ISeB (2005) 3: 363–376
DOI 10.1007/s10257-005-0015-4

ORIGINAL ARTICLE

George Feuerlicht

Design of service interfaces for e-business applications using data normalization techniques

Published online: 26 July 2005

Abstract Web Services are being increasingly used for implementing large-scale e-business applications, but at present there is a lack of comprehensive methodologies based on sound engineering principles that can guide designers of service-oriented applications. This lack of methodological support is likely to lead to poorly designed and difficult to maintain e-business applications. In this paper we describe a design method for service-oriented applications that applies data engineering principles and the theoretical framework of data normalization to service design to produce a set of orthogonal services with normalized interfaces. We consider the impact of increasing service granularity on cohesion and coupling of service operations, and discuss associated design trade-offs. We use a travel example based on the Open Travel Alliance specification to illustrate how a document-oriented standard can be transformed into a set of well-designed service interfaces.

Keywords Web Services · Service interfaces design · Data normalization · e-business applications

1 Introduction

The general trend towards service-oriented computing and universal support of Web Services (W3C 2004) across all major technology platforms presents

G. Feuerlicht
Department of Software Engineering, Faculty of Information Technology,
University of Technology, Sydney, NSW, Australia
Tel.: +123-45-678910, Fax: +123-45-678910, E-mail: jiri@it.uts.edu.au

a new opportunity to address the problem of e-business (electronic business) interoperability. The combination of relatively low implementation costs and wide acceptance of Web Services standards by technology vendors is likely to make this approach a preferred option for the implementation of e-business applications. However, as history of electronic commerce over the past two decades indicates the problem of automating e-business transactions is not purely a technological issue and requires addressing the more intractable problem of application interoperability, more specifically the issues associated with disparate business semantics used by different partner organizations.

Most existing e-business approaches such as (EDI 2004), (RosettaNet 2004) and (ebXML 2004) rely on standardization of business documents and processes. Although successful in some industry sectors, wide adoption of such standards has been slow and limited by their complexity, inflexibility and high implementation costs. Many industry domains have ongoing standardization efforts that aim to produce standard documents for communication between e-business partners. For example, the Open Travel Alliance (OTA) consortium specification (OTA 2004) defines XML Schemas and corresponding usage scenarios for messages that support business activities in the travel industry. A common characteristic of the above approaches is that they emulate paper-based business communications by shipping structured documents as message payloads; i.e. document exchange forms the basis of e-business communications. Such document-oriented standards require that organizations agree on the precise formatting, structure and semantics of the documents being used for communication. Industry-wide document standards tend to be highly complex as they need to accommodate the requirements of all key industry players and include many optional elements to support diverging needs of partner organizations. Conformance to document-centric standards involves mapping of standard documents into data structures used internally by each partner organizations and typically results in extensive data transformation to ensure compatibility with internal data standards. It can be argued that the requirements of today's dynamic e-business environment cannot be satisfactorily addressed using an interoperability mechanism based on document interchange as document-centric approaches suffer from limited scalability and flexibility. Scalability in this context is the ability of the solution to accommodate a large number of autonomous partner organizations with independently evolving business semantics, without unduly increasing the complexity of the specification. Flexibility is needed to allow the evolution of the standard specification over time to accommodate changes in business processes and data semantics without impacting on existing applications. Feuerlicht (2003) gives a more detailed discussion of the limitations of document-centric e-business and the opportunities to address these limitations with the service-oriented approach.

Web Services can be used to implement business interactions as services over the Internet. Instead of using document interchange, service-oriented applications interact via service interfaces that externalize operations and encapsulate data structures. Advantages of the service-centric approach include improved software reliability simplified development, and support for

evolution of interfaces (Bieber 2001). Web Services remove the need to use document shipping as a mechanism for application interoperability by providing an essentially "homogeneous" application deployment environment irrespective of the underlying technology platforms used by individual partner applications. Important advantage of the service-centric approach is that service interfaces can be designed to significantly limit exposure of data, reducing the complexity of message document structures and eliminating data redundancy. The problem of standardizing document formats and data semantics is reduced to a more manageable task of standardizing service interfaces for a given application domain, e.g. travel, healthcare, etc.

