

Regis University

ePublications at Regis University

Regis University Student Publications
(comprehensive collection)

Regis University Student Publications

Spring 2008

The Value Proposition of Service-Oriented Architecture

David Norman
Regis University

Follow this and additional works at: <https://epublications.regis.edu/theses>



Part of the [Computer Sciences Commons](#)

Recommended Citation

Norman, David, "The Value Proposition of Service-Oriented Architecture" (2008). *Regis University Student Publications (comprehensive collection)*. 99.
<https://epublications.regis.edu/theses/99>

This Thesis - Open Access is brought to you for free and open access by the Regis University Student Publications at ePublications at Regis University. It has been accepted for inclusion in Regis University Student Publications (comprehensive collection) by an authorized administrator of ePublications at Regis University. For more information, please contact epublications@regis.edu.

Regis University
College for Professional Studies Graduate Programs
Final Project/Thesis

Disclaimer

Use of the materials available in the Regis University Thesis Collection ("Collection") is limited and restricted to those users who agree to comply with the following terms of use. Regis University reserves the right to deny access to the Collection to any person who violates these terms of use or who seeks to or does alter, avoid or supersede the functional conditions, restrictions and limitations of the Collection.

The site may be used only for lawful purposes. The user is solely responsible for knowing and adhering to any and all applicable laws, rules, and regulations relating or pertaining to use of the Collection.

All content in this Collection is owned by and subject to the exclusive control of Regis University and the authors of the materials. It is available only for research purposes and may not be used in violation of copyright laws or for unlawful purposes. The materials may not be downloaded in whole or in part without permission of the copyright holder or as otherwise authorized in the "fair use" standards of the U.S. copyright laws and regulations.

Regis University

School for Professional Studies Graduate Programs

MSc in Computer Information Technology Program

Graduate Programs Final Project/Thesis

Certification of Authorship of Professional Project Work

Print Student's Name David Norman

Telephone 808-268-4381 Email davidlnorman@hotmail.com


Date of Submission 3-Mar-2008 Degree Program MSc in Computer Information Technology

Title of Submission The Value Proposition of Service-Oriented Architecture

Advisor/Faculty Name Mike Nimms / Donald J. Ina

Certification of Authorship:

I hereby certify that I am the author of this document and that any assistance I received in its preparation is fully acknowledged and disclosed in the document. I have also cited all sources from which I obtained data, ideas or words that are copied directly or paraphrased in the document. Sources are properly credited according to accepted standards for professional publications. I also certify that this paper was prepared by me for the purpose of partial fulfillment of requirements for the Master of Science in Computer Information Technology Degree Program.



Student Signature

3-March-2008

Date

**Regis University
School for Professional Studies
MSCIS Program**

Advisor/MSC 696 and 696B Faculty Approval Form

Student's Name: David Norman

Professional Project Title: The Value Proposition of Service-Oriented Architecture

Advisor's Declaration: I have advised this student through the Professional Project Process and approve of the final document as acceptable to be submitted as fulfillment of partial completion of requirements for the MSC 696 or MSC 696B course. The student has received project approval from the Advisory Board or the 696A faculty and has followed due process in the completion of the project and subsequent documentation.

ADVISOR

Mike Nims



March, 02, 2008

Name

Signature

Date

MSC 696 or MSC 696B Faculty Approval

Donald J. Ina



February 28, 2008

Name

Signature

Date

Regis University
School for Professional Studies Graduate Programs
MSc in Computer Information Technology Program
Graduate Programs Final Project/Thesis
Authorization to Publish Student Work

I, David Norman, the undersigned student, in the Master of Science in Computer Information Technology Degree Program hereby authorize Regis University to publish through a Regis University owned and maintained web server, the document described below ("Work"). I acknowledge and understand that the Work will be freely available to all users of the World Wide Web under the condition that it can only be used for legitimate, non-commercial academic research and study. I understand that this restriction on use will be contained in a header note on the Regis University web site but will not be otherwise policed or enforced. I understand and acknowledge that under the Family Educational Rights and Privacy Act I have no obligation to release the Work to any party for any purpose. I am authorizing the release of the Work as a voluntary act without any coercion or restraint. On behalf of myself, my heirs, personal representatives and beneficiaries, I do hereby release Regis University, its officers, employees and agents from any claims, causes, causes of action, law suits, claims for injury, defamation, or other damage to me or my family arising out of or resulting from good faith compliance with the provisions of this authorization. This authorization shall be valid and in force until rescinded in writing.

Print Title of Document(s) to be published: The Value Proposition of Service-Oriented Architecture



Student Signature

3-March-2008

Date

Check appropriate statement:

X The Work does not contain private or proprietary information.

_____ The Work contains private or proprietary information of the following parties and their attached permission is required as well: _____

Name of Organization and/or Authorized Personnel

Regis University

The Value Proposition of Service-Oriented Architecture

by
David Norman
davidlnorman@hotmail.com

A Project Report submitted in partial fulfillment of the requirements for
the degree of Master of Science in Computer Information Systems

February 2008

**Regis University
School for Professional Studies
MSCIS Program**

Certification of Authorship of Professional Project Work

Submitted to: Mike Nims, Don Ina

Student's Name: David Norman

Date of Submission:

Title of Submission: Research of Service-Oriented Architecture

Certification of Authorship: I hereby certify that I am the author of this document and that any assistance I received in its preparation is fully acknowledged and disclosed in the document. I have also cited all sources from which I obtained data, ideas, or words that are copied directly or paraphrased in the document. Sources are properly credited according to accepted standards for professional publications. I also certify that this paper was prepared by me for the purpose of partial fulfillment of requirements for the MSC 696 or the MSC 696B course.

Student's Signature:

David Norman

A handwritten signature in black ink, appearing to read "David Norman", written in a cursive style.

March 2, 2008

**Regis University
School for Professional Studies
MSCIS Program**

Advisor/MSC 696 and 696B Faculty Approval Form

Student's Name: David Norman

Professional Project Title: The Value Proposition of Service-Oriented Architecture

Advisor's Declaration: I have advised this student through the Professional Project Process and approve of the final document as acceptable to be submitted as fulfillment of partial completion of requirements for the MSC 696 or MSC 696B course. The student has received project approval from the Advisory Board or the 696A faculty and has followed due process in the completion of the project and subsequent documentation.

ADVISOR

Mike Nims



March, 02, 2008

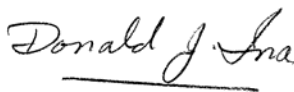
Name

Signature

Date

MSC 696 or MSC 696B Faculty Approval

Donald J. Ina



February 28, 2008

Name

Signature

Date

**An Abstract of a Project/Practicum Report Submitted to Regis University
School for Professional Studies in Partial Fulfillment of the Requirements
for the Degree of Master of Science in Computer Information Systems**

Abstract

by

David Norman

February 2008

The author of this thesis evaluates Service-Oriented Architecture (SOA) design and implementation strategies. The purpose is to provide the reader with the definition of Service-Oriented Architecture. This report discusses: (1) The definition of Service-Oriented Architecture, (2) The problems solved by Service-Oriented Architecture, (3) Application of design principles to achieve Service-Oriented Architecture. As a result of this investigation, Service-Oriented Architecture is a design style that is fundamentally about sharing and reuse of functionality across diverse applications, so that organizations can quickly adapt to changing business requirements while increasing IT asset reuse and minimizing integration and development costs.

