Service Granularity Considerations Based on Data Properties of Interface Parameters

George Feuerlicht

Faculty of Information Technology,
University of Technology, Sydney,
P.O. Box 123 Broadway, Sydney, NSW 2007, Australia
jiri@it.uts.edu.au

Abstract. Service granularity is a key determinant of the reusability and maintainability of services. It is evident that mapping existing components directly to Web Services leads to suboptimal design with large number of message exchanges and correspondingly increased risk of partial failures. Most practitioners recommend the use of coarse-grained, message-oriented Web Service that minimize the number of messages and avoid the need to maintain state information between invocations. In this paper we argue that excessive use of coarse-grained, document-centric message structures results in poor reusability and high levels of data coupling between services. We describe a design approach that provides a framework for making decisions about service granularity based on data properties of interface parameters. We treat the problem of service interface design from a data engineering perspective and use data normalization to reason about service granularity. We consider the impact of increasing service granularity on cohesion and coupling of services and discuss the associated design tradeoffs.

1 Introduction

Web Services are used mainly in inter-enterprise applications that rely on the Internet as the underlying transport mechanism, and consequently are subject to a different set of design considerations than intra-enterprise applications. Internet-scale distributed applications must be able to deal with unreliable connections, provider site failures, network latency, and trust issues. Most practitioners recommend the use of coarse-grained (i.e. services with aggregate message payloads), message-oriented Web Services on the basis that coarse-grained Web Services generate fewer (SOAP) messages and therefore have lower communication overheads and less opportunity for failure [1]. Another motivation for using services with aggregated message payloads is to avoid the need to maintain state information between individual requests. Proponents of the REST (Representational State Transfer) model [2] go even further arguing that Internet-scale distributed applications should rely entirely on Internet infrastructure and use stateless HTTP-based communications with coarse-grained XML message payloads. Given the slow response times, high latencies, and poor reliability of the present Internet environment, performance is clearly an important consideration.

However the design of Web Services should not be driven by performance-related objectives alone. The choice between fine-grained and coarse-grained services is a choice between two extreme design options, each with different impact on performance as well as software engineering properties of service-oriented applications.

Service granularity (i.e. level of aggregation) defines the scope of functionality that a given service (or service operation) implements, and is a key determinant of reusability and maintainability of service-oriented applications. While coarse-grained services achieve performance advantages by reducing the number of network interactions required to implement a given business function, when considered from the perspective of software engineering, they suffer from a number of significant drawbacks, including limited reuse and poor maintainability. Coarse-grained Web Services are characterized by complex message structures that arise from designing message payloads to include all the information needed to perform the corresponding business function (e.g. airline booking) avoiding the need to maintain state information between invocations. Complexity of messages is further increased by embedding business rules and constraints within the message data structures [3]. To illustrate this point consider, for example, the OTA (OpenTravel Alliance, www.opentravel.org/) flight booking business process. OTA defines request/response message pairs for individual business processes, and implements the flight booking business process using the OTA_AirBookRQ/OTA_AirBookRS message pair. The flight booking request document OTA_AirBookRQ (www.opentravel.org/downloads/2002B_XSD/OTA_AirBookRQ.xsd) is a complex, aggregate document that contains a large number of data elements (many optional) including flight booking, itinerary, traveler and payment details. The underlying assumption is that all of the information is available at the time of booking the flight, and that the airline tickets are paid for when the flight is booked. In reality, however flight booking is a *conversational* business process with payment often performed separately from the booking. The complexity and redundancy of message data structures makes it difficult to evolve the specification without producing undesirable side-effects that invalidate existing applications. Message payloads that externalize complex data structures result in high levels of data coupling and interdependencies between services, violating a fundamental design objective for distributed applications (i.e. minimization of coupling). Decomposing the flight booking request into separate, lower-granularity operations (e.g. flight enquiry, flight booking, payment, etc.) leads to simplification of the interface, improved flexibility, and potential for reuse (i.e. payment operation can be reused in another context, e.g. car rental, or a hotel room booking).