Interoperability of service-oriented applications relies on well-defined service interfaces used consistently across the application domain. Standardization of service interfaces ensures that service providers (e.g. airlines) publish identical interfaces, avoiding the need to interpret the semantics of interfaces published by individual service providers. The abstraction level of standardized, domain-specific service interfaces is closely related to business requirements for a particular application domain; in effect, standard service interfaces provide application developers with a high-level API (Application Programming Interface) for building domain-specific applications (Feuerlicht 2003).

Service interfaces must be well-designed to ensure reuse, avoid duplication in functionality, and minimize interdependencies between services. Web Services design is an active area of research, but at present there are no comprehensive design methodologies that can be used to assist designers with large-scale Web Services projects. In order to be effective, Web Services design methodology must be based on sound engineering principles and provide guidelines that assist designers in making design choices (Papazoglou 2002).

In this paper we first briefly review current Web Services design research (Sect. 2) and then describe a service interface design method based on data engineering principles and normalization of service interfaces (Sect. 3). We then briefly consider the impact of varying the granularity of service operations on cohesion and coupling (Sect. 3.3). We illustrate our design approach using a travel application example based on the OTA specification, showing how a document-centric specification can be transformed into a set of well-designed service interfaces. In conclusions (Sect. 4) we summarize the main contributions of this paper and identify opportunities for further research.

2 Web Services design

Most existing approaches describe Web Services design in the context of enterprize application development and rely on object-oriented methods or component-based techniques for designing Web Services applications. For example, Ambler (2002) proposed a method for deriving Web Services from UML models. The method involves identifying class contracts that define public interfaces for a given class, and combining the contracts to reduce the number of services resulting in a cohesive collection of classes called domain

packages (i.e. groups of highly coupled classes). The final set of service contracts are mapped to Web Services operations, and the input and output parameters of the operations defined using XML schemas. Ambler's method is an example of a bottom-up Web Service design approach that defines Web Services on top of existing components or objects and is useful for migration to service-oriented architectures. The main focus of this method is grouping of highly coupled classes into coarser components called domain packages, and refining the resulting component interfaces to produce larger grained services that are exported as Web Services.

Papazoglou and Yang (2002) describe a design methodology for Web Services that transforms business processes into sets of collaborative Web Services. The methodology provides service design guidelines based on the principles of minimizing coupling and maximizing cohesion to ensure that the resulting services are self-contained, modular, extendable and reusable, and produces definition of WSDL Web Service interfaces and WSFL service flow models. The methodology also covers non-functional service design guidelines including service provisioning strategies and service policy management models. The methodology uses both top-down and bottom-up approaches to design interfaces for composite Web Services. Top-down approach is used to analyze business processes and to identify service invocations required to implement each activity within the scope of the process, and the bottom-up approach is used to map existing services provided by external service providers to usage interfaces of the business processes.

Levi and Arsajani (2002) proposed component-based Web Services design method based on identifying business processes and dividing the problem domain into functional areas based on departmental boundaries, business process boundaries and value chains. The functional areas (i.e. major business areas) are mapped to enterprize components which are then decomposed to identify their constituent business processes creating a goal model using Goal-Service Graphs. The objective of this step is to identify high-level business goals, their sub-goals and services required to achieve the goals. The main benefit of the goal model is that it allows designers to define services based on business needs. The services identified in the Goal-Service Graphs are then assigned to enterprize components that are responsible for providing the services. Service identification involves making decisions about granularity of the services. Specifications for enterprize components are created defining the pre and post-conditions for each service and an abstract specification of component *behavior*. The main focus of this methodology is on designing enterprize components suitable for transformation into Web Services by exporting their interfaces.