Table of Contents

1	<i>Chapter One: Introduction - Thesis Statement</i>	1
1.1	Statement of the Problem	1
1.1.1	Inflexible Systems	3
1.1.2	Limited or No Reusability	3
1.1.3	Lack of Interoperability	4
1.1.4	Poor Maintainability	5
1.2	Service-Oriented Architecture (SOA) Defined	6
1.3	Business Need for SOA	7
1.4	Goals of SOA	7
1.4.1	IT Flexibility	8
1.4.2	Reusability	9
1.4.3	Integration and Collaboration	9
1.4.4	Maintainability	10
1.5	Barriers and/or Issues	11
1.6	Summary	12
2	<i>Chapter Two: Review of Literature and Research</i>	14
2.1	What is Architecture	14
2.2	Application Architecture	15
2.3	Enterprise Architecture	16
2.4	Service-Oriented Architecture	17
2.5	Comparison of Client Sever and Service-Oriented Architecture	18
2.5.1	Location of Application Code	19
2.5.2	Presentation and Logic Separation	20
2.5.3	Software Distribution	21
2.6	Comparison of Distributed and Service-Oriented Architecture	22
2.6.1	Location of Application Code	23
2.6.2	Communication Protocol	24
2.7	Summary	25
3	<i>Chapter Three: The Value Proposition of SOA</i>	26
3.1	Analysis of Business Benefits	26
3.1.1	Reduced Integration Expense	27
3.1.2	Increased Asset Reuse	28
3.1.2.1	Governance	28
3.1.2.2	Granular Level Design	29
3.1.2.3	Standard Interface/Web Services	29
3.1.3	Competitive Advantage	30
3.2	Analysis of Financial Benefits	31
3.2.1	Financial Metrics	32
3.2.2	Consideration of Soft Benefits	34

3.3	Risks of Investing in SOA	35
3.3.1	Weakest Link in the Service Chain	35
3.3.2	Nature of the Organization	36
3.3.3	Web Service Framework Immaturity	37
3.3.4	Inexperience with SOA Design Principles	38
3.4	Cost of not Adopting SOA	38
3.5	Summary	39
4	<i>Chapter Four: The Components of SOA</i>	40
4.1	Implementation Strategy	40
4.2	Enterprise Service Bus (ESB)	41
4.2.1	ESB Defined	41
4.2.2	Interoperability	42
4.2.3	Implementation Scenario	42
4.3	Business Process Management (BPM)	43
4.4	Business Process Execution Language (BPEL)	46
4.5	Web Services	48
4.6	Application Server	49
4.7	Universal Description, Discovery and Integration (UDDI)	51
4.8	Summary	53
5	<i>Chapter Five: Analysis and Design of SOA</i>	54
5.1	SOA Analysis and Design Overview	54
5.1.1	Task-centric Services	55
5.1.2	Entity-centric Services	56
5.2	Design Fundamentals	57
5.2.1	Service Granularity	58
5.2.2	Service Contracts	59
5.2.2.1	Standardization of Contracts	60
5.2.3	Loose Coupling	61
5.2.4	Abstraction	63
5.2.5	Reusability	63
5.2.6	Autonomy	64
5.3	Summary	65
6	<i>References</i>	67
7	<i>Appendix</i>	76

List of Figures

<i>Figure 1 – Silo-Based IT Systems (Brown, 2007)</i>	2
<i>Figure 2 – Core BPM Infrastructure Components (Howard, 2007 p. 17)</i>	44
<i>Figure 3 – Business Process Realized as Services (Jussuttis, 2007)</i>	47
<i>Figure 4 – Registry Components (Manes, January 2007)</i>	52
<i>Figure 5 – Contract Naming Standard (Erl, 2007, p. 133)</i>	60

List of Tables

<i>Table 1: Sample Total Cost of Ownership (Haddad 2005, p. 21)</i>	32
<i>Table 2: Sample Net Cash Flow (Haddad 2005, p. 22)</i>	32
<i>Table 3: Sample Discounted Cash Flow (Haddad 2005, p. 22)</i>	33
<i>Table 4: Sample Net Present Value (Haddad 2005, p. 23)</i>	33
<i>Table 5: Sample Return on Investment (Haddad 2005, p. 23)</i>	34

1 Chapter One: Introduction - Thesis Statement

According to Anne Manes, “organizations have hundreds (sometimes thousands) of legacy applications with an abundance of duplicate functionality” (Manes March 2007). To remain competitive, businesses must continually evolve to meet the changing demands of their customers. As business needs change, Information Technology (IT) needs also change. Paul Patrick states that “the necessary information is not all stored in a single data store, such as a database, but is instead stored in individual silos from which each application must drink” (Patrick, 2005). Each of these silos has its own characteristics that usually differ from other systems in the enterprise. Methods to integrate these silos have long been the focus of software architects.

1.1 *Statement of the Problem*

Service-Oriented Architecture (SOA) has surfaced as a software development method to address the problems that have typically infiltrated IT. Erl (2007) describes the historical approach to IT systems as follows:

Over the course of IT's history, the majority of such solutions have been created with a common approach of identifying the business tasks to be automated, defining their business requirements, and then building the corresponding solution logic (p. 76).

Brown (2007) reinforces this by explaining that “the majority of IT projects have traditionally focused on a single business process activity (or group of activities) residing entirely within a single application silo”. Achieving additional

value from this type of application is “usually inhibited because their capabilities are tied to specific business requirements and processes” (Erl, 2007, p.77).

Figure 1 illustrates isolated nature of silo-based applications that make up most IT systems.

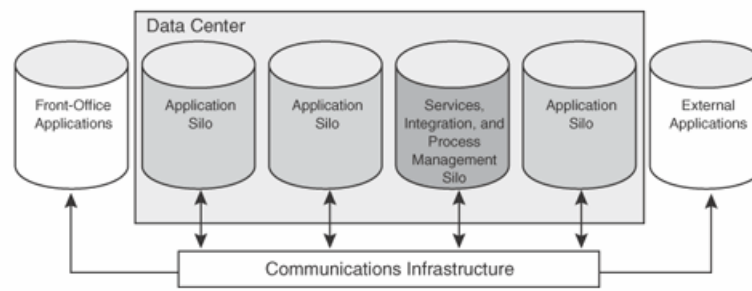


Figure 1 – Silo-Based IT Systems (Brown, 2007)

When new requirements and processes are introduced, organizations are forced to either make significant changes or build new applications altogether (Erl, 2007, p. 77). As a result, the “traditional application landscape of multiple stovepipes was extremely complex and expensive to maintain or extend. So when the business needed a change in systems, IT seemed slow to deliver it” (Mehul, 2007).

The silo-based nature of IT systems is the heart of the problem for organizations and SOA offers the possibility of eliminating these silos (Sippl, 2005, p. 6). There are 4 major recurring issues that stem from silo-based systems. They include inflexibility, lack of reusability, poor interoperability and poor maintainability (Erl, 2007). According to Manes, SOA addresses each of these issues (Manes, July 2006, p. 6). These issues are discussed in the following sections.

1.1.1 Inflexible Systems

Silo-based systems are inflexible because their capabilities are tied to specific requirements and processes (Erl, 2007, p77). Van der Vlist (2002) expands on this by explaining that these types of systems “make extra work when an application has to adapt to changing business requirements, because each modification to one application may force developers to make changes in other connected applications”. Organizations that have followed the historical norm in IT development may find the work load to adjust the system to be much more difficult than expected.

