Understanding Service Reusability

George Feuerlicht,
Josip Lozina

Faculty of Information Technology,
University of Technology,
Sydney, Australia
jiri@it.uts.edu.au, jlozina@it.uts.edu.au

Abstract: *Reusability of software components is a key determinant of application development productivity and maintainability of applications. Reuse is regarded by many organizations as the top driver for the adoption of SOA. However the mechanism for achieving reusability of services is poorly understood at present and there is evidence that design of services for reuse is not a prime objectives when implementing SOA. The design of services is driven primarily by performance and scalability considerations, rather than any sound software engineering principles. In this paper we argue that excessive use of coarse-grained, document-centric messages results in poor reuse and undesirable interdependencies between services. We discuss the relationship between granularity of services, service reuse and service composability and describe how developers of Web Services applications can get a better understanding of the tradeoffs between performance, reuse, and maintainability of service-oriented applications.*

Keywords: SOA, Web Services, Service Granularity, Service Reusability

1 Introduction

Many organizations are moving towards Service Oriented Architecture (SOA) in order to closely align IT capabilities with their business goals. SOA based on Web Services standards allows applications to exchange data and participate in business processes regardless of the underlying technological platform, and provides a more agile technical infrastructure that can rapidly adapt to new business requirements (Newcomer and Lomow 2004). Among the many perceived SOA benefits it is the expectation of greater reuse that is regarded by many organizations as the top driver for the adoption of SOA (Hurwitz 2006). Service reusability is essential for organisations to achieve good return on investment (McGovern, Tyagi, Stevens & Matthew 2003).

From a business point of view there is a direct relationship between reuse and ROI (Return on Investment); reuse reduces the costs associated with design, development and testing, as well as significantly reduces maintenance effort (Erl 2005). For example, in a recent Credit Suisse study of SOA implementation the main benefits of SOA were attributed to reuse of services that resulted in more efficient application development (Krafzig et al 2004). Another case study describes a project conducted in a life insurance organisation in the UK with the strategic objective to leverage its catalogue of reusable business services to lower the costs and increase competitive advantage. The organisation achieved service reuse of 51% which represented a development savings of over £2,000,000 without taking into account other areas of the business such as operational support and service delivery functions (Bieberstein et al 2005).

Software reuse is a well-established principle in object-oriented programming and component-based development. Existing object-oriented and component-based methodologies provide many principles and practises that can be applied to developing SOA applications. Component-based development has

demonstrated ability to build large software systems by integrating new and existing software components, promoting software reuse and reducing development time and cost. SOA embraces the basic principles of component-based design such as information hiding, modularization, separation of concerns and abstraction, but also differs from component-based design in a number of important respects. Unlike components, services are implemented at a higher-level of abstraction directly supporting business processes and functional units (Zimmermann et al. 2004). SOA is regarded as the next evolutionary step extending object-oriented and component-based methods to facilitate the development of loosely-coupled inter-enterprise applications, providing a mechanism for composing complex applications that consist of highly autonomous program modules (services) (Chappell & Jewell 2003). Services interact via service interfaces that define the contract between a service provider and a consumer and determine the scope of functionality implemented by the service, i.e. service granularity. Because of the emphasis on high-level of abstraction, most SOA practitioners recommend the use of coarse-grained services that typically encapsulate entire business processes (e.g. air travel booking, course enrolment, etc.). Coarse-grained services avoid the need to maintain state information between service invocations, and reduce the number of network interactions required to implement a given business function, improving performance and simplifying recovery in the case of failure. However, there is an inverse relationship between service granularity and service reusability; as the scope of functionality implemented by a given service increases, the potential for its reuse diminishes. Also importantly, excessive service granularity impacts on the ability to evolve service-oriented applications without producing undesirable side-effects. Coarse-grained services externalize complex data structures and exhibit high levels of data coupling (Feuerlicht 2005). Consequently, one of the main challenges for the developers of service-oriented applications is to determine the most appropriate level of service granularity to ensure that services are reusable, exhibit a high degree of mutual independence, and at the same time have good performance characteristics. Decisions about service granularity play a key role in the design of services, and need to be considered in the context of a methodological framework, rather than based purely on heuristics. While most experts agree that service granularity impacts on service reuse, the exact mechanism for reuse of autonomous services in highly distributed (domain-wide) service-oriented applications is poorly understood at present (Feuerlicht 2006).

In this paper we first discuss service design principles (section 2), and then focus on service granularity (section 3) and analyse the relationship between service granularity and service reuse (section 4). We conclude by noting that further research is needed to develop a comprehensive methodological framework for making design decisions about service granularity so that reuse can be maximized (section 5).