Alternatively, existing specifications of business processes as defined using e-business standards such as RosettaNet (http://www.rosettanet.org) can be used as a starting point for Web Services design. Masud (2002) demonstrates how RosettaNet Partner Interface Process (PIP) specifications can be translated into WSDL and BPELWS definitions (BPELWS 2003). Web Services are modeled from RosettaNet PIP specifications mapping actions and their corresponding document schemas to Web Service operations. Input and output parameters are described in WSDL, and the partner

roles, process and data flows between the partners are mapped to a flow language specification. The methodology describes how relevant information can be extracted from RosettNet PIP specifications and the corresponding document schemas, and used to define Web Service interfaces and choreography descriptions of the interaction semantics between business partners. Designing Web Services from existing e-business standards enables design of interfaces and interaction dialogues based on industry-wide standard business processes and vocabularies. This avoids the need to define Web Services for individual partner organizations and results in significantly improved interoperability. RosettaNet standard consists of specifications that include dictionaries, implementation framework, and PIP specifications. Masud focuses on deriving Web Services interface and choreography descriptions from corresponding PIP specifications and associated Message Structure definitions. The initial phase of the methodology involves definition of Web Services operations and their input and output messages using WSDL. Relevant messages are identified from choreography diagrams and PIP specifications. WSDL message elements based on RosettaNet message definition are defined and corresponding operations specified by mapping messages into operations. The second phase of the methodology deals with creating BPEL descriptions for interactions between trading partners using Web Services defined during the first phase. The methodology uses business process definitions and choreography information defined using UML diagrams and associated tables within the PIP specification as the basis for BPELWS specifications. Partner roles defined in PIP are mapped into BPELWS, and then the PIP choreography is implemented as choreography of abstract and executable business processes in BPELWS.

Other approaches include design methods that rely on functional decomposition to produce modular Web Services (Wieringa et al. 2003), methods based on translating UML activity diagrams into WSDL descriptions (Hammond 2002), and methods based on Model-Driven Architecture (MDA) such as (Frankel 2002).

2.1 Design of domain-specific services

Web Services are being increasingly used to implement e-business applications in various industry sectors. As an alternative to considering the design of Web Services for individual enterprize applications or components (as in most of the above papers) we focus on the design of domain-specific services. Travel industry examples of domain-specific Web Services implementations include Galileo Web Services (Fontana 2002; Schwartz 2002), Dollar Rent A Car (http://www.dollar.com/), and Southwest Airlines (Metz 2001). At present, such applications mostly deploy Web Services to transmit XML documents via the document style binding using SOAP as the transport mechanism. This mode of operation, while providing a relatively inexpensive transport mechanism for interchange of electronic documents, suffers from the same interoperability limitations as the document-centric approaches described in Sect. 1. In order to take full advantage of Web Services,

e-business applications need to adopt the service-oriented approach and focus on designing interoperable service interfaces. The key issue is developing a suitable methodological support for such design activity.

The task of designing domain-specific service interfaces is conceptually similar to the design of programming interfaces or classes in object-oriented programming, and we can draw from the extensive literature on this topic to identify guiding principles for interface design, e.g. (Yourdon and Constantine 1979; Meyer 1997), etc. Two design principles are of particular relevance: maximizing cohesion and minimizing coupling of service operations (i.e. methods in the context of object-oriented programming). Maximizing method cohesion refers to the requirement for methods to implement a single conceptual task and is closely related to the concept of orthogonality that requires that functionality of methods does not overlap. Minimizing method coupling (i.e. interdependencies between methods that cause changes in one method to necessitate changes to related methods), results in improved robustness of applications and ability to accommodate change. Applying these principles to service interface design leads to improved clarity of interfaces, reduction in undesirable side effects, and improved flexibility of applications Venners (1998, 2002). In the following section (Sect. 3) we interpret these principles using data engineering concepts and use data normalization to provide the theoretical foundation for service interface design in order to achieve high levels of orthogonality.

