# Design Framework for Domain-Specific Service Interfaces

George Feuerlicht, Sooksathit Meesathit

Department of Software Engineering, Faculty of Information Technology,
University of Technology, Sydney,
PO Box 123 Broadway Sydney NSW 2007 Australia

**Abstract.** Following the rapid evolution of Web services standards and technologies over the last three years many organizations are now beginning to make significant investments in the implementation of Web services applications. However, so far only limited attention has been paid to design issues for service-oriented applications. This paper describes a design framework for domain-specific service interfaces. The design framework provides guidance for transformation of a document-oriented message specification into a definition of service interfaces that can be used to generate Web services. We illustrate the design framework using an example based on the Open Travel Alliance (OTA) specification.

## 1 Introduction

While some of the features required for the implementation of secure and reliable e-business applications using Web services are still under development many organizations are making significant commitments to Web services standards and technology platforms. As with other technology platforms, the success of large-scale development projects using Web services will to a large extent depend on effective design and development methodologies used in the construction of application systems. As Web services constitute basic building blocks of service-oriented applications, decisions about what constitutes a service, which operations should the service support, and what service interfaces should be exposed are of vital importance and will determine the quality and reliability of Web services applications. Specification of stable, well-designed Web service interfaces is a key requirement for ensuring high level of interoperability in complex e-business applications. Web services implementation projects conducted in the absence of a design framework are likely to suffer from poor reuse and extensibility as poorly designed interfaces lead to duplication of functionally and poor maintainability of applications. Web services design is an active area of research, but at present there are no comprehensive design frameworks that can be used to assist designers with large-scale Web services projects. Most existing approaches describe Web services design in the context of enterprise application development and rely on object-oriented methods or component-based techniques. For example, Papazoglou and Yang [1] focus on Web services composition based on business process analysis, transforming business

Feuerlicht G. and Meesathit S. (2004).
Design Framework for Domain-Specific Service Interfaces.
In *Proceedings of the 2nd International Workshop on Web Services: Modeling, Architecture and Infrastructure*, pages 109-115
DOI: 10.5220/0002686001090115

process definitions to Web services interface definitions. The design methodology is based on the concepts of coupling and cohesion and the resulting set of Web services is described using a Web services flow language (e.g. WSFL). Levi and Arsajani [2] use a component-based approach, first decomposing a business domain into main business processes, describing these business processes in the form of use cases, and then using this information to design software components. Other approaches include methods based on Model-Driven Architecture (MDA) [3], and Design by Contract methodology [4]. Hammond [5] proposes the use of UML activity diagrams to model business processes and information flows, translating UML models into WSDL descriptions. Alternatively, existing specifications of business processes defined using e-business standard such as RosettaNet can be used as a starting point for Web services design. Masud [6] demonstrates how RosettaNet PIP (Partner Interface Process) specifications can be translated into WSDL and BPEL4WS definitions. Web services are modeled from RosettaNet PIP specifications mapping actions and their corresponding document schemas to Web service operations.

### 1.1 Design of Service Interfaces for e-Business Applications

In the context of e-business applications the use of Web services represents a shift from a document-centric to a service-oriented model for e-business communication [7]. This will have a major impact on the design and implementation methods used for development of e-business applications, making most existing methods unsuitable. Rather then considering the design of individual enterprise applications we focus on the problem of defining industry domain-specific service interfaces. This is an important distinction as the key benefits of service orientation can be only achieved if a consistent set of Web service interfaces is defined and used across an entire industry domain (e.g. travel). This ensures that service providers (e.g. airlines, hotels, etc) publish the same service interfaces, avoiding the need to interpret the semantics of the interface for individual cases. The task of designing domain-specific service interfaces is conceptually similar to designing a programming API (Application Programming Interface); such APIs are used extensively in programming environments (e.g. J2EE). More recently, APIs are being defined to facilitate interoperability for learning technology platforms under the auspices of the Open Knowledge Initiative consortium [8]. This MIT led initiative aims to provide specification for educational services in the form of Java APIs to enable the sharing educational objects across universities. The benefits of standardized service interfaces include reusability, extensibility, and maintainability and lead to significant application development productivity gains. In this paper we describe a design framework for Web service interfaces that uses a industry domain standard specification as a starting point and produces interface definitions for Web services (section 2). The approach is illustrated using a travel application example based on the OTA (Open Travel Alliance) specification [9]. In conclusion (section 3) we discuss the benefits of standardization of domain-specific Web service interfaces.

