces, where each client needs to know the exact definition of the service interface. When the interface changes all service clients need to be updated, which might cause a lot of extra work in large systems. For

example, adding a parameter to the method regBookLoan in figure 1 require that all the clients of the interface must be recompiled and re-deployed.

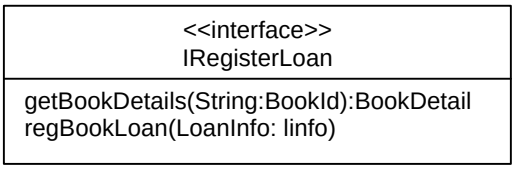

Fig. 1. A method centric interface (parameter structure omitted).

The web services standards WSDL and SOAP are central for describing method centric interfaces, while the use of XML schema to describe the parameter structures is "optional". WSDL files are important when using method centric interfaces, because they describe the operations that the interface supports and the parameters that the operations can handle. Most development tools support the creation of WSDL files from a language specific definition of the service interface (e.g. a Java interface). The need for using XML schemas can be considerably lessened by using standardized XML object serialization ("SOAP Section 5 encoding"). For example, the parameters and return values in the IRegisterLoan interface can automatically be serialized into XML by using the standardized SOAP object encoding. The need for a domain specific XML schema is then alleviated completely.

The SOAP standard also got specific support for transferring calls made to method centric interfaces. The SOAP/RPC message style specifies where in the SOAP message operation names and parameter data should be encoded. This makes it possible for middleware servers and development tools to separate operations and their parameters, messages can thus be handled in a "method centric" way.

2.2 Message Centric Interfaces

Message centric design promotes the use of message structures instead of operations. The call semantic is then embedded in the messages sent to the web service. A message centric interface could be viewed as a method centric interface with only one operation, "send(msg)". This means that all information about a service request is embedded in the message structure. This approach has several advantages. Firstly, the interface is fixed, changes are only made to the message structure. Secondly, messages can be handled by intermediate parties (such as message queues) without them having to know the details of the interface. However, a message centric design makes it difficult to interpret and understand the functionality provided by a service. To understand the functionality of a service, a developer has to examine the structure of the messages that the service can handle. In a method centric interface the developer can get a simple overview of the functionality by examining the available operations.

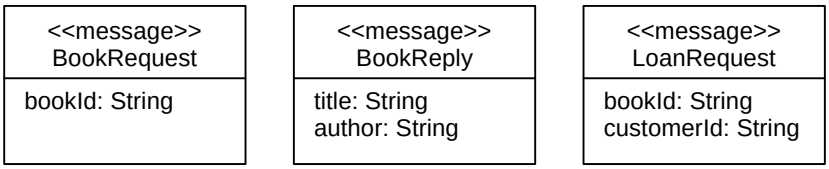

Fig. 2. A message centric interface, consisting of message definitions

The usage of message centric interfaces requires that all message structures are described, for example by using XML schema. Therefore, schema design is a central activity when designing message oriented systems. However, WSDL plays a minor part when dealing with pure message oriented interfaces. In the extreme case there simply are no operations that need to be described by using WSDL. Note that this is precisely the opposite of method centric interfaces, where WSDL plays a major role, and XML Schema a minor.

Just as for method centric interfaces the SOAP standard has support for message centric interfaces by virtue of the SOAP/Document message style. By using the SOAP/Document style developers are free to structure messages in (almost) any way possible, without the need to define operation and parameter names explicitly.

2.3 Constrained Interfaces

Constrained interfaces [8] are interfaces that contain a fixed set of standardized operations. Figure 3 shows a simple constrained interface for a library service. The operations in the interface depicts the basic operations needed for requesting information (the get operation) and creating entities (the create operation). The generic parameters contain information about what information to retrieve or create. This kind of interface can be used for a wide range of requests, for example the get operation in figure 3 might fetch information about books or customers, depending on the contents of the data parameter. A common way to define a constrained interface is to include all the CRUD operations (Create, Read, Update and Delete). However, common operations like performing access control might also be included (for example by adding the operation “checkAccess(data, userId)”).

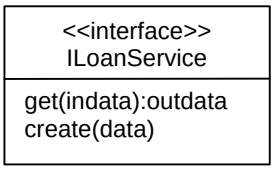

Fig. 3. A constrained interface for the Library service (parameter structure omitted).

A (well known and controversial) example of an architecture entirely relying on constrained interfaces is Representational State Transfer (REST) [10]. REST promotes the use of HTTP and its basic operations for building large-scale distributed systems. In this architecture the HTTP operations PUT, POST, GET and DELETE

form a standardized, constrained interface. These operations are then applied to resources, located with Unified Resource Locators (URLs). By using only the four operations it is possible to build large distributed systems.