3 Design of service interfaces

In this section we describe a methodology for the design of service interfaces. We view the task of service interface design from a data engineering perspective, using OTA message specifications as a starting point and decomposing the message structures and associated business processes into a set of orthogonal service operations with normalized interfaces. OTA defines a large number of message (document) formats addressing various aspects of travel industry activities and providing a comprehensive specification of information requirements for travel applications. We base our design examples of the Flight Booking Service on simplified versions of the airline availability request/response messages: OTA_Air_AvailRQ and OTA_Air_AvailRS.

Defining service interfaces involves specification of operations and corresponding input and output parameters. This task is similar to designing method signatures in the context of object-oriented design, and requires that suitable candidate methods are identified. The key guiding principles for the design of service interfaces are orthogonality (i.e. the functionality of operations should not overlap), maximization of method cohesion, and minimization of method coupling (Feuerlicht and Meesathit 2004). The proposed design framework consists of three steps:

(1) Identification of candidate operations using business function decomposition
(2) Refining interface design using interface normalization

(3) Adjusting granularity of operations based on interface parameters.

Our approach is based on decomposition of complex business functions into elementary business functions, i.e. simple atomic functions that cannot be further decomposed (Eriksson and Penker 2000; Larman 2001). We then map these elementary business functions to *simple* (service) operations and identify input and output parameters using the corresponding OTA message data structures (Sect. 3.1). This approach is consistent with maximizing method cohesion as elementary business functions typically accomplish a single conceptual task and exhibit high levels of cohesion. We then perform normalization of service interfaces eliminating redundant input and output parameters (Sect. 3.2), and finally consider combining operations to fine-tune the granularity of services (Sect. 3.3).

3.1 Identification of candidate operations

Consider the Flight Booking Business Function Hierarchy model illustrated in Fig. 1.

For the purpose of this example we make a number of simplifying assumptions. We assume that a flight booking is for a single flight segment, (i.e. a direct flight between the origin and destination location), and that at most one flight is available for a given flight enquiry (i.e. flight destination and departure date combination). Figure 1 illustrates function hierarchy that results from progressive function decomposition until elementary *leaf* functions are identified (shaded on the diagram). Table 1 contains the corresponding descriptions of e-business functions for the Flight Booking process.

Table 1 Flight booking elementary business functions

Business function	Description
Flight enquiry	The travel agent requests flight availability for a given pair of origin and destination cities and a departure date. The airline response includes flight number, departure airport, departure time, arrival date, arrival time, and arrival airport
Seat Enquiry	The travel agent requests seat availability for a particular flight specifying the flight number, departure airport, arrival airport, departure date, and cabin type (e.g. economy). The airline responds with the quantity of seats available
Price enquiry	The travel agent requests pricing information specifying the flight number, departure airport, arrival airport, departure date, and cabin type. The airline responds with base fare and base fare code
Seating request	The travel agents request a traveler seating preference (e.g. an aisle seat). The airline responds with a seat number
Meal request	The travel agent requests a traveler meal preference (e.g. Vegetarian meal). The airline responds with special meal confirmation