Inflexible systems corner organizations into a constant state of catch-up to adapt their IT systems to evolving business needs (Erl, 2005). This could make IT systems the bottleneck for the business. “When business processes cannot efficiently evolve, enterprises find themselves hamstrung as they try to respond to changing opportunities and pressures” (Brown, 2007).

1.1.2 Limited or No Reusability

The concept of reuse is a very simple idea: make a software component that is useful for more than one purpose (Erl, 2005). The reuse of existing IT assets can be a complicated and problematic ambition to achieve in a silo-based system (Mehul, 2007). As established previously, the majority of IT systems have multiple applications that are isolated from each other. This “lack of centralized control and communication between business units within organizations caused the same solutions to be reinvented over and over again”

(Mehul, 2007). Solutions that are continually reinvented across business units ultimately results in disposable applications (Mehul, 2007).

Disposable IT assets are obviously less than ideal because “something that is useful for a single purpose will provide value, something that is repeatedly useful will provide repeated value and is therefore a more attractive investment” (Erl, 2007, p. 254). In silo-based systems, organizations suffer financial consequences through the inefficiency of implementing existing functionality time and again (Brown, 2007).

1.1.3 Lack of Interoperability

In a silo-based system, “sharing information among applications is difficult due to differences in technology platforms and data models” (Newcomer, 2004). Applications can not collaborate across silo’s with each other to exchange data in a business process without “resource-intensive manual processes for tasks like loading data from different data sources, transforming data into the right common format, and checking the accuracy of these data” (Mehul, 2007). This has the circular effect of making the system inflexible and reducing the organizations ability to respond to changes.

The challenge that organizations face is coordinating the work of multiple application silos in order to achieve enterprise goals (Brown, 2007). Business and technology teams that remain in silos will not have a common view of the world and therefore may be working counter to the goals of the organization (Manes, 2007, p. 21). Integration of silo-based systems is a complex endeavor

because it “is a multifaceted problem, many different technologies, products, and processes have been used over the years to address it” (Newcomer, 2004).

Not only do organizations face the challenge of integrating internal systems, but there is also a growing need to integrate with business partners from external organizations (Erl, 2007). The complexity and resource-intensive processes needed to integrate silo-based applications could leave organizations in a position where they find it difficult to establish partnerships and cannot respond to market changes quickly (Manes, 2005, p.20).

1.1.4 Poor Maintainability

In a silo-based system it is common for IT groups to be burdened with maintaining many different types of applications and data sources, often written in differing languages or platforms (Brown, 2007). “This traditional application landscape of multiple stovepipes was extremely complex and expensive to maintain or extend” (Mehul, 2007). The expertise to maintain applications of varying languages and platforms can easily get out of hand. Adding to the issues is the fact that duplication in data and functionality is quite common in organizations that have many software applications and data sources (Mehul, 2007). Redundancy in functionality and data often leads to confusion and errors for those who carry out the daily business tasks because they may receive conflicting results from silo’s that are out of sync with each other (Mehul, 2007). Maintenance of the numerous disparate systems could be very complex and labor intensive (Erl, 2007).

1.2 Service-Oriented Architecture (SOA) Defined

Service-Oriented Architecture (SOA) can be thought of as “a methodology for achieving application interoperability and reuse of IT assets” (Newcomer, 2004). According to Krafzig (2005), SOA can be defined as follows:

A Service-Oriented Architecture (SOA) is a software architecture that is based on the key concepts of an application front-end, service, service repository, and service bus. A service consists of a contract, one or more interfaces, and an implementation.

The focus of an SOA is centered on the business processes of an organization. A service is used to meet the needs of a business process. For example “When we use the term ‘service’, we have in mind a business service such as making airline reservations or getting access to a company's customer database” (Krafzig, 2005). These services provide the business processes that carry out the business needs such as getting a reservation, or canceling a booking.

These business services should not be confused with infrastructure processes. Infrastructure processes may include such things as connecting to and accessing data from the database. “Actually, the SOA must decouple business applications from technical services and make the enterprise independent of a specific technical implementation or infrastructure” (Krafzig, 2005).

Delivering the requested information to users is the main value of an SOA. All of the objects that are used to deliver the information are transparent to the

user. These objects may include technical infrastructure objects and business components.

1.3 *Business Need for SOA*

“SOA presents the possibility of finally eliminating corporate silos” (Sippl, 2005, p. 6). SOA enables interoperability across diverse technical platforms (Manes, Jan 2006, p. 7). This interoperability is one key to breaking down the silos and uniting IT systems so they are able to collaborate to meet the goals of the enterprise (Brown, 2007). By adopting SOA principles, IT assets can be “reused, mixed and matched, and assembled and reassembled into the new agile applications” (Sippl, 2005, p. 5). SOA addresses all of the issues that stem from silo based applications to enable the creation of IT systems that promote flexibility, reusability, interoperability and maintainability (Manes, July 2006). This means “applications, services, and products can be offered more quickly and securely, giving an advantage over competitors” (Manes, July 2006).

1.4 *Goals of SOA*

The main goals of SOA are to increase IT systems flexibility, reusability, interoperability and maintainability (Manes, July 2006). “Organizations that don’t adopt SOA will be faced with maintaining an ever-increasing pile of inflexible application silos, duplicate data and functionality, and spaghetti integration challenges” (Manes, July 2006). The following sections take a detailed look at how SOA addresses each of the goals.

1.4.1 IT Flexibility

SOA provides flexibility through loose coupling. “Loose coupling is a fundamental concept of SOA (and large distributed systems in general) aimed at reducing dependencies between different systems.” (Josuttis, 2007). To achieve loose coupling, a service interface must be abstracted from its implementation. By abstracting the service interface from its implementation, SOA removes the complexity of attempting to make software from different languages or from different platforms communicate.

Since the abstracted interface can be a universally understood medium such as XML, it is possible to leverage any application that fulfills a business need. Anne Manes illustrates this point by saying, “XML has had a huge impact on data integration. By providing a standard data encoding format and syntax, XML has significantly advanced the ability to integrate data sets” (Manes, 2006, p. 21).

All that is needed is the ability to wrap the process in a service. This not only supports the short term goals of being agile enough to adapt systems to changing business needs, but it also provides a long term architecture that increases the flexibility of an organizations software systems (Newcomer, 2004). By increasing flexibility, organizations will be able to adapt critical systems quicker than could be achieved by traditional proprietary systems (Sippl, 2005, p. 4).

1.4.2 Reusability

SOA design principles facilitate reusability by taking all of the processes an organization needs to conduct business and organizing them into services. These services can then be reused as needed. According to Manes, some advantages include:

- Increased consistency of business-process execution across diverse applications that reuse common services
 - Reduced duplication of development work among distributed teams
 - Streamlined deployment and maintenance of service code
- (Manes, 2006, p. 5)

An SOA should enable an organization to deal with the latest business requirements by reusing existing business logic (Erl, 2007, p. 505). This allows them to minimize risks while reducing costs from resource, and maintenance overhead (Brown, 2007).