Since a constrained interface is standardized across an entire system, this design has the same advantages as the message centric design. This means that the interfaces operations are not affected by changes. Instead, changes affect the operation parameter structures. However, compared to the message centric approach constrained interfaces do provide a coarse grained overview of the service functionality. For example, it's easy to identify if a system (or subsystems) support deleting entities if the constrained interface has a Delete operation. A detailed examination of a constrained interface requires the examination of the parameter structures passed to and from the operations, just like the examination of message centric interfaces requires this.

WSDL can be used to define constrained interfaces. However, most effort of defining constrained web service interfaces must go into the definition of the XML schema that defines the parameter structures. A single system might only need a single WSDL file containing the operations of the constrained interface, and link to the used XML schemas.

The three presented design approaches are in no way exhaustive for representing possible service interface designs. The three approaches should rather be viewed as a scale where on one side the notion of operations are paramount (the method centric approach), and on the other side the notion of operations dissolves and all focus is put on the messages definitions (the message centric approach). On this scale, constrained interfaces can be put in the middle, since they do use the notion of operations, but only in a limited, tightly standardized way.

3 Conclusion and Further Work

In this paper we briefly presented and discussed three approaches for interface design. Further examined, these approaches can be used to categorize, explain and select an appropriate service interface design approach.

We started this work when we examined several large Swedish companies use and design of web services. What we discovered then was that there is no single prevailing approach for designing service interfaces. We found examples of all of the three design approaches, some companies even employed several different approaches for their services. This view was confirmed when studying literature on service design, some sources advocated a "message centric" design, some (often more developer centric) sources gave examples of "method centric" web services, others argued heavily for the REST architecture (thereby pointing towards the use of constrained interfaces).

Providing a rough classification of service interface design approaches is just the first step in providing a framework for choosing an appropriate service interface design. Beside the approach classification we identify the definition of service properties and the examination of technology support as important research areas:

Definition of desirable service properties. These properties define in which environment and under which requirements the service should work. For example a desired property of a service oriented system might be that it should be "loosely coupled", this means that the services should be independent. The desire to create loosely coupled systems might steer an architect towards the use of message centric interfaces. Other desirable properties of services must be identified, and the support for the property must be examined for each design approach. These properties form the basis for explaining why a certain design is preferable in a particular situation.

Further examination of technology support for the design approaches. In this paper we briefly discussed how each approach could impact the use of technologies such as SOAP and WSDL. However, we must also consider how other common technologies such as message oriented middleware and process orchestration engines affect the service design. Examining the technical limitations and possibilities will provide the fundament for explaining how a design approach can be supported, and well as how it is influenced by technology issues.

We plan to pursue these research directions in forthcoming papers.

Acknowledgement

This work is a part of the Serviam project, partly funded by the Swedish Agency for Innovation Systems. For further information about the Serviam project, please refer to www.serviam.se.

References

1. Allen, P., and Frost, S.: Component-Based Development for Enterprise Systems: Applying the Select Perspective, Cambridge University Press (1998)
2. Cheesman, J., and Daniels, J.: UML Components, Addison-Wesley (2001)
3. Monday, P.B,: Web Service Patterns: Java Edition, Apress (2003)
4. Piccinelli, G., Salle, M., Zirpins, C.: Service-Oriented Modelling for e-Business Application Components, HP Labs Technical Report HPL-2001-123 (2001)
5. Vogels, W.: Web services are not distributed objects, IEEE Internet Computing, Vol. 7, Iss. 6. (2003) 59- 66.
6. Prescod, P.: Second Generation Web Services, www.xml.com/pub/a/2002/02/06/rest.html (2002), accessed 13 Dec 2004.
7. Arsanjani, A.: Developing and Integrating Enterprise Components and Services, Communications of the ACM, Vol. 45, No. 10 (2002)
8. Orchard, D.: The four Major Constraints to Loosely Coupled Web Services, Webservices.org 2003, http:///www.webservices.org/index.php/article/articleprint/1246/-1/24/, Accessed 15 Dec 2004.
9. Chappell, D.: Asynchronous Web Services and the Enterprise Service Bus, WebServices.org June 2002, www.webservices.org/index.php/article/articleprint/352/-1/24/, Accessed 15 Dec 2004.
10. Fielding, R. T.: Architectural Styles and the Design of Network-based Software Architectures, Doctoral dissertation, University of California, Irvine (2000)