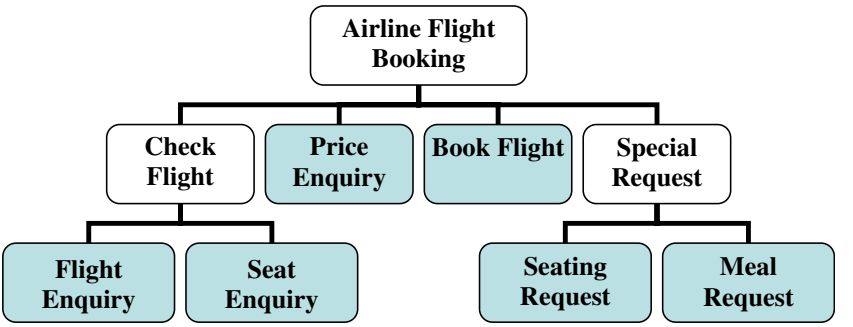

Fig. 1 Flight booking business function hierarchy

We note that similar result can be obtained by modeling the interaction between a travel agent and an airline using a sequence diagram. Each step in the Sequence Diagram dialog produces a request/response message pair and corresponds to an elementary business function (Feuerlicht and Meesathit 2004). We can now map the elementary (leaf) business functions in Fig. 1 to candidate service operations and define interfaces (input and output parameters) for each operation using corresponding OTA message structures as shown in Table 2. Using this approach the granularity (i.e. level of

Table 2 Candidate Operations for the Flight Booking Service

Service operations	Input parameters	Output parameters
FlightEnquiry (Query Request)	OriginLocation DestinationLocation DepartureDate	FlightNumber DepartureAirport DepartureTime ArrivalAirport ArrivalDate ArrivalTime
SeatEnquiry (Query Request)	FlightNumber DepartureAirport ArrivalAirport DepartureDate CabinType	Quantity
PriceEnquiry (Query Request)	FlightNumber DepartureAirport ArrivalAirport DepartureDate CabinType	FareBasisCode BaseFare
BookFlight (Update Request)	FlightNumber DepartureAirport DepartureDate TravelerName CabinType	BookingReferenceID
SeatingRequest (Update Request)	BookingReferenceID SeatPreference	SeatNumber
MealRequest (Update Request)	BookingReferenceID MealPreference	MealType

aggregation) of service operations is determined by the granularity of the corresponding elementary business functions, and this leads to a *fine-grained* solution.

3.2 Refining interface design

As noted earlier, important interface design goal is to minimize coupling between services. We now need to ensure that the service interfaces defined in Table 2 are consistent with the principle of minimizing method coupling, and at the same time maintain a high level of cohesion. Methods (operations) can be regarded as *data transformers* and minimization of coupling involves defining input and output parameters so that interdependencies between methods and side-effects are minimized (Venners 1998, 2002). This leads to consideration of the data properties of parameters, and the general rule that all parameters must be used by the method *as data*, i.e. not to control the execution of the method. Minimization of coupling can be interpreted as a requirement for elimination of redundant interface parameters so that both input and output parameter sets are *minimal*. Minimality in this context implies that parameters are mutually independent; i.e. cannot be derived from each other. Treating method parameters as data leads to consideration of functional dependencies between parameters and the application of data normalization rules (Codd 1971). Functional dependencies (FDs) provide the underlying theoretical foundation for normal forms used extensively for elimination of redundancy in database design. We apply these principles here to reduce method coupling by removing redundant data parameters. So that, for example, if Flight Number determines Departure Time, then the Departure Time parameter should not be included in the parameter list of operations that also include the Flight Number. To capture this requirement we formulate the following interface design rule:

> Rule 1: *input and output parameter sets should be minimal, i.e. parameters must be mutually independent*

Venners (2002) classifies methods according to the type of the request performed into three types: state-view methods (query operations that return data in output parameters, given query formulated using input parameters), state-change methods (methods that result in update, insert, and delete transactions based on input parameters), and utility methods (notifications, etc.—methods that do not use data parameters). We use a similar classification, and categorize methods into *Query* and *Update* requests. We formulate an additional rule for query request methods to maximize cohesion by removing extraneous parameters not generated by the method from the output parameter set:

> Rule 2: *output parameters must be fully functionally dependent on the input parameter set*