From a software engineering perspective, services need to be designed to maximize cohesion and minimize coupling [4], [5]. Maximization of cohesion refers to the requirement for methods to implement a single conceptual task and is closely related to reusability and maintainability of application components. High level of cohesion produces orthogonal services and improves the stability of the application as modifications can be typically confined to a specific service, or service operation. Minimization of coupling (i.e. interdependencies between services), results in improved ability to accommodate change. Applying these principles to service design leads to improved clarity of the interfaces, reduction in undesirable side effects, and improved

flexibility of applications [6], [7]. Such requirements tend to favor finer-granularity services, and therefore conflict with performance considerations. Balancing performance and software engineering considerations involves design tradeoffs and requires good understanding of the impact of service granularity on cohesion and coupling of service-oriented applications.

In this paper we consider the problem of designing service interfaces from a data engineering perspective, applying data normalization to reason about service granularity. In the following section (section 2) we briefly review research literature dealing with the design of Web Services. We then describe a framework for the design of services (section 3) and use this framework to illustrate how this approach can be applied to produce a set normalized service interfaces (section 4). We then evaluate the impact of increasing service granularity on cohesion and coupling (section 5). In the concluding section (section 6) we summarize the benefits of the proposed design framework and identify potential for further work.

2 RELATED WORK

In this section we review related research literature dealing with Web Services design. The approaches can be broadly classified into methodologies based on object-oriented design [8], [9], [10], [11] [12], methods for transformation of industry domain specifications [13], and business process transformation approaches [4], [14], [15].

Ambler [8] 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.

Existing industry domain specifications have been used as a starting point for Web Services design. For example, Masud [13] gives guidelines for the definition of Web Services interfaces using WSDL, and Web Services interaction flows using BPEL from existing RosettaNet standards (www.rosettanet.org). The methodology describes how relevant information can be extracted from RosettNet PIP (Partner Interface Process) 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 domain standards enables design of interfaces and interaction dialogues based on industry-wide standard business processes and vocabularies. This avoids inconsistencies arising from individual partner organizations defining their own Web Services and results in significantly improved interoperability.

Papazoglou and Yang [4] describe a design methodology that gives a set of 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. The methodology produces definition of WSDL Web Service interfaces and WSFL service flow models, and also includes non-functional service design guidelines that relate to service provisioning strategies and service policy management models.

Radeka [15] proposes a Web Services design methodology for intra-enterprise Web Services-based application development. The starting point for the method is writing scenarios to describe the user experience. Then high level business process models are developed based on these scenarios. Refining the business model involves maximizing the number process steps that are reused across individual business process diagrams and replacing process steps with corresponding services. Information flows between services are then specified, providing basis for defining service interfaces. The main objective of this methodology is design of sharable component services that can be reused in multiple application contexts.

Stevens [16] focuses on the problem of designing Web Services with appropriate level of granularity, differentiating between coarse-grained and fine-grained services based on the scope of functionality covered by the service. Granularity of service operations relates to the amount of information returned by the operation and determines a number of calls a client application makes. Fine-grained operations result in a larger number of calls increasing network traffic, while coarse-grained operations may need to transmit unnecessary information.

In summary, Web Services design is an active research area and while there is some agreement about the basic design principles there are no widely accepted design methodologies that can guide designers of Web Services applications. Most methodologies focus on identifying Web Services given a set of application requirements or industry domain standard specification. The focus of this paper is on the data properties of interface parameters and the application of data engineering principles to determining the level of service aggregation.

3 PROCEDURES vs. MESSAGES

Decisions about service granularity need to be made in the context of a design methodology based on software engineering principles. We adopt key component-based design principles of orthogonality, maximization of cohesion, and minimization of coupling and adapt these principles to service design. For this to be possible, we need to view SOA (Service-Oriented Architecture) from a programmatic perspective, not as a message-based paradigm.

From an architectural point of view service-oriented applications can be considered at different levels of abstraction. From one perspective they can be regarded as distributed systems that use message interchange as the basic communication mechanism, i.e. messages are regarded as the key artifacts of service-oriented applications. Message-oriented approaches and Message-Oriented Middleware (MOMs) have been used with varying degrees of success in the context of Enterprise Application Integration (EAI) for the implementation of loosely-coupled, asynchronous applications.