2 Service Design

Reusability is not an automatic consequence of implementing SOA or using Web Services. Similar to reusability of objects and components, reusability in the context of services is a matter of good design. More specifically, it is the design of service interfaces that determines service reusability. Of equal importance is the stability of the resulting service interfaces so that the impact of changing requirements is minimized (i.e. confined to small number of services, ideally a single service). The task of designing service interfaces is conceptually similar to the design of method signatures in object-oriented applications. The key guiding principles for interface design include orthogonality, maximization of method cohesion and minimization of method coupling. The orthogonality requirement is closely related to service cohesion and states that each service (or service operation) should implement a distinct function (i.e. the functionality of services should not overlap). The principles of maximization of cohesion and minimization of coupling have been studied extensively in the context of structured and object-oriented programming and have been applied to service design with some minor refinements (Feuerlicht, 2005a).

2.1 Service Coupling

Coupling is a measure of the *strength* of the relationship between two or more services; minimization of coupling reduces interdependencies between services facilitating service evolution (Larman 2004). The concept of loose coupling is one of the most important SOA design principles, but there is a considerable confusion about its interpretation. For example, Kaye (Kaye 2003) differentiates between tight and loose coupling in the context of Web Services using a set of characteristics listed in Table 1 below.

	Tightly Coupled	**Loosely coupled**
Interaction	Synchronous	Asynchronous
Messaging Style	RPC	Document
Messaging Paths	Hard Coded	Routed
Technology Mix	Homogenous	Heterogeneous
Data Types	Dependent	Independent
Syntactic Definition	By Convention	Published Schema
Bindings	Fixed and Early	Delayed
Semantic Adaptation	By Re-coding	Via Transformation
Software Objective	Reuse, Efficiency	Broad Applicability
Consequences	Anticipated	Unexpected

Table 1: *Loose versus tight coupling (Kaye 2003 p.132)*

The table includes a wide range of characteristics, and in some cases the interpretation of coupling is too simplistic. For example, it can be argued that the use of the document-centric approach (i.e. document messaging style) leads to increased data coupling as the service interface typically contains complex data structures. Conversely, using RPC messaging style does not necessarily result in tight coupling as RPCs can be deployed asynchronously and the service operations can be designed to minimize data coupling. It is primarily the (data) coupling via interface parameters that determines both the stability of the interface and its reuse potential and needs to be minimized when designing services (Feuerlicht, 2006). As the service interface constitutes a contract it cannot be changed once externalized, and should be the subject of careful design considerations. Once the service interface is designed, other issues (e.g. interaction and messaging style) can be considered separately during later design stages, or even managed during the deployment of applications.

2.2 Service Cohesion

Cohesion refers to the level of interrelationships between the elements of a software module (Stevens 1999). High level of service cohesion increases application stability as cohesion limits the impact of changes to a small number of services (ideally, a single service operation). According to Larman (Larman 2004) the benefits of highly cohesive methods include easy maintenance (as the method interfaces are less likely to change over time) and increased reuse potential. Functional cohesion is achieved in situations where each service operation implements a single clearly-defined task; this leads to low coupling and high levels of reuse (Vinoski 2005). Maximizing functional cohesion leads to modular design; this well-known principle has been exploited in object-oriented design that promotes modularity via decomposition of systems into a set of cohesive and loosely coupled modules (Booch 1993). Applying this principle to the design of services results in simple, mutually independent

interfaces with good reuse potential. There is a close relationship between service cohesion and coupling as poor cohesion leads to high levels of coupling. Furthermore, poor service cohesion leads to overlapping service functionality, i.e. loss of orthogonality.

2.3 Service Granularity

In follows from the above discussion that one of the most important aspects of designing service-oriented applications relates to decisions about service granularity. In practice, such decisions are often based on heuristics and *rules of thumb*. For example, Schmelzer (Schmelzer 2006) defines granularity as "a relative measure of how broad a required piece of functionality must be in order to address the need at hand". Service granularity is often measured in terms of the amount of data in the message payload of a service; thus coarse-grained services typically have aggregated message payloads resulting from executing multiple operations before returning the data (McGovern, Tyagi, Stevens & Matthew 2003). Fine-grained services, on the other hand typically implement individual operations directly, resulting in excessive *chattiness* and complex interaction dialogues (Erl 2005).

In general, SOA best practices advocate the use of stateless, asynchronous, and coarse-grained Web Services (Sperberg-McQueen 2003). Stateless asynchronous services are significantly more scalable and result in improved performance. Implementation of coarse-grained services can be achieved using a façade pattern over fine-grained service operations (Keen M. 2004). Using this design pattern the client sends all the required information in a single call, avoiding invocation of separate methods (Singh et al 2004). This *Coarse-Grained Remote Interface* design pattern has been used extensively in distributed computing as a technique for reducing communication overheads (Larman 2004). Coarse-grained services typically correspond to individual business functions identified by top-down analysis, or are the result of bottom-up service aggregation. XML provides a natural mechanism for transmitting coarse-grained messages using the Web Services SOAP protocol (Bellur & Narendra 2006), (McKusick 2003).

