Serviam Literature Survey

Part III

Web Service Design

2004-02-12

Martin Henkel

SERVIAM-LIT-03

Version 1.3

10 pages

Table of Contents

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 (Allen, 1998). Components separation of interface from implementation (Cheesman, 2001) 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 (Allen, 1998) (Monday, 2003) (Piccinelli, 2001).

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. This chapter gives an overview of the state-of-art regarding web service design, as found in industry as well as academic publications.

In order to give an overview of the state-of art, three views of service design can be examined:

- *Service scope* deals with how to select a suitable amount of functionality to implement as a web service (this is sometimes called service granularity).
- *Designing interfaces* describes current high-level approaches to he selection of individual web services operations and messages (parameters).
- *Web service interaction models* describe common communication patterns for web services, e.g. the exchange of SOAP messages.

The first two views (service scope and interface design) is described in this document, the third view (interaction models) is described in part 4 of the literature survey.

Some of the design principles found in the literature are widely agreed upon as being “best practice” while there exist a lot of controversy regarding others.

2 Selecting Service Scope

Designers of objects-oriented databases have a deceptive simple task; map each concept in the domain to a class in the database, a 1:1 mapping. However, this kind of mapping from business domain into parts of an IT system is not as simple when building systems based on web services, there is no simple 1:1 mapping. There exist several loosely defined guidelines that can be applied selecting the scope of a web service. Commonly, these guidelines are based on either dynamic or static aspects of the business.

2.1 Guidelines Based on Static Aspects

Guidelines based on *static aspects* describe how services should be identified with input from the concepts in the business domain using a domain model/information model. Example of such an approach to design is constructing a web service for each business object/concept in the business domain (Monday, 2003). This will result in one service for “Customer”, one for “Product” etc. However, this approach taken literally is discouraged for large-scale systems (Monday, 2003), the services will commonly be to fine grained using this approach. A design principle lent from component-based methods (Cheesman, 2001) is to select a few central concepts from the domain model and create a service for each of them. These central concepts are sometimes referred to as “focus classes”, or “strong entities” (Connolly, 1998), basic heuristics how to select these exists. Compared to using only the concepts, this approach of selecting central concepts will result in

more coarse-grained services. This will also result in that component and services have the same granularity. The authors that use this approach, thus consider components to be a good basis for building services (Arsanjani, 2003).

2.2 Guidelines Based on Dynamic Aspects

Guidelines based on dynamic aspects of the business, such as business processes are often recommended when selecting services (Monday, 2003) (Dumas, 2001) (Channabasavaih, 2003) (Wald, 2001). An example of this approach is to create a web service for each major business process, such as "purchasing". This will give the services a higher granularity, compared to having services correspond to objects (Vinoski, 2002). The following "dynamic" business aspects have been suggested as a starting point for building services:

- Business processes (Dumas, 2001)
- Business functions (Channabasavaih, 2003)
- Business use cases (Sundblad, 2003)

An interesting side note is that most of the sources recommended a "dynamic" design perspective, although this approach have since long been considered to create systems that are not easy to extend (Parnas, 1972).

2.3 Summary

The following table (table 1) summarizes the various high level "guidelines" described previously.

Basic design principle	A service corresponds to a	Example of services
Dynamic	Business Process	Purchasing
	Business Use case	Search for Orders
	Business Function	Create Order Statistics
Static	Central concept, "focus class"	Order (including order rows, order statistics)
	Business object	Order

Table 1, Overview of high-level design approaches for selecting service scope

Since processes can be broken down into sub processes, the statement that "each service should correspond to a single business process" does not clearly define the granularity of a service. Although not precisely defined the above guidelines can give a hint of what is considered to be the right granularity for web services.

A typical layered architecture uses both the dynamic and structural approach to service design. Commonly a structural design is used for a "data access" layer (sometimes called the entity layer), while a dynamic design approach is used as a packaging layer on top of the data access layer. The appropriate design approach is thus highly dependent on the type of service that is going to be constructed.