Alternatively, service-oriented applications can be viewed as programmatic environments that use procedure calls to execute local and remote procedures (RPCs). RPC-based programming environment is typically (but, not necessarily) used to implement synchronous, tightly-coupled applications. Discussions of the RPC approach in the context of Web Services often assume synchronous operation. However, syn-

chronicity and the invocation style are essentially orthogonal concepts, and there are many implementations of an asynchronous RPC mechanism [17].

We exploit this duality between messages and procedures and describe a design framework that leverages object-oriented design principles and data engineering techniques for the design of message structures for service-oriented applications. Procedures typically implement well-defined functions and use simple data parameters. However, it is possible to pass complex objects (e.g. XML documents) as procedure parameters, in effect using RPCs to interchange documents, as is the prevailing practice for Web Services applications. Given this programmatic perspective, the interface contract is the signature of the corresponding procedure call (service operation), for example:

FlightEnquiry(INPUT: OriginLoc, DestinationLoc, DepartureDate,
OUTPUT: FlightNumber)

Given the message-oriented (document-centric) perspective, message payloads (i.e. XML documents within SOAP envelopes) define the interface contract. For example, the XML schemas of the messages OTA_AirBookRQ and OTA_AirBookRS constitute the interface contract the BookFlight and specify the method signature as:

BookFlight(INPUT: OTA_Air_BookRQ, OUTPUT: OTA_Air_BookRS)

Importantly, these abstractions are independent of the (physical) implementation of Web Services application that the designer may eventually choose. So that adopting the programmatic perspective during the design stage does not imply that the implementation of services will be based on synchronous RPCs. It is, for example, possible to conduct the design using the programmatic perspective and adopt the document style, asynchronous Web Service implementation. We regard decisions about the implementation style (i.e. binding style, RPC or document) and interaction model (i.e. synchronous or asynchronous) as orthogonal to the task of designing service interfaces, and defer such decisions to later stages of the systems development process. This separation of concerns allows focus on interface design without introducing implementation dependent constraints during the early design stages. Taking the document-centric perspective makes it difficult to reason about design tradeoffs associated with different message design strategies (e.g. level of message aggregation). However, changing the level of abstraction from messaging to programmatic interactions and regarding the messages structures as service interfaces makes it possible to apply well-established program design techniques to Web Services message payloads.

Using the programmatic perspective, the task of designing interfaces for service-oriented applications is conceptually similar to design of methods for object-oriented applications. The guiding principles for interface design include orthogonality (i.e. each interface should define a distinct function), maximization of method cohesion and minimization of method coupling. Cohesion and coupling have been studied extensively in the context of structured and object-oriented programming [18]. Myers [19] defined module cohesion as a degree of interaction within programming modules and coupling as the degree of interaction between programming modules, and classi-

fied both measures according to type. According to Myers, the highest levels of cohesion are Informational (all functions within a module share the same data) and *functional cohesion* (module performs a single function). Minimal (i.e. the most desirable) types of coupling are *stamp coupling*, where modules use data structures as parameters, and *data coupling* where individual data elements are used as parameters. Thus the combination of functional cohesion and data coupling produces the most desirable situation from the point of view of reuse and maintainability. To achieve the highest level of cohesion the designer must ensure that service operations use the same data structures (i.e. informational cohesion) and that each service operation (i.e. method) implements a well-defined, atomic task (functional cohesion). Importantly, high level of method cohesion leads to orthogonality as functional overlap is minimized, or eliminated altogether. The requirement for data coupling dictates that interfaces consist of individual data parameters rather than complex data structures or object references. Furthermore, using individual data parameters for interface specification rather than coupling via complex data structures (i.e. stamp coupling) enables the application of data engineering principles to minimize interdependencies between service operations, as described in the following sections.

4 DESIGN FRAMEWORK

The definition of 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 involves identifying suitable candidate methods that are progressively refined to produce a set of well-defined service interfaces [5], [20]. The design framework consists of three design stages; the first two stages involve top-down decomposition with the objective of identifying elementary, reusable services. The final stage involves service aggregation with the aim of optimizing granularity.

i) Initial design of service interfaces

The purpose of this stage is to produce a set of candidate interfaces that will be refined during the following design stages. Complex business functions are progressively decomposed into elementary functions and then mapped to corresponding service interfaces. This approach is consistent with maximizing method cohesion as elementary business functions typically accomplish a single conceptual task and exhibit high levels of cohesion.