Fine-grained services typically correspond to elementary business functions and implement highly reusable business logic (Bieberstein et al 2005). The benefits of fine-grained services include improved cohesion, reduction in coupling and better clarity of the design (Buchmann 2002). It has been demonstrated that data properties of service interfaces determine the level of service cohesion and that data normalization techniques can be used to maximize cohesion and minimize coupling of services (Feuerlicht 2005).

A recent quantitative comparison of fine-grained and coarse-grained services confirmed that fine-grained services exhibit lower levels of coupling resulting in easier propagation of changes and more flexible and maintainable services (Perepletchikov et al, 2005). The study compared coarse-grained services with business logic embedded into a hierarchical object-oriented design structure with a fine-grained BPEL implementation based on embedding business rules and logic into executable BPEL scripts.

The level of abstraction of coarse-grained services corresponds to business processes while fine-grained services tend to operate at a more technical level. The recursive nature of service composition allows the construction of business level services from reusable fine-grained services. Business-level services can then be made available to external client applications (i.e. external to the enterprise or administrative domain), while fine-grained services are used within the enterprise taking advantage of greater flexibility and reuse (Umar 2004), (Warner 2006). BPEL (Business Process Execution Language) is regarded as the standard service composition language and can be used to implement high level business processes by assembling and composing individual Web Services. BPEL forms a layer on top of services and provides a mechanism for the implementation of business logic, state maintenance, and failure recovery. Services can be designed for reuse in various BPEL composition scenarios, and the explicit implementation of business logic (i.e. business process workflows) using BPEL makes maintenance easier (Erl 2005).

3 Service Reusability

Software reuse has been the subject of extensive investigation in the context of object-oriented programming and component-based development. The primary mechanism for achieving reuse in object-oriented programming is (class) inheritance. Inheritance creates strong dependencies (i.e. coupling) among application objects and for that reason is generally not used in distributed environments. The mechanism for maximizing reuse of service components in the context of highly distributed (domain-wide) service-oriented applications with services developed by autonomous services providers is not fully understood at present. We define service reuse as the ability to participate in multiple service assemblies (compositions). Given this perspective, services must be *composable* to achieve good levels of reuse, i.e. service reuse is closely related to service composability. For example, a well-designed credit card verification service can be reused in a large number of payment applications, and therefore is highly reusable. It follows from the above discussion that the bias towards coarse-grained (i.e. aggregated), message-oriented services, favoured by most SOA practitioners makes achieving reuse difficult in practice as such services are not readily composable.

3.1 Identifying Reusable Services

In this section we briefly consider the problem of design of reusable services. Only a brief overview of the method is presented here; the design method has been described in detail in previous publications (Feuerlicht, 2005; Feuerlicht, 2005a). The design method consists of three main design stages: the first stage involves top-down decomposition with the objective of identifying elementary, reusable service components (i.e. service operations). The second stage involves service aggregation with the aim of optimizing service granularity with respect to the requirements of a particular message interchange scenario (e.g. airline travel booking dialogue). This stage could also include consideration of performance, state management and other related issues. The final design stage involves mapping the resulting service operations to a BPEL process that implements a specific (high-level) business function.

During the decomposition stage complex business functions are progressively decomposed into elementary functions and then mapped to corresponding candidate service operations. This is consistent with maximizing cohesion as elementary business functions typically accomplish a single conceptual task and exhibit high levels of cohesion. Decomposition can be achieved by modeling the interaction between services using a Sequence Diagram. Each step in the Sequence Diagram dialog produces a request/response message pair that corresponds to an elementary business function. Alternatively, elementary business functions can be identified as leaf functions in a business function hierarchy. Given the initial set of candidate service operations, further decomposition can be achieved by applying data normalization to the interface data parameters (Feuerlicht, 2005a). However, this produces a large number of service operations and results in an excessively complex interaction dialogue characterized by a large number of runtime calls. While the resulting services are highly reusable this low-granularity solution is not practical. Finding an optimal level of service granularity requires further examination, identifying operations that are suitable candidates for aggregations. We have shown that service aggregation can be performed by applying relational operations over service interface parameters. Finally, the resulting (aggregated) services are mapped to a BPEL process and externalized as a high-level service.

4 Conclusion

We have noted that reuse is one of the main reasons for organizations to adopt SOA, however so far the perception of improved reuse can be mainly attributed to the ability to derive business value from legacy applications by externalizing existing functionality as Web Services. While reusing

functionally *locked* in legacy applications is clearly important, it is the reusability of services in newly developed applications that will ultimately determine the long-term business benefits derived from SOA. Some argue that the costs involved in identifying, understanding and administering reusable services may outweigh the benefits of gained by reuse (Krafzig et al 2004). Others maintain that promoting reuse potential is important even if it there is not an immediate opportunity for reuse (Erl 2005).