1.4.3 Integration and Collaboration

“The most fundamental requirement of a SOA infrastructure is that it enable interoperability across diverse technical platforms” (Manes, Jan 2006, p. 7). The services of a well designed SOA enable application integration through industry-standard interfaces. These standard interfaces make it possible for applications to make requests, and return results. Web services are perfect for implementing SOA because they provide a way to be accessed from anywhere on the web and they use an industry standard document language (XML) to communicate requests and responses to and from web services. Anne Manes points out that the Enterprise Service Bus (ESB), a component of SOA, “provides the tools and runtime frameworks that developers use to encapsulate legacy

applications and expose them as web services” (Manes, Jun 2005). Since web services can use XML to communicate service requests and responses, they make it possible to integrate applications regardless of the underlying platform or programming languages.

1.4.4 Maintainability

Since SOA design principles strive for reusable services, one desirable result is a more maintainable IT system (Erl, 2007, p. 61). A reduction in redundancy means that maintenance tasks are eased because when changes are required in a process, or problems are identified, there is only one place to make the change (Erl, 2007, p. 61). The complexity of searching through all of the possible implementations of the functionality is greatly reduced by decreasing redundancy. Thomas Erl (2007) makes this point in the following statement:

By centralizing reusable services, logic redundancy can be dramatically reduced. When applied to significant portions of an enterprise, this effectively decreases the quantity of solution logic that needs to be hosted, governed, and maintained. As a result, the physical size of an IT enterprise can shrink, along with the effort and budget required to operate it. (p. 507)

Organization could experience considerable maintenance savings with SOA (Erl, 2007). SOA makes it possible to modify or add processes with less effort, thereby reducing maintenance and development costs (Brown, 2007).

1.5 *Barriers and/or Issues*

Since contemporary SOA is a relatively young architecture, standards that define what it is, and how to achieve it are still emerging (Manes, June 2005, p. 6). As with any new concept, there has been confusion on how to implement a best practice SOA (Manes, June 2005, p.6). For example, many people mistakenly believe that with a web service framework (WSF), they automatically have a SOA (Erl, 2005). These people are under the misguided notion that SOA is only about implementation technology (Erl, 2005). This type of misconceptions has led to confusion that has caused some organizations to flounder when attempting to adopt SOA (Manes, June 2005, p.6). If there is not a clear understanding of the design principles and implementation options that follow the emerging best practice standards, they most likely will not maximize the benefits that SOA can deliver (Erl, 2005).

SOA is not a technology; it is a style of design that is composed of services that can be shared and reused to accomplish specific business functions. Manes (2006) attempts to resolve this misconception as follows:

SOA has at least as much to do with behavior as it does with technology.

Fundamentally, SOA is a style of application design that focuses on implementing software functionality as shared, reusable services, in which each service represents a relatively autonomous business or technical function. (p.6)

An ideal SOA would provide organizations with a way to create new applications from existing services with little or no coding (Josuttis, 2007).

Applications could be assembled from a library of services. A potential barrier to

SOA is that in order to compose business processes, there must be a complete portfolio of services that can be assembled (Manes, June 2005, p. 6). These services must not only be capable of supporting component assembly, they must also be reusable. To achieve this goal of SOA, careful and methodical planning must be undertaken to ensure that every business process is addressed in a reusable service that supports assembly (Erl, 2007, p.270).

“Unfortunately, the industry has not yet codified SOA principles and practices into well-defined design patterns” (Manes, 2006, p.6). The IT industry is still learning how best to implement SOA. According to Manes, some believe that SOA requires asynchronous, event-driven communications, while others argue that SOA should support many different message exchange patterns (Manes, 2006, p. 6). These debates are likely to rage while SOA matures. As SOA adoption becomes more prevalent, best practices will probably evolve into widely accepted standards. This will allow organizations to maximize the benefits of SOA.

1.6 Summary

SOA is a design style for creating distributed systems that provide application functionality and data as services to end-users or to other applications or services (Manes, July 2006, p. 31). Although SOA can be implemented using web services, it is also possible to use other technologies for implementation (Erl, 2005).

According to Manes, SOA could provide a powerful competitive advantage for organizations that adopt it because it provides flexibility, reusability,

maintainability and the ability to integrate data sources and applications (Manes, July 2006, p. 31). This allows businesses to respond quickly to changing business needs such as competitive threats, new partners and new products and services (Josuttis, 2007).

2 Chapter Two: Review of Literature and Research

According to Erl (2005), “in older environments, the construction of the solution was so straight forward that the task of abstracting and defining its architecture was seldom performed“. As IT solutions became more complex, formal architectures began to evolve.

Before the internet, architects generally only needed to be concerned with building systems that supported users within the organization (Erl, 2005). The architect was usually armed with information such as how many users and the typical usage patterns that would need to be supported with the systems architecture (Erl, 2005). Along with the internet came focus on building systems architectures that support a potentially large number of users that access the organization’s IT resources in unpredictable ways (Erl, 2005).

2.1 *What is Architecture*

One popular definition of architecture from the Rational Unified Process is as follows:

Software architecture encompasses the following:

- The significant decisions about the organization of a software system
- The selection of the structural elements and their interfaces by which the system is composed together with their behavior as specified in the collaboration among those elements
- The composition of these elements into progressively larger subsystems; the architectural style that guides this organization, these elements, and their interfaces, their collaborations, and their composition

Software architecture is concerned with not only structure and behavior, but also usage, functionality, performance, resilience, reuse,

comprehensibility, economic and technologic constraints and trade-offs, and aesthetic issues. (Kruchten, 2003, Glossary)

Sun Microsystems (2002) defines architecture as follows: “Architecture is a set of structuring principles and patterns that, when applied to a problem, provides the framework for a solution, which can then be assembled from a set of simpler subsystems or components” (p. 8).

2.2 *Application Architecture*

The application architecture describes the systems structure and how the requirements will be supported. “Application architecture is to an application development team what a blueprint is to a team of construction workers.” (Erl, 2005)

There could be as many different application architectures as there are applications in the organization. One reason that application architectures might vary within an organization is that advances in technology lead to a new and better architecture. An older application may have been developed using COBOL and mainframes. The ability to create distributed software on newer and faster hardware could lead an organization to adopt a different approach for new development efforts.

Many organizations use several different application architectures (Erl, 2005). An organization with many application architectures should “almost always be accompanied by and kept in alignment with a governing enterprise architecture.” (Erl, 2005)

2.3 Enterprise Architecture

The need for enterprise architecture sprang up from complexity that was introduced by having multiple application architectures (Erl, 2005). The ability to manage multiple application architectures was addressed with the concept of enterprise architectures. An enterprise architecture is “a master specification to be created, providing a high-level overview of all forms of heterogeneity that exist within an enterprise, as well as a definition of the supporting infrastructure” (Erl, 2005). Erl (2005) goes on to explain that “an enterprise architecture specification is to an organization what an urban plan is to a city”. For example, if a blueprint is comparable to application architecture, then an urban plan would be comparable to enterprise architecture.

Enterprise architecture provides an all encompassing view of organizations systems (Erl, 2005). It also ensures that individual applications fit into the system as value added components that carry out the goals of the organization. It is common for enterprise architectures to attempt to plan for all future service level requirements (Erl, 2005). According to Cade (2002), the service level requirements are as follows:

- Performance – this is usually measured in response time for a given screen transaction per user.
- Scalability – the ability to support the required quality of service as the system load increases without changing the system. A system can be considered scalable if, as the load increases, the system still responds within the acceptable limits.
- Reliability – ensures the integrity and consistency of the application and all its transactions.
- Availability – ensures that a service/resource is always accessible.