Rule 2 ensures that method output does not include parameters that are not directly generated by the method from the input parameter set. In effect, the parameters of a query request form a relation where the input parameters

are the key attributes and output parameters are non-key attributes. Data normalization rules can be applied to this situation to ensure that output parameters are fully functionally dependent on the input parameter set. Satisfying the conditions of Rules 1 and 2 ensures that the relation formed by the query request input and output parameters is in the BCNF normal form (Boyce-Codd Normal Form) and therefore does not contain redundant parameters (Date and Fagin 1998). This consideration does not directly apply to update request methods as the values supplied via input parameters typically create new records (i.e. insert records) or change existing records (i.e. update or delete existing records), and return a value that identifies the new record (e.g. BookingReferenceID, for flight booking), or an acknowledgement (e.g. when ordering a special meal). Assuming the following five functional dependencies for our travel scenario, we can now apply data normalization rules to candidate operations in Table 2:

F1 = {OriginLocation, DestinationLocation, DepartureDate → FlightNumber}
F2 = {FlightNumber → DepartureAirport, DepartureTime, ArrivalAirport, ArrivalTime}
F3 = {FlightNumber, DepartureDate → ArrivalDate}
F4 = {FlightNumber, DepartureDate, CabinType → Quantity}
F5 = {FlightNumber, DepartureDate, CabinType → BasicFareCode, BasicFare}

Table 3 Normalized interfaces for Flight Booking Service

Service operations	Input parameters	Output parameters
FlightEnquiry (Query Request)	OriginLocation DestinationLocation DepartureDate	FlightNumber
ScheduleEnquiry (Query Request)	FlightNumber	DepartureAirport DepartureTime ArrivalAirport ArrivalTime
ArrivalEnquiry (Query Request)	FlightNumber DepartureDate	ArrivalDate
SeatEnquiry (Query Request)	FlightNumber DepartureDate CabinType	Quantity
PriceEnquiry (Query Request)	FlightNumber DepartureDate CabinType	FareBasisCode BaseFare
BookFlight (Update Request)	FlightNumber DepartureDate TravelerName CabinType	BookingReferenceID
SeatingRequest (Update Request)	BookingReferenceID SeatPreference	SeatNumber
MealRequest (Update Request)	BookingReferenceID MealPreference	MealType

Applying Rule 1 and using functional dependency F2 we eliminate the parameters DepartureAirport and ArrivalAirport from the input parameter sets of SeatEnquiry, PriceEnquiry, and BookFlight operations as these parameters can be derived from FlightNumber. Similarly, using Rule 1 and F2 we can eliminate DepartureAirport, DepartureTime, ArrivalAirport, and ArrivalTime from output parameters of operation FlightEqnuiry. This leaves FlightNumber and ArrivalDate in the output parameter set of FlightEnquiry; but this violates Rule 2 as ArrivalDate is partially dependent on input parameter DepartureDate. This leads to the elimination of ArrivalDate from the output parameter set. The resulting set of normalized interfaces (Table 3) includes two new operations: ScheduleEnquiry, and ArrivalEnquiry that maintain functional dependencies F2 and F3, respectively in order to preserve completeness. Applying interface normalization rules to all operations in Table 2 we produce a set of interfaces that is consistent with the design objectives of minimal coupling and maximum cohesion (Table 3).

3.3 Adjusting granularity of operations

Finding an optimal level of granularity for Web Services and individual service operations requires further examination. Coarse-grained operations tend to lack cohesion and suffer from increased levels of coupling, while fine-grained operations increase the number of service interfaces, and consequently the number of procedure calls at runtime. The above analysis leads to normalized service interfaces and results in fine-granularity operations. While this may be theoretically appealing, the associated increase in the number of runtime calls makes this approach difficult to implement in practice given the existing low-reliability and slow response time Internet infrastructure.