3 Designing Interfaces

From the beginning the basic web service protocol SOAP was designed to be 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 (Vogels, 2003). However method centric interface design is not the only design approach suggested as a “good” way to design interfaces. Other “message centric” styles 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 (Prescod, 2002). 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 (Arsanjani, 2002) (Orchard, 2003).

3.1 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. However the RPC style design of interfaces has several drawbacks when applied in an environment where several separated applications need to communicate (Chappell, 2002). 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.

WSDL are central for describing method centric interfaces, while the use of XML schema to describe the parameter structures is “optional”. WSDL files are important because they describe the operation that the interface supports and the parameters that the operations can handle. Most development tools support the creating 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”). However, the use of this encoding scheme is considered deprecated.

3.2 Message Centric Interfaces

Message centric design promotes the use of message structures instead of operations. Taken to the extreme message centric design can result in web service interfaces with only one method “send(msg)”. The call semantic is the embedded in the message sent to the web service. 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.

The usage of message centric interfaces requires that all messages are described 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.

3.3 Constrained Interfaces

Constrained interfaces (Orchard, 2003) are interfaces that adhere to a fixed set of standardized operations. An example of a constrained interface is HTTP. HTTP defines the operations PUT, POST, GET and DELETE. These operations are then applied to resources, located with Unified Resource Locators (URLs). Using only the four operations it is possible to build large distributed systems. Since the interface is standardized, this design has the same advantages as the message centric design. The interface does not have to be updated. Compared to the message centric approach constrained interfaces do provide a course grained overview of the service functionality. For example, it's easy to identify that resources that support the "DELETE" operation in HTTP can be deleted.

3.4 Summary

The decision to use a message centric or method centric approach can be affected by the choice of protocols, architecture and products. Below is an example of a protocol, an architecture and a middleware product and how they support the interface design approaches:

Product. Message oriented middleware (MOM) products, are solely based on the message centric approach. Communication with a service is seen solely as a message exchange where the semantic is entirely encoded in the messages.

Protocol. SOAP supports using both document base and RPC based encoding. Document based encoding are meant to be used for services that are designed with constrained or message based interfaces. SOAP-RPC is meant to be used with services using a method centric approach. When using SOAP-RPC the method and parameter names of the called method is encoded into the exchanged SOAP messages.

Architecture. The architecture Representational State Transfer (REST) promotes the use of HTTP and its basic operations for building large-scale distributed systems (Fielding, 2000). Thus, REST is an example of an architecture that is based on a constrained interface.

The three examples are summarized in Figure 1, below.

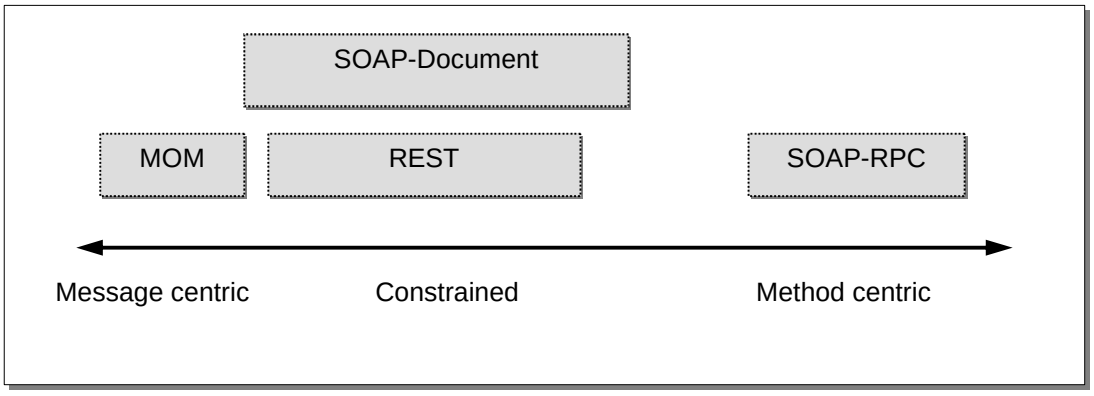

Fig 1. Approaches to interface design

Generally, most sources recommend a message centric design for web services. A message centric design promotes loosely coupled services, which is desirable in situation where interface changes are expensive (commonly in B2B scenarios). However, a method centric design is better suited for well-controlled environments such as building single applications deployed in an intranet.