- Extensibility – the ability to add additional functionality or modify existing functionality without impacting existing system functionality.
- Maintainability – the ability to correct flaws in the existing functionality without impacting other components of the system.
- Manageability – the ability to manage the system to ensure the continued health of a system with respect to scalability, reliability, availability, performance, and security.
- Security – the ability to ensure that the system cannot be compromised.

(p. 7)

These service level requirements represent consideration for dependencies between the individual application architectures and the enterprise architecture. Since changes to the enterprise architecture are likely to have an impact on dependent application architectures, it is not uncommon for enterprise architectures to include a plan for evolving the technology to minimize impact of changes (Erl, 2005).

2.4 Service-Oriented Architecture

Service-oriented architecture arose from the labor of enterprise architects searching for a better plan (Erl, 2005). A major hurdle in designing enterprise architecture is making it flexible enough to integrate heterogeneous application architectures into the enterprise. “When numerous, disparate application architectures co-exist and sometimes even integrate, the demands on the underlying hosting platforms can be complex and onerous.” (Erl, 2005). The concept of service-oriented architecture evolved from plans to minimize the impact of adding or changing applications in an enterprise (Erl, 2005). Building reusable and interoperable services that can be exposed as web services provides the capability to integrate virtually any application, regardless of the underlying platform (Josuttis, 2007).

It is important to note that SOA does not require the use of web services (Erl, 2005). All that is required is a vendor-neutral communications platform. Since web services have emerged as the dominant vendor-neutral communications platform, the author will focus on this aspect. The key factor that has generated so much excitement about SOA is that web services enable it to extend across both enterprise and application architecture domains (Erl, 2005).

The value of SOA becomes apparent when it is applied across heterogeneous solutions. “The benefit potential offered by SOA can only be truly realized when applied across multiple solution environments” (Erl, 2005). New services can be created using any programming language on any platform and added to the SOA without impacting any other service, regardless of language or platform. This provides enormous flexibility for any organizations IT systems (Brown, 2007). As new programming languages emerge, they can be adopted, added and integrated with legacy systems (Brown, 2007). There is no need to redesign the old architecture and rewrite old systems. Since SOA facilitates a vendor-neutral communications framework, IT organizations are not tied to a single proprietary development or platform (Newcomer, 2004). If the only available human resources have Java skills, then new service could be written in Java, even if all other systems are written in .Net.

2.5 Comparison of Client Sever and Service-Oriented Architecture

According to John Sullivan (2006), “client/server describes the relationship between two computer programs in which one program, the client, makes a

service request from another program, the server, which fulfills the request". This basically means there is a client that communicates with a server to complete processing tasks.

According to Erl (2005), the common configuration of client server "consisted of multiple fat clients, each with its own connection to a database on a central server. Client-side software performed the bulk of the processing, including all presentation-related and most data access logic".

2.5.1 Location of Application Code

In client server systems, the bulk of the application logic, and therefore the majority of the processing work reside on the client (Erl, 2005). "This results in a monolithic executable that controls the user experience, as well as the back-end resources" (Erl, 2005). This is a one of the most significant obstacles to client server technology. It is very difficult to build client software that will work properly with the multitude of operating systems and varying configurations that act as clients. In a client server environment, it is common to have several copies, or branches of source code that are customized for the operating system or client environment. Writing and maintaining multiple branches of code is time consuming, expensive and is often problematic. The processing power of the client machine may also be a factor in the performance of the client software. This issue is often dealt with by advertising minimum hardware requirements.

In a SOA environment, processing logic is highly distributed. "Each service has an explicit functional boundary and related resource requirements" (Erl, 2005). Having the processing logic located on the server puts the control

back in the hands of the architects and developers. In a SOA environment, since the code resides on the server, the developers do not need to spend time and energy in writing multiple sets of code and trying to address environment issues. Developers can concentrate on writing one set of code for the server environment that addresses the business needs. This removes the complexity of trying to satisfy all possible client environments and puts the focus back on the task of translating business requirements into services.

2.5.2 Presentation and Logic Separation

Client server environments tend to have a presentation layer that is tightly coupled with the application logic. A typical client server application would require an installation on the client that included both the application logic and the presentation. There is usually very little or no separation between the logic and presentation. “This results in a monolithic executable that controls the user experience, as well as the back-end resources” (Erl, 2005).

By contrast, the presentation layer within contemporary service-oriented architecture is loosely coupled to the service logic. Web services facilitate communication between thin clients and the service logic. The thin client could be a web page or “any piece of software capable of exchanging SOAP messages” (Erl, 2005). While it is commonly expected for requestors to be services as well, presentation layer designs are completely open and specific to a solution's requirements.

2.5.3 Software Distribution

Client server environments require the software to be installed on each client machine. This type of distribution can be challenging for IT departments. Imagine installing client software on hundreds or thousands of personal computers within an organization. Consider the possible variations in operating systems and environment variables that may impact the client software. It is possible that some machines may have Windows or Mac or Linux or dual-boot. Software distribution in a client server environment can be a daunting undertaking.

Because web services can be automatically available to anyone in the world with a web connection, there are no software distribution issues in a contemporary SOA environment. Sandy Carter (2007) points out how contemporary SOA solves traditional deployment issues in the following statement:

Leveraging SOA as the underpinning technology for deployment dramatically reduces process times and deployment costs. If the business model has been done right, when business processes or business rules change, they change in only one place, and the results are seen everywhere as needed. This means that IT can implement solutions faster, with better communication and fewer errors.

In a contemporary SOA environment, distribution of the software is a simple matter of publishing the web service. The services that contain the business logic are accessible through the web service.

2.6 Comparison of Distributed and Service-Oriented Architecture

The concept of distributed architecture was formed in an effort to solve some of the problems related to client server architecture. The idea of “breaking up the monolithic client executable into components” and distributing these components across multiple hardware devices “(some residing on the client, others on the server)” became known as distributed architecture (Erl, 2005). Distributed architecture alleviated much of the deployment issues by moving more of the application logic to the servers. “Server-side components, now located on dedicated application servers, would then share and manage pools of database connections, alleviating the burden of concurrent usage on the database server” (Erl, 2005).

In the mid 90s, the internet had a huge impact on distributed computing. Internet technology allowed the client component to be replaced by the browser. “Not only did this change radically alter (and limit) user-interface design, it practically shifted 100% of application logic to the server” (Erl, 2005). As can be seen in the following excerpt from Thomas Erl’s (2005) book, internet technology had a positive impact on communication protocols:

Distributed Internet architecture also introduced a new physical tier, the Web server. This resulted in HTTP replacing proprietary RPC protocols used to communicate between the user’s workstation and the server. The role of RPC was limited to enabling communication between remote Web and application servers.

2.6.1 Location of Application Code

SOA can be thought of as a distributed architecture because the application logic is distributed across one or more servers. However, the traditional distributed architecture differs from SOA in how the logic is divided up. In a traditional distributed architecture, the business logic is created by building components, usually with a homogenous code base. The code base can be heterogeneous, but the added complexity leads most organizations to choose a single programming platform (Erl, 2005). The most dominant platforms are .Net and J2EE. The components that reside on the servers “are designed with varying degrees of functional granularity, depending on the tasks they execute, and to what extent they are considered reusable by other tasks or applications” (Erl, 2005).