3.3.1 Combining operations based on interface parameters

We can use the normalization framework introduced in this section to identify interfaces that can be combined without introducing parameter redundancies. For example, it is possible to combine operations based on common input parameters (i.e. key parameters) without violating the interface normalization rules introduced in Sect. 3.2. For example, query request operations SeatEnquiry and PriceEnquiry share a common input parameter set FlightNumber, DepartureDate, CabineType. Combining the two interfaces produces a composite operation SeatPriceEnquiry, as shown in Table 4:

Table 4 Composite Operation SeatPriceEnquiry

Service operations	Input parameters	Output parameters
SeatPriceEnquiry (Query Request)	FlightNumber DepartureDate CabinType	Quantity FareBasisCode BaseFare

Table 5 Composite Operation SeatingMealRequest

Service Operations	Input Parameters	Output Parameters
SeatingMealRequest Update Request	BookingReferenceID SeatPreference, MealPreference	SeatNumber MealType

This clearly leads to some loss of cohesion as the resulting operation no longer implements a single conceptual task, and in situations where it is used to perform a partial enquiry (e.g. seat availability enquiry only) the output parameter set returns values that are not used by the application. This trade-off can be justified in this instance on the basis that both operations are frequently performed together, and that the benefits of reduced number of operations and runtime procedure calls outweighs the loss of cohesion. Similar considerations apply to update request operations, for example SeatingRequest and MealRequest can be combined into a composite operation SeatingMealRequest as illustrated in Table 5 below:

This time, partial request, e.g. seating request only, produces non-homogeneity in input and output parameter sets, i.e. MealPreference and MealType remain undefined.

3.3.2 Combining operations by composition

As an alternative to combining operations based on common interface parameters, complex operations can be constructed programmatically using elementary operations. For example, a flight between origin and destination locations (e.g. Sydney and London) in general consists of a number of flight segments (e.g. Sydney to Singapore, and Singapore to London) so that the corresponding flight availability request has to check availability for each segment separately and provide programming logic to determine if the entire flight is available. Another example involves the travel agent requesting flight availability information from a number of airlines before making a booking decision based on some criteria (e.g. the lowest price).

4 Conclusions

In this paper we presented a design methodology for service interfaces that can be used to transform document-centric specifications into a set of service interface definitions. The design approach relies on the principles of orthogonality, maximizing method cohesion, and minimizing method coupling, and uses data normalization techniques to avoid externalization of redundant interface parameters. As noted in Sect. 3.3 above using the proposed design framework for Web Service interfaces in e-business applications leads to an increased number of operations for a given Web Service and consequently to a corresponding increase in the number of procedure calls required to implement a specific business function. This represents a challenge given the current Internet environment characterized by unreliable

network connectivity and unpredictable response times, making the programmatic approach using low-granularity operations advocated in this paper only suitable for fast and reliable Intranet environments. However, the proposed design framework facilitates making informed decisions about service granularity based on the theory of normalization applied to service interfaces. As illustrated in Sect. 3.3, composite operations can be constructed from operations with fully normalized service interfaces by combining operations based on the properties of interface parameters, and the impact of loss of cohesion can be evaluated.

We have argued that well-designed service interfaces are a key requirement for e-business interoperability. Service interfaces can be designed to avoid externalizing complex and often redundant data structures that lead to high-levels of interdependency between applications in the document-centric approach. The application of data engineering principles to service interface design has the potential to improve our understanding of the impact of increasing granularity of operations on cohesion and coupling. Further research is needed to determine how elementary operations can be combined into larger granularity operations while minimizing undesirable side effects associated with loss of cohesion and increase in coupling.

Finally we note that the design framework presented in this paper is of general applicability and can be used to transform any document-centric specifications (e.g. EDI document definitions) into well-designed service interface definitions. Importantly, the methodology can be also applied to design of object-oriented applications and components, as data properties of interface parameters play an equally important role in determining cohesion and coupling of methods in object-oriented applications.