4 Summarized Sources

In this section short summaries of resources that discuss the design of web services are presented. These sources have been selected for inclusion here because they contain important insights, and/or they document the current state of art regarding the design of web services. These sources, and others, have been used as an input to the overview. Other sources used for the overview are listed in the references section.

Web Service Patterns: Java Edition

Paul B. Monday, Apress 2003.

In this book Paul B. Monday starts with pointing out that conventional object-oriented design might not be the ideal choice for building web services. According to the author the design of web services requires that the business logic is divided into domain objects and "process" objects that perform the manipulation of data (page 32). However, this distinction is not the main point of the book, and is thus not described in detail.

The contribution is instead the 15 architectural patterns that are presented. These patterns can roughly be divided into three categories: Basic service structure, Infrastructure, and Messaging.

The *basic service structure* patterns describe four basic types of web services: Business object, Business Object Collection, Business Process and Asynchronous Business process. The difference between these types of services is the concepts they represents (object and process respectively), the type of service hence affect its granularity.

Infrastructure patterns describe how to implement physical architecture tiers in Java. The book is very focused on Java, so the in the code example Java Remote Method Invocation (RMI) is used to communicate between tiers.

The *messaging* patterns describe how to implement the well-known patterns such as publish-subscribe, partial population and data-transfer objects (DTOs, sometimes referred to as "Value objects").

Web Services are not Distributed Objects

Werner Vogels, IEEE Internet Computing, Volume: 7, Issue: 6, 2003, Page: 59- 66

In this article Werner Vogels discuss several common misconceptions about web services. These misconceptions can cause sub-optimal design of web service based systems. The main misconceptions are the following:

Web services are just like distributed objects. Systems based on distributed objects (or stateful COM/EJB components) have the notion of object references, object factories and object instances. These notions do not exist in systems build with web services. Web services do not have any state, nor factories to create instances.

Web Services are RPC for the Internet. The focus of RPC protocols is to relay procedure calls over a network, this includes serializing attribute values and return values. According to the author web services do not exhibit this focus on methods and parameters. The focus of web services is instead on the document that is passed on each web service request. Web services are thus document-oriented in nature, rather than RPC or object-oriented. (This argument assumes that the deprecated SOAP-RPC is not used for web service communication.)

Web Services need HTTP. The author points out that HTTP is just one of the protocols that can be used as transport protocol.

Web Services Interaction Models, Part I: Current Practice

S. Vinoski, IEEE Internet Computing, Volume: 6, Issue: 3, 2002, Page: 89- 91

In this article Steve Vinoski (chief architect, IONA) describe problems that arise when using Web services as interfaces to legacy systems. Basically the problems arise when the legacy system uses a stateful interaction model. The author point out two solutions to this problem (holding state in the web service address URI or let the web service make the state persistent between calls), neither approach is recommended. The author instead suggests that the level of abstraction must be raised for web services. A web service should be implemented at the level of business process flows and business documents rather than mirroring the underlying legacy system structure.

Web Services Architecture

David Booth et. al, W3C Working Draft August 2003, www.w3.org

This W3C draft identifies the major concepts needed to describe web service architectures. The draft document is intended as a "guide to the community", and can also be used to test that the concepts used in an architecture conforms to their proposed definitions.

4.1 Short papers

The four Major Constraints to Loosely Coupled Web Services

David Orchard (BEA), Webservices.org, 2003, http://www.webservices.org/index.php/article/articleprint/1246/-1/24/ Accessed 2004-01-15.

In this short paper ten different techniques that can be applied to achieve loose coupling is discussed. These ten techniques can be applied to overcome the four constraints of loosely coupled web services:

Extensibility and versioning. Here the author points out that constrained interfaces (interfaces with a small, predefined set of methods) are easier to extend with new functionality without changing the interfaces. Rather than adding operations to the "constrained interface" the information structures that are sent as parameters to the methods can be changed.