## 2 Domain-specific Service Interface Design

Existing e-business domain standards are a good starting point for developing domain-specific service interfaces as they capture extensive domain expertise and contain comprehensive business models for a given industry sector. Industry domain standards have been defined in most industry sectors, using EDI (Electronic Data Interchange) format, ebXML, RosettaNet, or various industry-wide specifications such OTA (Open Travel Alliance), and HL7 [10]. Existing industry domain standards are mostly document-centric, i.e. they use document exchange as an interoperability mechanism. This limits the interoperability and scalability of e-business applications as the number of business partners increases and the complexity of the specification grows. Web services remove the need to use document exchange as the interoperability mechanism by providing a homogeneous application deployment environment irrespective of the underlying technology platforms used by individual partner enterprise applications. The key benefit of this approach is that e-business applications can be *programmed*, in effect creating *virtual* applications operating transparently across multiple partner computing environments. The success of this approach is critically dependent on designing a set of standard service interfaces for a given industry domain that can then be used consistently across all applications.

### 2.1 Travel Application Example

We illustrate our design approach using an example Travel Application based on the OpenTravel Alliance consortium specification. OTA defines message payloads using XML Schema for various aspects of the travel business, including air travel, hotel accommodation, and car rentals. Comprehensive message formats specify the information required for various business transactions. For example, OTA_AirAvailRQ message format is a schema specification for a request for flight availability information given a pair of cities on a specific date for specific number and type of passengers. OTA_AirAvailRQ message contains a large number of schema elements including passenger travel preferences (e.g. diet, seating preferences, etc.). OTA specification is based on the request/response paradigm, and a corresponding response message (OTA_AirAvailRS) that contains information about flights matching the request criteria is defined. The design of the messages is optimized with respect to performance over slow networks, maximizing the amount of information transmitted within a single message payload. Each message contains complex data structures and inherently represents a complex business process that the receiving partly needs to map to their enterprise applications. Using Web services to transmit messages with such complex data structures, while possible (i.e. using the document style binding), would not take advantage of the benefits of service-oriented computing, and result in limited scalability characteristic of document-centric e-business applications.

Our approach is to de-compose complex business processes into elementary business functions (i.e. business functions that cannot be further decomposed) and identify the corresponding information requirements. We note that although, this is a

bottom-up composition approach, we rely on previously performed top-down analysis of information requirements and business processes implicit in the OTA message structures and accompanying description of business processes.

### 2.2 Identifying Operations

Given the OTA message format specifications, the task is to convert it into a set of well-defined (domain standard) service interfaces that can be used to accomplish a given business process, for example airline travel booking. The first step in this process is identification of operations; this is then followed by specification of input and output parameters for each operation.