References

Ambler SW (2002) Deriving Web Services from UML models, Part 1: Establishing the process. Available on: http://www-106.ibm.com/developerworks/webservices/library/ws-uml1/

Bieber G, Carpenter J (2001) Introduction to Service-Oriented Programming (Rev 2.1) Available on: http://www.openwings.org/download/specs/ServiceOrientedIntroduction.pdf

BPELWS (2003) Business Process Execution Language for Web Services, Version 1.1, 5 May 2003, Available on: ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf

Codd EF (1971). Normalized data structure: a brief tutorial. In: Proceedings of 1971 ACM-SIGFIDET workshop on data description, access and control (San Diego, California, November 11–12, 1971). ACM, New York, pp 1–17

Date CJ, Fagin R (1992) Simple conditions for guaranteeing higher normal forms in relational databases. ACM Trans Database Syst 17(3):465–476, SSN:0362–5915

ebXML (2004), Available on: http://www.ebxml.org

EDI (2004) UNECE. UN/EDIFACT, Available on: http://www.unece.org/trade/untdid/welcome.htm

Eriksson HE, Penker M (2000) business modeling with UML: business patterns at work. Wiley, New York

Feuerlicht G (2003) Implementing Service interfaces for e-business applications. In: Proceedings of the second workshop on e-Business, WeB 2003, (Seattle, USA, December 13–14, 2003). ISSN:1617–9846

Feuerlicht G, Meesathit S (2004) Design framework for domain-specific service interfaces. In: Proceedings of the 2nd international workshop on Web Services: modeling, architecture, and infrastructure, (Porto, Portugal, April 13–14, 2004). INSTICC Press, pp 109–115, ISBN 972–8865–09–0

Fontana J (2002) Galileo travels down Web services path. Network World [Online], April 9, 2002. Available on: http://www.nwfusion.com/news/2002/0429galileo.html

Frankel D, Parodi J (2002) Using Model-Driven Architecture™ to Develop Web Services. IONA. Available on: http://portals.devx.com/assets/iona/2974.pdf

Hammond J (2002) Introducing Web services into the software development lifecycle, Rational software Corporation. Available on: http://www.rational.com/media/whitepapers/TP033.pdf

Larman C (2001) Applying UML and patterns : an introduction to object-oriented analysis and design and the unified process, 2nd edn. Prentice Hall, Upper Saddle River

Levi K, Arsanjani A (2002) A goal-driven approach to enterprise component identification and specification. Commun ACM 45(10):45–52

Masud S (2002) Use RosettaNet-based Web services, Part 1: BPEL4WS and RosettaNet. Different thinking. Available on: http://www106.ibm.com/developerworks/webservices/library/ws-rose1/

Metz C (2001) Testing the waters. PC Magazine [Online], November 13, 2001. Available on: http://www.pcmag.com/article2/0,4149,154693,00.asp

Meyer B (1997) Object-oriented Software Construction. Prentice Hall, Englewood Cliffs. ISBN: 0–13–629155–4

OTA (2004) The Open Travel Alliance website. Available on: http://www.opentravel.org/

Papazoglou MP, Yang J (2002) Design methodology for Web services and business processes. In: Proceedings of the 3rd VLDB-TES workshop (Hong Kong, August, 2002). Springer, Berlin Heidelberg New York, pp 54–64

RosettaNet (2004) Available on: http://www.rosettanet.org/

Schwartz E (2002) Triple A Launches Web Service Airline Reservation System. InfoWorld [Online], August 12, 2002. Available on: http://www.infoworld.com/article/02/08/12/020812hntriplea_1.html

Venners B (1998) Introduction to Design Techniques. Available on: http://www.javaworld.com/javaworld/jw-02–1998/jw-02-techniques.html

Venners B (2002) API Design: the object. Available on: http://www.artima.com/apidesign/object.html, April 26, 2002

W3C (2004), W3C Web Services Activity. Available on:http://www.w3.org/2002/ws/

Wieringa RJ, Blanken HM, Fokkinga MM, Grefen PWPJ (2003) Aligning application architecture to the business context. In: Proceedings of 15th international conference on advanced information systems engineering (CAiSE 2003) (Klagenfurt, Austria, June 16–18, 2003). Springer, Berlin Heidelberg New York, pp 209–225

Yourdon E, Constantine L (1979) Structured design. Prentice-Hall, Englewood Cliffs