In an SOA environment, the idea of components remains in tact, however, the SOA components are carefully designed as services. One service may include some or all of the components in a similar distributed architecture. “These services are designed according to service-orientation principles and are strategically positioned to expose specific sets of functionality” (Erl, 2005). Once the service has been created, it can be exposed via a web service. This “use of Web services establishes a loosely coupled environment that runs contrary to many traditional distributed application designs” (Erl, 2005). A library of carefully designed services can be assembled to build applications that satisfy business needs. Unlike traditional distributed architecture, SOA “fosters reuse and cross-

application interoperability on a deep level by promoting the creation of solution-agnostic services” (Erl, 2005).

2.6.2 Communication Protocol

Traditional distributed computing relies on proprietary APIs to facilitate communication between objects that reside on the same machine.

Communication between components on separate servers, typically rely on RPC protocols. “At design time, the expected interaction components will have with others is taken into account—so much so that actual references to other physical components can be embedded within the programming code” (Erl, 2005).

Communication that relies on embedded code or specific physical references is a prime example of tight-coupling. “This rigid and brittle connection was forged out of necessity because interfaces to and from these chunks of code were not well defined, and connections usually needed to be created via custom code” (Carter, 2007). Once this type of tight-coupling becomes a part of the system, it is very difficult to make modifications or extensions to the system.

In a SOA environment, communication is accomplished through SOAP messages passed between web services. This is an important advantage over the distributed method of using APIs or RPCs. Implementing SOA with web services allows a simple document markup language approach (i.e. XML) to be used to communicate requests and responses to and from the web services. Another important advantage is “the fact that a lightweight document transfer protocol such as HTTP can provide an effective, universal data transfer mechanism” (Newcomer, 2004). It doesn't matter what the underlying operating

system or software happens to be, a web service understands the XML request and understands how to fulfill the request. “Web services can be added to any computer that understands XML and HTTP or XML and most other popular communications transports” (Newcomer, 2004).

2.7 Summary

Many of the characteristics of contemporary SOA were derived from the architectures of the past. “SOA is a radical departure from client-server architecture” (Erl, 2005). Some of the principles that were used to build client server applications are still used for SOA. However, SOA has made huge gains in improving on client server architecture (Erl, 2005).

Although distributed architecture is very similar, SOA has “distinct characteristics relating to both technology and its underlying design principles” (Erl, 2005). SOA provides several improvements over distributed architecture, but the most notable is the achievement of loose coupling. By achieving loose coupling, SOA has overcome the biggest shortcomings of all of its predecessors. These past architectures have evolved into contemporary SOA. The fact that SOA with web services is platform independent and promotes service reuse makes it a very compelling architectural solution.

3 Chapter Three: The Value Proposition of SOA

Challenges such as constant change, rigid IT budgets, increased regulation, and global competition require that investments in new technology deliver value to the business. According to Chris Haddad (2005) of the Burton Group, “After the spending sprees of the year 2000 and the dot-com eras, enterprises have learned a measure of fiscal responsibility and now scrutinize spending initiatives such as service-oriented architecture (SOA) projects very closely” (p.5) .

According to Erl (2007), for established technologies it is a strait forward endeavor to understand the facts and how they will provide value over time in order to measure the return on investment (ROI) (p. 61). New approaches to IT, such as contemporary SOA are more challenging to measure because “the emphasis on increasing ROI typically goes beyond the returns traditionally sought as part of past reuse initiatives (Erl, 2007, p.62). Organizations must make architectural investments long before they can realize tangible return. Recognizing and measuring the value of adopting SOA is more of an art form than a science. Organizations must attempt to recognize the soft benefits as well as the tangible benefits in order to understand the true value of SOA (Haddad, 2005, p. 11).

3.1 Analysis of Business Benefits

There are a couple of important SOA concepts to understand when analyzing business benefits of this architectural approach (Erl, 2007, p. 276). First, services are created at a granular level to represent a piece of a business process (Erl, 2007, p. 276). Once a library of services has been created,

business processes can be built from these services using the Business Process Execution Language (BPEL) (Josuttis, 2007). BPEL is “an XML language for describing business flows and sequences, which in themselves are services” (Josuttis, 2007). According to Josuttis (2007), the beauty of composing business processes using BPEL is that the services used in any given process could be written in any language on any piece of hardware literally anywhere in the world. Josuttis (2007) supports this by saying, “in practice, you can compose processes and services that use different middleware and even native technologies such as J2EE calls”.

3.1.1 Reduced Integration Expense

One of the most significant factors in using Web services is that data exchange is governed by open standards. According to Thomas Erl (2005), “After a message is sent from one Web service to another it travels via a set of protocols that is globally standardized and accepted”. The key concept to grasp here is that Web services are “globally standardized and accepted” (Erl, 2005). Since pieces of business processes are exposed as Web services in contemporary SOA, they can be incorporated into any application that can consume a Web service (Erl, 2005).

“The use of an open, standardized messaging model eliminates the need for underlying service logic to share type systems and supports the loosely coupled paradigm” (Erl, 2005). That means that any organization in the world can integrate those services into their own application regardless of language, platform or database. It does not matter whether they use a Windows or UNIX,

J2EE or .Net. Since Web services are globally standardized, they can be integrated into any application that can communicate with Web services.

3.1.2 Increased Asset Reuse

There are three main areas of asset reuse that can be achieved by following SOA principles. These areas can be categorized as governance, standard interface and granular level design and are discussed in the following sections (Erl, 2007).

3.1.2.1 Governance

A common mistake within IT departments is the waste introduced when time and money are spent building software that has already been built (Josuttis, 2007). Reducing this type of redundancy and achieving software reuse has long been a goal for the IT industry (Brown, 2007). A common reason for redundant software is poor organization or governance of software (Erl, 2007, p. 363). A deficiency of centralized control and interaction between the various departments of organizations results in the same software solutions being duplicated over and over (Erl, 2007, p. 363). “Hence, a centralized registry of services is required for easy discovery and promoting service reuse at the enterprise level” (Mehul, 2007). This centralized registry allows SOA stakeholders to find existing services so they can determine their effectiveness for reuse. According to Mehul (2007), “an entry in such a registry provides functional information such as the name of a service, service operations, and service location for its invocation”. A central repository makes services discoverable so project teams can avoid the mistake of recreating their functionality.

3.1.2.2 Granular Level Design

One of the best known approaches to reduce redundancy is the Object Oriented paradigm which has a “stronger emphasis on modularity and offers a more advanced form of reusability. The latest design architecture in this evolution of reusability is Service-oriented architecture (SOA)” (Mehul, 2007). SOA is similar to the Object Oriented paradigm in that they both attempt to minimize redundancy by organizing software into granular levels of reusable components. This granular level design is intended to enable services to provide encapsulation of reusable logic that does not overlap other services. “Service Reusability emphasizes loose coupling because the lower the dependency requirements of a service, the more easily it can be reused.” (Erl, 2007, p. 279).

3.1.2.3 Standard Interface/Web Services

Web services are the preferred standards-based way to enable messaging for SOA. Web services provide a vendor neutral communications framework that has become a significant enabler of SOA. By organizing pieces of reusable business logic into services and exposing them as web services, they can be incorporated and reused by any application and/or BPEL in any organization that has internet access (and is authorized to use them). A single service could be used in any number of applications. “Because service logic can now be accessed via a vendor-neutral communications framework, it becomes available to a wider range of service consumer programs” (Erl, 2007, p. 50).