Late binding is an essential feature for building loosely coupled web services, since it allows the target of a method call to be decided at runtime. The author goes further than this, and also suggests that standardized interfaces would enable software systems to handle a wide variety of decisions in runtime. An example is security, standardized interfaces would enable systems to switch security features at runtime. In this case the security mechanisms are "lately bound" to the running service.

Asynchronous calls are essential to archive loose coupling. In order to use asynchronous call both systems need to know the address of each other, this addressing issue (and others) is being solved by the WS-Addressing standard and discussed in the WS-Callback white paper from BEA.

How to design a service-oriented architecture using Web services

Chris McManaman, www.znet.com.au, 2003

In this short article the author present several practical tips on how to build service oriented systems, these tips are from a real-world project. First, using XSLT for message transformation in highly recommended. Secondly the need for dynamic invocation, possibly by using an internal UDDI registry is recommended. Thirdly a tool to debug SOAP messages is considered invaluable.

The Enterprise Service Bus: Making Web Services Safe for Application Integration

Ronan Bradley (Polarlake), Webservices.org, 2003, http://www.webservices.org/index.php/article/articleprint/1048/-1/24/ Accessed 2004-01-14.

In this paper four problem areas are identified when designing systems that handle SOAP documents; validation, enrichment (extending existing messages with additional content), transformation and exception handling. The author suggests that these four problems as well as "technology gaps" can be mitigated by using a Message Oriented Middleware (MOM).

5 References

5.1 Books

Allen, P., and Frost, S., 1998, "Component-Based Development for Enterprise Systems: Applying the Select Perspective," Cambridge University Press.

Cheesman, J., and Daniels, J., "UML Components", Addison-Wesley 2001.

Connolly, T. and Begg, C., "Database Systems", Third Edition, Addison-Wesley 2002.

Monday, P.B, "Web Service Patterns: Java Edition", Apress 2003.

Sundblad, S. and Sundblad P., "Design Patterns for Scalable Microsoft .NET Applications", Sundblad&Sundblad 2003. Sample chapter accessible from www.2xsundblad.com.

5.2 Articles

Arsanjani, A., "Developing and Integrating Enterprise Components and Services", Communications of the ACM, October 2002, Vol. 45, No. 10.

Channabasavaih, K., Holley, K., Tuggle, E. M., "Migrating to a service-oriented architecture, Part2", IBM developerworks, December 2003, http://www-106.ibm.com/developerworks/webservices/library/ws-migratesoa2/, Accessed 16 January 2004.

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 January 2004.

Dumas, M., O'Sullivan, J., Heravizadeh, M., Edmond, D., and Ter Hofstede, A., "Towards a Semantic Framework for Service Description," Proceedings of the 9th IFIP Conference on Database Semantics, 2001.

Fielding, R. T., "Architectural Styles and the Design of Network-based Software Architectures", Doctoral dissertation, University of California, Irvine, 2000.

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 January 2004.

Parnas, D., "On the Criteria to be Used in Decomposing Systems Into Modules", Communications of the ACM, December 1972.

Piccinelli, G., Salle, M., Zirpins, C., "Service-Oriented Modelling for e-Business Application Components", HP Labs Technical Report HPL-2001-123, June 2001.

Prescod, P., “Second Generation Web Services”, February 6, 2002, www.xml.com/pub/a/2002/02/06/rest.html, accessed 13 January 2004.

Vinoski, S., “Web Services Interaction Models, Part I: Current Practice”, IEEE Internet Computing, Volume: 6, Issue: 3, 2002, Page: 89- 91.

Vogels, W., “Web services are not distributed objects“, IEEE Internet Computing, Volume: 7, Issue: 6, 2003, Page: 59- 66.

Wald, E., and Stammers, E., “Out of the Alligator Pool: A Service-Oriented Approach to Application Development,” EAI Journal, March 2001.