ii) Refining interface design using interface normalization

Given the initial set of service interfaces, we refine the design using data normalization techniques. Normalization of service interfaces eliminates redundant input and output parameters and further increases cohesion via decomposition of interface data structures based on functional dependencies between parameters. The resulting set of normalized interfaces conforms to the maximal level of cohesion (i.e. functional cohesion) and minimal level of coupling (i.e. data coupling).

iii) Finalizing design by adjusting service granularity
During the final design stage we consider combining operations to fine-tune the granularity of services interfaces. The preceding design steps produce fine-grained services that would require highly conversational implementations and state maintenance between invocations. This final design stage is concerned with service aggregation with the aim of achieving an optimal level of service granularity taking into account performance, state management and other related considerations.

4.1 Flight Booking Example

We base our design example on the OTA airline availability request/response messages: OTA_Air_AvailRQ/OTA_Air_AvailRS and booking request message pair: OTA_Air_BookRQ/ OTA_Air_BookRS. Decomposition of the Flight Booking business function can be achieved 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. Alternatively, elementary business functions can be identified as leaf functions in a business function hierarchy. Both of these approaches have been described in detail elsewhere [5], [21]; we reproduce the resulting service interfaces that correspond to the elementary business functions and use interface parameters derived from the corresponding OTA messages:

FlightEnquiry(INPUT: OriginLocation, DestinationLocation, DepartureDate, OUTPUT: FlightNumber, DepartureAirport, ArrivalAirport, DepartureTime, ArrivalDate ArrivalTime)

SeatEnquiry(INPUT: FlightNumber, DepartureAirport, ArrivalAirport, DepartureDate, CabinType, OUTPUT: Quantity)

PriceEnquiry (INPUT: FlightNumber, DepartureAirport, ArrivalAirport, DepartureDate, CabinType, OUTPUT: FareBasisCode, BaseFare)

BookFlight (INPUT: FlightNumber, DepartureAirport, DepartureDate, ArrivalAirport, CabinType, TravelerNameOUTPUT: BookingReferenceID)

SeatingRequest (INPUT: BookingReferenceID, SeatPreference, OUTPUT: SeatNumber)

MealRequest (INPUT: BookingReferenceID, MealPreference, OUTPUT: MealType)

5. Interface Normalization

Detail analysis of the above service interfaces reveals that further decomposition of services is possible. We use interface normalization to identify potential candidates for decomposition. Normalized data structures have been used extensively in database design [22], [23]; we use the same principles here in order to minimize data coupling of service interfaces. Data coupling involves two or more interfaces being coupled via

interface parameters, i.e. output parameters of one interface match input parameters of another. Removing data parameter interdependencies for the input and output parameter sets will ensure that both parameter sets are minimal (i.e. do not contain redundant parameters). We have classified service operations according to type into query (i.e. operations that return data in output parameters given a query specified using input parameters) and update operations (i.e. operations that update data given update operation specified using input parameters), and formulated interface design rules that test for mutual independence of data parameters [5]. We regard the interfaces of query operation as relations where the input parameter set corresponds to the relation key, and the output parameter set are the non-key attributes. Output parameters of normalized interfaces are fully functionally dependent on the input parameter set, i.e. the interface parameters form a BCNF (Boyce-Codd Normal Form) relation. This ensures that parameters are used as data, not as control parameters avoiding control coupling that involves using interface parameters to control the execution of the method [19]. Normalization of interfaces of query operations also ensures mutual independence of interfaces parameters for both input and output parameter sets (i.e. input and output parameter sets are minimal). Update operations, in general, do not exhibit functional dependencies between input and output parameters. However, both input and output parameters sets should be minimized by removing redundant data parameters avoiding unnecessary data coupling. Now assuming the following functional dependencies:

FD1: OriginLocation, DestinationLocation, DepartureDate → FlightNumber
FD2: FlightNumber → DepartureAirport, DepartureTime, ArrivalAirport, ArrivalTime
FD3: FlightNumber, DepartureDate → ArrivalDate
FD4: FlightNumber, DepartureDate, CabinType → Quantity
FD5: FlightNumber, DepartureDate, CabinType → BasicFareCode, BasicFare

such considerations lead to further decomposition of service interfaces and produces the interface definitions below (see [5] for detail explanation of interface normalization):