3.1.3 Competitive Advantage

For many organizations, business processes and their corresponding software systems are very tightly integrated. It is difficult, if not impossible, to change one without impacting the other. “Altering business processes inevitably requires system changes. Conversely, system changes inexorably alter business processes.” (Brown, 2007). This integration of business processes and IT systems can be a major inhibitor of an organizations ability to adapt to changes. A new business opportunity could be lost to more agile competitors if the software systems cannot be efficiently modified to address the new processing needs.

SOA positions services as reusable assets that can be repeatedly used in different applications. A library of loosely coupled services allows modification of existing processes without major impact on other services. It also allows new business processes to be composed from existing services rather than developed from scratch. “As a result, the time and effort required to automate new or changed business processes is correspondingly reduced because development projects can now be completed with significantly less custom development effort” (Erl, 2007, p. 63).

The benefit to organizations is “heightened responsiveness and reduced time to market” (Erl, 2007, p. 63). This increase in ability to rapidly respond to changes in business can provide enormous strategic advantage over competitors who lack these capabilities.

3.2 Analysis of Financial Benefits

Financial Benefits can be very difficult to measure for SOA endeavors because there are many indirect monetary advantages. A primary concern with SOA is creating IT assets that can be repeatedly assembled into various business processes. Instead of developing a new software component from scratch every time a business need arises, services can be reused to compose new business processes. According to Erl (2007), “logic can be designed for reuse, thereby lowering the subsequent effort to build applications that require the same type of logic” (p. 452). This can lead to faster development of software and a reduction in coding effort, resulting in lower development costs. SOA can effect the operations of organizations by allowing new and improved products or services. Some other benefits include enabling business partner opportunities and simplifying customer facing processes (Erl, 2007, p. 452).

All of these improvements can significantly strengthen the position of an organization. “SOA promises financial economies of scale through sharing and reuse of services” (Haddad, 2007, p. 9). The challenge is quantifying the improvements so that they can be measured in terms of financial benefit. Computing the financial benefit of reusing a service instead of writing new code is not as straight forward as with traditional IT systems. The financial gain achieved by adopting SOA is elusive because the true value is in SOA’s ability to deliver economies of scale and meet future business needs (Haddad, 2005, p. 8).

3.2.1 Financial Metrics

Some of the typical costs that should be considered when applying financial metrics to a SOA project include software licenses, maintenance, hardware, nonrecurring integration cost and administration costs. Table 1 shows a sample of these costs for a three-year period.

Costs	Year 1	Year 2	Year 3	Total
Software license	-100,000	0	0	-100,000
Software maintenance	0	-20,000	-20,000	-40,000
Hardware	-50,000	0	0	-50,000
Nonrecurring engineering	-75,000	0	0	-75,000
Administration	-25,000	-25,000	-15,000	-65,000
Operations	-15,000	-15,000	-15,000	-45,000
Total	-265,000	-60,000	-50,000	-375,000

Table 1: Sample Total Cost of Ownership (Haddad, 2005, p. 21)

Table 2 illustrates net cash flow by subtracting the costs from the estimated benefits.

Benefits	Year 1	Year 2	Year 3	Total
Improved productivity	0	50,000	75,000	125,000
Elimination of redundant development costs	25,000	75,000	150,000	250,000
Increased sales volume	100,000	125,000	225,000	450,000
Total	125,000	250,000	450,000	825,000
Net cash flow (total cost minus total benefits)	-140,000	190,000	400,000	450,000

Table 2: Sample Net Cash Flow (Haddad, 2005, p. 22)

According to Manes of the Burton Group, the “discounted cash flow (DCF) adjusts the cash flow expectations to reflect the cost of capital over time” (Manes, 2006, p. 22). Manes goes on to explain that “DCF is calculated using the following formula: $\text{amount}/(1 + \text{interest rate})^n$, where n = the forecasted year”

(Manes, 2006, p. 22). Table 3 uses the net cash flow numbers from table 2 and computes the DCF using a discount factor of 10%.

	Year 1	Year 2	Year 3
Net cash flow	-140,000	190,000	400,000
Discounted cash flow (10%)	-127,273	172,727	363,636

Table 3: Sample Discounted Cash Flow (Haddad, 2005, p. 22)

After the DCF has been established, the Net Present Value (NPV) can be computed. The NPV “is the current value of all expected future cash flows, discounted by the cost of capital, minus the present value of the proposed investment” (Haddad, 2005, p.22). Table 4 displays the NPV which was computed using the values from the DCF in Table 3. Haddad (2005) explains that the formula used to “calculate NPV is the summation of the cash flows divided by a calculated value equaling the exponent (number of cash flows) of one plus the discount factor” (p. 23).

	Year 1	Year 2	Year 3	Total
Net cash flow	-140,000	190,000	400,000	450,000

	Year 1	Year 2	Year 3	NPV
Discounted cash flow (10%)	-127,273	172,727	363,636	330,278

Table 4: Sample Net Present Value (Haddad, 2005, p. 23)

A positive NPV indicates that the return from a project exceeds the cost of capital. Financial justification for a project can be expressed in terms of NPV. The higher the NPV, the more financially compelling a project is for the organization.

Another important financial metric to consider is Return on Investment. “Return on investment (ROI) is the result of subtracting the project costs from the

benefits and then dividing by the costs” (Schwalbe, 2004, p. 147). Table 5 illustrates how ROI can be computed using sample data.

	Year 1	Year 2	Year 3	Total
Costs	265,000	60,000	50,000	375,000
Benefits	125,000	250,000	450,000	825,000

3-year ROI	220%
------------	------

Table 5: Sample Return on Investment (Haddad, 2005, p. 23)

The higher the ROI, the more an organization will benefit from the project (Haddad, 2005, p. 23). An important thing to note about these financial metrics is that they do not incorporate the economies of scale that can be introduced by SOA. For example, a central principle in SOA is that services can be reused by many different processes and applications (Erl, 2005). To accurately calculate ROI, you would need to estimate the reuses potential and factor that into the benefits (Erl, 2005). This could significantly change the ROI.

3.2.2 Consideration of Soft Benefits

Soft benefits are those that are difficult to measure in terms of financial advantage (Haddad, 2006, p. 10). An organization that thrives on being able to establish a method to share information with business partners would benefit greatly from SOA (Haddad, 2006, p. 11). SOA may even provide the most enabling factor in allowing an organization to outmaneuver competitors (Manes, 2006, p. 11). However, quantifying the value of faster partnership capabilities would leave most organizations in uncharted territory (Haddad, 2006, p. 11). Other soft benefits may include “increased employee productivity, improved customer/partner service, and improved competitive standing” (Haddad, 2006, p.

11). Soft benefits should be given their due attention when considering an investment in SOA. According to Haddad (2005), “the inclusion of soft benefits, or intangibles, can buttress the hard-dollar analysis and make the case even more compelling” (p. 11).

3.3 Risks of Investing in SOA

Adopting SOA does not guarantee that all of the benefits will be realized by all organizations. “Unfortunately, SOA is not an off-the-shelf product. SOA adoption requires careful planning and a profound willingness to change” (Manes, 2005, p. 31). SOA implementations can be more complicated than those of traditional IT system. It can be very difficult to design reusable services that are loosely coupled, heterogeneous, stable and scalable. Some of the challenges that adoption of SOA can introduce are discussed in the following sections.

3.3.1 Weakest Link in the Service Chain

“One of the goals of SOA is to enable organizations to mix and match services and rapidly create new applications in response to changing business imperatives” (Manes, 2007, p. 23). SOA applications can be composed from services on not only internal systems, but from any number of partner systems from all over the globe. That means a composed application is only as stable as the weakest service from the most instable system of a partner organization. If a partner server goes down, it could potentially bring down every business process in the world that uses it. For SOA initiatives to be successful, it is critical that services be dependable in terms of scalability and stability.