Most SOA practitioners advocate coarse-grained services mainly on the basis of performance considerations, but also because SOA is regarded as a message-oriented paradigm. This results in poorly designed services that suffer from limited reuse. Applying proven software design principles will ensure that organisations will achieve higher levels of service reuse and better return on investment in the long term.

5 References

Bellur, U. & Narendra, N., 2006 'Towards a Programming Model and Middleware Architecture for Self-configuring systems' First International Conference on Communication System Software and Middleware, 2006. Comsware 2006. 08-12 Jan. 2006 Page(s):1 - 6

Bieberstein, N., Bose, S., Fiammante, M., Jones, K., Shah R., 2005 Service-Oriented Architecture Compass: Business Value, Planning, and Enterprise Roadmap, IBM Press

Booch, G., 1993 Object-Oriented Analysis and Design with Applications, Benjamin-Cummings Publishing Co.

Buchmann, F. & Buchmann, A., 2002 'Technologies for E- Services' Third International Workshop, Tes 2002, Hong Kong, China

Feuerlicht, G., 2005, 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

Feuerlicht, G., 2005a Design of service interfaces for e-business applications using data normalization techniques, Information Systems and E-Business Management, Volume 3, Number 4, December, 2005, pages 363-376

Feuerlicht G., Service Granularity Considerations Based on Data Properties of Interface Parameters, International Journal of Computer Systems Science & Engineering, Vol 21 No 4, July 2006, Special issue: Engineering Design and Composition of Service-Oriented Applications, ISSN 0267 6192, pp 315-327.

Hurwitz J, Bloor R, Baroudi C, Thinking from Reuse - SOA for Renewable Business, 2006, Available Online: http://www.hurwitz.com/PDFs/IBMThinkingfromReuse.pdf

Kaye, D., 2003 Loosely Coupled: The Missing Pieces of Web services. Rds Associates

Keen, M., Bishop, S., Hopkins, A., Milinski, S., Nott, C., Robinson, R., Adams, J., Verschueren, P., Acharya. A., 2004 Patterns: Implementing an SOA using an ESB, IBM Redbook

Krafzig, D. 2004, Enterprise SOA: Service Oriented-Architecture Best Practices, Dirk Krafzig, Karl Banke, Dirk Slama, Prentice Hall PTR, The Coad Series, ISBN: 0131465759; 2004

Larman, C., 2004 Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, Third Edition. Prentice Hall PTR;

McKusick, K., 2003 A Conversation with Adam Bosworth, ACM Queue vol. 1, no. 1 - March 2003 Retrieved November 12, 2006, from http://www.acmqueue.org/modules.php?name=Content&pa=showpage&pid=29

McGovern, J., Tyagi, S., Stevens, M., Matthew, S., 2003 Java Web services Architecture. Morgan Kaufmann

Myers, G.J.: Composite Structured Design, 1978, Van Nostrand Reinhold, ISBN 0-442-80584-5

Newcomer, E., Lomow, G., 2004 Understanding SOA with Web services. Addison-Wesley Professional

Perepletchikov, M., Ryan, C., Frampton, K., 2005 'Comparing the Impact of Service-Oriented and Object-Oriented Paradigms on the Structural Properties of Software',Proc. of 2nd International Workshop on Modeling Inter-Organizational Systems (MIOS), in conjunction with the OTM 2005, October 2005

Schmelzer, R., 2006 Solving the service granularity challenge. Retrieved November 1, 2006, from http://searchwebservices.techtarget.com/tip/1,289483,sid26_gci1172330,00.html

Singh, I., et al, 2004 Designing Web services with the J2EE™ 1.4 Platform JAX-RPC, SOAP, and XML Technologies Addison-Wesley Professional

Stevens, W.P., Myers, G.J., and Constantine, L.L., Structured Design, IBM SYSTEMS JOURNAL, VOL38, NOS2&3, 1999

Warner, C., 2006 'Composite Apps ', SOA Web services Journal May 2006 Volume 6 Issue 5 Sperberg-McQueen, C., 2003 Web services and the W3C. Retrieved November 10, 2006, from http://www.w3.org/2003/Talks/0818-msm-ws/Overview.html

Umar, A., 2004 E-Business and Distributed Systems Handbook: Architecture Module Nge Solutions

Vinoski, S., 2005 Old measures for new services, Internet Computing, IEEE Volume 9, Issue 6, Nov.-Dec. 2005 Page(s):72 – 74

Zimmermann, O., Krogdahl, P., Gee, C., 2004 Elements of Service-Oriented Analysis and Design Retrieved November 1, 2006, from http://www-128.ibm.com/developerworks/webservices/library/ws-soad1/