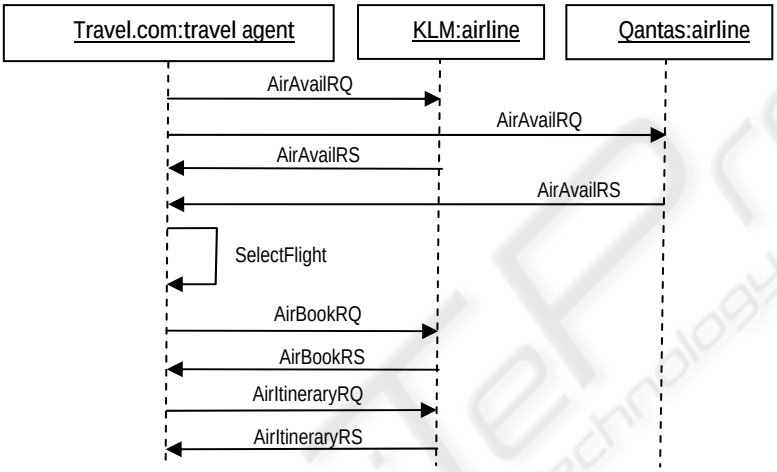


**Figure 2.** Air travel booking process represented as a Sequence diagram

Airline travel booking typically consists of several discrete operations that can be modeled using a Sequence Diagram as illustrated in Figure 2. We make a number of simplifying assumptions in our Travel Application example, including that the travel agent interacts with only two airlines (KLM and Qantas), and that the flight is between a single pair of departure and destination cities. We assume that the business process operates in the following way:

The travel agent sends a travel availability request messages to both airlines specifying the departure and destination city, the date of travel, and other relevant information. When the responses from both airlines are received, the travel agent selects a particular flight based on some criteria (e.g. price), and possibly, after consulting the traveler, and makes a booking with the selected airline. Finally, the travel agent makes a request for itinerary information; optionally this could be followed by additional requests (e.g. request to add royalty points). We summarize the steps in the Travel Booking business process below showing the corresponding OTA message request/response messages in the parentheses.

1. Availability check (AirAvailRQ/AirAvailRS)
2. Flight booking request (AirBookRQ/AirBookRS)
3. Request for itinerary (AirItineraryRQ/AirItineraryRS)
4. Request for rules and conditions (AirRulesRQ/AirRulesRS)
5. Request to add royalty points (AirRoyaltyRQ/AirRoyaltyRS)

For completeness we also define a cancellation request (AirCancelRQ/AirCancelRS) not specified by OTA. Individual messages in the sequence correspond to *elementary* business functions. Elementary business functions can be regarded as candidate Web services operations. Using this approach the granularity of Web service operations is determined by the corresponding elementary business functions. Larger granularity operations can be created by composition, i.e. by implementing an interface that uses the basic operations to implement a more complex business process. For example, the lowest airfare search operation (LowestAirFareSearch) could be implemented using availability request operations (AirAvailRQ) executed for various airlines, and by determining the minimum airfare.

We note here that we are not concerned with workflow aspects of the business process, and purely use the Sequence Diagram as a modeling tool to facilitate the design of service interfaces.

### 2.3 Defining Interfaces

Service interface definitions consist of specification of operations and assignment of input and output parameters for each operation. Following the identification of operations in section 2.2 above, the interfaces (input and output parameters) for each operation are defined. As noted earlier (see discussion in section 2.1 above), the original OTA messages contain complex data structures combining multiple business functions into a single message, and include a large number of optional data items to accommodate various customization requirements – this is characteristic of the document-centric approach and leads to unnecessary dependencies that inhibit evolution of the application. An important interface design goal is to minimize the exposure of metadata in order to reduce inter-dependencies between applications. The corresponding message structures need to be decomposed and relevant parameters for individual operations identified. The interface should only contain parameters that are required for a specific operation. For example, the original OTA AirAvialRQ message contains many data items that are not directly required to check flight availability, and the interface can be reduced to a relatively small number of input and output parameters. It is useful to classify the operations according to the type of the request performed. We classify operations into three types: query operations that execute a query on a remote resource (e.g. availability check), transactions that perform a transaction on a remote resource (e.g. booking request, or flight cancellation), and document request (e.g. itinerary request). The type of operation determines the Web services binding style; typically the binding style for query requests and transactions is *RPC* (Remote Procedure Call), and for document requests is *document*. Other approaches use different classifications, for example J2EE specification for Web services design uses three categories: information Web services, transactions, and business processes Web services [11]. Table 1 shows operations and the corresponding interface definitions for the Airline Booking Service based on the business process described in section 2.2 above.