Query Operations:

FlightEnquiry(INPUT: OriginLocation, DestinationLocation, DepartureDate, OUTPUT: FlightNumber)

ScheduleEnquiry(INPUT: FlightNumber,
OUTPUT: DepartureAirport DepartureTime, ArrivalAirport, ArrivalTime)

ArrivalEnquiry(INPUT: FlightNumber, DepartureDate,
OUTPUT: ArrivalDate)

SeatEnquiry(INPUT: FlightNumber, DepartureDate, CabinType,
OUTPUT: Quantity)

PriceEnquiry(INPUT: FlightNumber, DepartureDate, CabinType,
OUTPUT: FareBasisCode, BaseFare)

Update Operations:

BookFlight(INPUT: FlightNumber, DepartureDate, CabinType, TravelerName, OUTPUT: BookingReferenceID)

SeatingRequest(INPUT: BookingReferenceID, SeatPreference,
OUTPUT: BookingReferenceID)

MealRequest(INPUT: BookingReferenceID, MealPreference,
OUTPUT: MealType)

6. Further Considerations of Service Granularity

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 and the complexity of the interaction dialogue makes this approach difficult to implement in practice given the existing Internet infrastructure. Finding an optimal level of granularity for services and individual service operations requires further examination.

We can use the normalization framework introduced in section 5 to study the impact of aggregating service operations. Consider, for example, the query operations SeatEnquiry and PriceEnquiry that share common input parameters FlightNumber, DepartureDate, CabineType. Combining the two interfaces produces a composite operation SeatPriceEnquiry:

SeatPriceEnquiry(INPUT: FlightNumber, DepartureDate, CabinType,
OUTPUT: Quantity, FareBasisCode, BaseFare)

This clearly leads to loss of cohesion as the resulting operation no longer implements a single atomic task, and in situations where it is used to perform a partial enquiry (e.g. seat availability enquiry only) the operation returns values that are not used by the application. In the context of normalization this lack of cohesion is reflected by a partial functional dependency between the input and output parameter sets of the SeatPriceEnquiry interface (i.e. the loss of *full* functional dependence). This tradeoff 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:

SeatingMealRequest(INPUT:BookingReferenceID, SeatPreference, MealPreference,
OUTPUT: SeatNumber, MealType)

This time, a partial request, e.g. seating request only, produces non-homogeneity with MealPreference and MealType left undefined.

6.1 Note Implementation Style and Interaction Model

Following decisions about the appropriate level of service granularity, the final design stage involves decisions about the implementation style (i.e. binding style, RPC or document) and interaction model (i.e. synchronous or asynchronous, stateful or stateless). The above considerations of service granularity are orthogonal to such implementation decisions, and the designer is free to choose the most suitable approach given the characteristics of a particular implementation environment. Adopting the document-centric (message-oriented) approach the resulting interface definitions are transformed into document-style WSDL specifications. Alternatively, the resulting interfaces can be mapped directly into Web Services operations using the RPC binding style [3]. Detailed discussion of such implementation issues is outside the scope of this paper.

7. Conclusions

In this paper we presented a design methodology for service-oriented applications that applies data engineering principles to the design of service interfaces. 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 data parameters.

While we have argued that excessive use of coarse-grained, document-centric message structures results in poor reuse and undesirable interdependencies between services, we do not advocate fine granularity services as a universal solutions. Equally, this paper does not represent an argument for any particular Web Services implementation style (i.e. RPC or document style), as such decisions need to be made in the context of specific application requirements and taking into account the implementation environment.

The main benefits of the described design framework is that it facilitates making informed decisions about the level of service granularity based on the analysis of data properties of interface parameters. As illustrated in section 6, composite services can be constructed from elementary services by combining operations based on the properties of interface parameters, and the impact of service aggregation on cohesion and coupling can be evaluated. Further research is needed to understand how service aggregation based on interface parameters can be used to achieve optimal service granularity given a set of application requirements. Another area of research interest concerns the application of this methodology in the more general context of services composition.

8. References

[1] Huhns, Michael N. and Munindar P. Singh, "Service-Oriented Computing: Key Concepts and Principles," IEEE Internet Computing, vol. 9, no. 1, 2005, pp. 75-81.