| Operations | Type | Input | Output |
|---|---|---|---|
| AirAvail | Query (RPC) | OriginalLocation<br>DestinationLocation<br>DepartureDate<br>CabinPref {Optional}<br>Airline {Optional}<br>FlightNumber {Optional} | DepartureTime<br>ArrivalDate<br>ArrivalTime<br>Airline<br>FlightNumber<br>AirFare |
| AirBook | Transaction (RPC) | Airline<br>FlightNumber<br>DepartureDate<br>DepartureAirport<br>ArrivalAirport<br>TravelerName | BookingReferenceID |
| AirItinerary | Document | BookingReferenceID | TravelerName<br>Airline<br>FlightNumber<br>DepartureAirport<br>DepartureDate<br>DepartureTime<br>ArrivalAirport<br>ArrivalDate<br>ArrivalTime<br>ActionCode<br>BaseFare<br>Taxes |
| AirCancel | Transaction (RPC) | BookingReferenceID | CancellationID<br>CancellationFee {Optional} |

**Table 1.** List of operations and corresponding parameters for Airline Booking Service

As described in our earlier publication [7], given the specification of interfaces the corresponding WSDL definitions for the Airline Booking Service can be generated, and the Web service implemented and deployed.

## 3. Conclusion

In this paper we have presented a simple design framework for domain-specific service interfaces and we have illustrated this approach using a travel example. The approach uses industry domain document-centric message specification as a starting point and produces a set of service interfaces that can be implemented using Web services. Using this design framework we map message specifications to a sequence diagram and then transform the sequence diagram into service interface definitions, minimizing the exposure of metadata. The interface encapsulates details of the service implementation, so that for example the AirCancel operation is implemented as an RPC call with a single input parameter (BookingReferenceID).

Using standard service interfaces across an industry domain such as travel or education results in an environment where e-business applications can be implemented as a series of calls to remote services, with significant application development productivity gains, and greatly improved application maintainability. Service providers can expose additional (non-standard) service interfaces to provide specialized business functions, if required. Another benefit of this approach is that evolution can be supported via interface versioning, so that existing interfaces can be maintained to support *legacy* applications. This avoids many of the problems associated with evolution of document-centric, message-based standards (e.g. EDI).

In conclusion, service-oriented computing based on Web services standards and technologies provides an opportunity to finally address many of the issues that inhibit interoperability and automation of e-business applications. Industry standard service interfaces are a key component of the service-oriented approach to e-business applications. This paper illustrates how such interfaces can be developed from document-centric message structures using a simple design framework.

## References


1. Papazoglou, M.P. and J. Yang: Design Methodology for Web Services and Business Processes. In Proceedings of the 3rd VLDB-TES Workshop. Hong Kong. Springer (2002)
2. Levi, K. and A. Arsanjani: A goal-driven approach to enterprise component identification and specification. Communications of the ACM. Vol. 45:(10). (2002) 45 - 52
3. Frankel, D. and J. Parodi: Using Model-Driven Architecture™ to Develop Web Services. IONA. (2002) http://portals.devx.com/assets/iona/2974.pdf
4. Meyer, B.: Object-oriented software construction. 2nd edn. Prentice Hall, Upper Saddle River, N.J. (1997)
5. Hammond, J.: Introducing Web services into the software development lifecycle. Rational software Corporation. (2002) http://www.rational.com/media/whitepapers/TP033.pdf
6. Masud, S.: Use RosettaNet-based Web services, Part 1: BPEL4WS and RosettaNet. DifferentThinking. (2003) http://www106.ibm.com/developerworks/webservices/library/ws-rose1/
7. Feuerlicht, J. Implementing Service Interfaces for e-Business Applications, The Second Workshop on e-Business (WeB 2003), December 13-14, 2003, Seattle, USA.
8. Thorne, S. et al, OKI Architecture Overview, http://web.mit.edu/oki/learn/papers.html (March, 2002)
9. OTA 2002, Available from: http://wwww.opentravel.org/2002a.cfm
10. HL7 Message Development Framework Version 3.3, December 1999, http://www.hl7.org/
11. Designing Web Services with the J2EE 1.4 Platform - Early Access. (2003) http://java.sun.com/blueprints/guidelines/designing_webservices/