[2] Fielding, R.T. Architectural Styles and the Design of Network-based Software Architectures, PhD Dissertation, 2000, Available on: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
[3] Feuerlicht, G. Implementing Service Interfaces for e-Business Applications. In Proceedings of the Second Workshop on e-Business (WeB 2003), Seattle, USA, December 2003. ISSN: 1617-9846.
[4] Papazoglou, M.P. and Yang, J. (2002), Design methodology for Web services and business processes. In Proceedings of the 3rd VLDB-TES workshop (Hong Kong, August, 2002). Springer, pages 54-64.
[5] Feuerlicht, G, Designing Service-Oriented e-Business Applications using Data Engineering Techniques, The Third Workshop on e-Business, in conjunction with ICIS 2004, December 11, 2004, Washington D.C., USA, ISBN:957-01-9161-9
[6] Venners, B. (1998) Introduction to Design Techniques. Available on: http://www.javaworld.com/javaworld/jw-02-1998/jw-02-techniques.html, February, 1998.
[7] Venners, B. (2002) API Design: The Object. Available on: http://www.artima.com/apidesign/object.html, April 26, 2002.
[8] Ambler, S.W. (2002) Deriving Web Services from UML models, Part 1: Establishing the process. Available on: http://www-106.ibm.com/developerworks/webservices/library/ws-uml1/
[9] Levi, K. and A. Arsanjani (2002) A goal-driven approach to enterprise component identification and specification. Communications of the ACM. Vol. 45:(10). (2002) 45 - 52
[10] Luo, M. et al. 2005, Service-Oriented Business Transformation in the Retail Industry Part 1: Apply SOA to Integrate Package Solutions and Legacy Systems [Online]. Available: http://www.ibm.com/developerworks/webservices/library/ws-retail1/ [Accessed 15 April 2005].
[11] Meyer, B. Object-oriented Software Construction. 2nd ed. Prentice Hall, Upper Saddle River, N.J., 1997.
[12] Smith, R. Modeling in the Service Oriented Architecture, 2003. http://archive.devx.com/javasr/articles/smith1/smith1-1.asp.
[13] Masud, S. RosettaNet-based Web Services, Part 2: BPEL4WS and RosettaNet, 2003. http://www-106.ibm.com/developerworks/webservices/library/ws-rose2/.
[14] Leymann, F. Web Services Flow Language (WSFL 1.0), 2001.http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.p
[15] Radeka, K. Designing a Web Services Project for Maximum Value: the 90 Day Challenge. In Proceedings of Conference on Object Oriented Programming Systems Languages and Applications archive (OOPSLA 2002) Practitioners Reports, Seattle, Washington, November 2002. ACM Press New York, NY, USA. ISBN:1-58113-471-1.
[16] Stevens, M. Multi-Grained Services. http://www.developer.com/design/article.php/1142661, May 21, 2002.
[17] A.L. Ananda, B.H. Tay, and K.E. Koch. *A Survey of Asynchronous Remote Procedure Calls*. ACM Operating Systems Review, 26(2):92--109, July 1992.
[18] Stevens, W.P., Myers, G.J., and Constantine, L.L., Structured Design, IBM SYSTEMS JOURNAL, VOL38, NOS2&3, 1999
[19] Myers, G.J.: Composite Structured Design, 1978, Van Nostrand Reinhold, ISBN 0-442-80584-5, 175 pages
[20] Feuerlicht, G., Design of Service Interfaces for e-Business Applications using Data Normalization Techniques, Journal of Information Systems and e-Business Management, Springer-Verlag GmbH, 26 July 2005, pages 1-14, ISS:1617-98
[21] Feuerlicht, G. and S. Meesathit. Design Framework for Interoperable Service Interfaces. In Proceedings of the 2nd International Conference on Service Oriented Computing, pp. 299-307, New York, NY, USA, November 2004. ACM Press. ISBN:1-58113-871-7.

[22] Codd, E.F (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, 1971, 1-17.
[23] Date, C. J. Fagin, R. (1992) Simple Conditions for Guaranteeing Higher Normal Forms in Relational Databases, ACM Transactions on Database Systems (TODS) Volume 17, Issue 3 (September 1992) Pages: 465 - 476, 1992, SSN:0362-5915