3.3.2 Nature of the Organization

Organizations that are heavily segmented may find it difficult to centralize efforts. It would not be surprising to find organizations that duplicate SOA implementation efforts. “A decentralized culture may lead to multiple SOA infrastructure initiatives that result in redundant systems and fewer economies of scale” (Manes, 2006, p. 15). One reason for this type of decentralization could be a lack of communication. Organizations that have departmentalized their operations often operate as separate businesses with their own IT departments. Another reason could be the natural desire to maintain control of all components of a system. There is often a “reluctance to accept a hard dependency on something that’s out of one’s control” (Manes, 2005, p. 24).

To be successful in a SOA effort, care must be taken in the design phase to ensure that services are reusable (Erl, 2007, p. 276). This type of design requires more time without an immediate benefit (Erl, 2007, p. 276). “Why should a line-of-business manager agree to accept the increased burden of developing reusable services just so someone else can benefit?” (Manes, 2005, p. 24).

Before an organization can realize a successful SOA endeavor, they must address these types of cultural issues (Manes, July 2006, p. 6). Anne Manes (July 2006) describes a common cultural impediment to SOA as follows: “In most organizations, current IT efforts are focused on delivering applications as quickly as possible at the lowest possible cost. Organizational structure, accounting practices, and incentive systems all reinforce this goal” (p. 12).

The reward system should be altered to focus on the long term benefit of the entire organization, not just the immediate needs of a single department (Manes, 2005, p.24). Employees should also be educated on the concepts of SOA to ensure everyone is working toward the same goal (Manes, 2005, p.24).

3.3.3 Web Service Framework Immaturity

The Web Service Framework (WSF) is a young technology that has two major governing bodies, OASIS and W3C. These two organizations define the standards that make up web services framework (WSF). W3C is currently working on finalizing the WSF 2.0 standards. Manes (2007) explains some of the issues revolving around the current state of the WSF 2.0 as follows:

The core standards on which the WSF is based (SOAP 1.1 and WSDL 1.1) were never vetted and ratified by a formal standards body, and they contain a number of ambiguities, inconsistencies, and errors. The WS-I Basic Profile addresses the most grievous issues that impede interoperability, but nonetheless, revised specifications are required to address a number of shortcomings in the core framework, such as inadequate support for attachments, asynchronous messaging, routing, and versatile MEPs. (p. 33)

There are obvious risks associated with depending on a technology where the standards are in a state of flux. Changes in the standards could have serious impacts on the technical implementation of a SOA. Until WSF 2.0 is finalized and the technology has matured, there is no guarantee that the IT won't set off in a new direction.

3.3.4 Inexperience with SOA Design Principles

A major design principle for SOA is that software be organized into loosely coupled, reusable components (Erl, 2007, p. 452). This requires a different way of thinking for IT professionals. “In SOA, the focus is on building reusable services and then assembling those services to implement a business process” (Manes, 2006, p. 15). The person who assembles applications may not be the same person who developed the services.

Instead of focusing on individual applications, designers of SOA must focus on designing at a much broader level (Erl, 2007, p. 254). Careful planning must be carried out to design and build services that can be reused by multiple applications often at different geographic locations (Erl, 2007, p. 255). In order to be employed across disparate applications, the services must be granular enough that they are compatible with other business processes (Erl, 2007, p. 255). Too much functionality or too many attributes could render them useless to other applications. “Designers must shift from an application-centric to a service-centric design approach” (Manes, 2006, p. 15).

3.4 Cost of not Adopting SOA

Although there are risks associated with an investment in SOA, the arguments for moving forward with SOA are very compelling. The competitive market position of organizations may depend on their ability to respond to market conditions (Josuttis, 2007). SOA has the potential to provide significant return on investment (Haddad, 2005, p. 23). It promises to provide high value IT assets that are reusable and enable flexible business processes. Manes (2006)

describes the cost of not investing in SOA as follows: “Without SOA, the organization can find itself in a position where it can’t respond to market changes as quickly, is slow to introduce new services, finds it difficult to establish partnerships, or can’t easily join collaborative communities.”

The choice not to pursue SOA can negatively impact an organization’s competitive market position. This is especially true if competitors moving forward with SOA projects. As an organization becomes weaker, they may find it difficult to recover.

3.5 Summary

SOA comes with some risks, but the potential benefits far outweigh those risks. It can provide technology benefits such as “reuse of existing IT assets; quicker development of new software; simplified, integrated, and standardized IT portfolios” (Finneran, 2006). By enabling quicker development, businesses can respond to market conditions faster (Mehul, 2007). Quicker responses to market conditions increase revenue while lowering development and maintenance costs (Josuttis, 2007). It can be a significant undertaking, but if implemented properly, SOA can provide organizations a competitive advantage that enables their success (Manes, July 2006, p. 11).

4 Chapter Four: The Components of SOA

SOA is a design style, not a specific technology (Josuttis, 2007).

However, it is the recent implementation practices revolving around web services that have generated so much excitement about SOA (Josuttis, 2007). Erl (2005) illustrates this point as follows:

Perhaps one day Web services will be supplanted by a superior platform even more capable of bringing the world closer to pure service-orientation. For now, though, the Web services platform (and all that comes with it) is as good as it gets.

The web service implementation details make contemporary SOA solutions stand above those of the past (Josuttis, 2007). Understanding how to bring the theory into practice can help see the reason SOA has come back into the lime light (Josuttis, 2007). This chapter is devoted to delving into these contemporary SOA implementation technologies.

4.1 Implementation Strategy

There has been much debate over how to properly implement infrastructure to support SOA (Manes, June 2005, p.6). Some of the top vendors have formed a group called Open Service-Oriented Architecture (OSOA) collaboration in an attempt to reach a common understanding of SOA (Wikipedia, November 23, 2007). The group's members are made up of IBM, Oracle and many others. The primary objective of the group is "defining a language-neutral programming model that meets the needs of enterprise developers who are developing software that exploits Service-Oriented Architecture characteristics

and benefits” (Edwards, 2007). The vendors from this collaboration effort all have similarities in the major infrastructure components for their respective SOA software solutions. These components are discussed in the following sections of this chapter.

4.2 Enterprise Service Bus (ESB)

There has been a lot enthusiasm in the industry around the subject of ESBs (Manes, October 2007, p.6). There have also been a lot of opinions about what an ESB actually is and does (Josuttis, 2007)). According to Manes (October 2007), “the term “ESB” has been redefined, overloaded, and diluted to the point where it has no precise meaning” (p. 4). Many people have confused the ESB with SOA. In other words, there is a common misconception that an organization could purchase an SOA by buying an ESB from a vendor (Manes, October 2007, p. 7). There has even been an assertion that ESB is an architectural approach, not a middle-ware software product (Manes, October 2007, p. 7).

4.2.1 ESB Defined

Despite all of the confusion, the industry is starting to come together on some of the major characteristics of ESBs (Manes, October 2007, p. 29). The ESB’s main role is to provide interoperability among heterogeneous environments (Manes, October 2007, p. 8). Manes (October 2007) goes on to explain that “an ESB is a service-oriented middleware solution that enables integration of heterogeneous systems by modeling application endpoints as services” (p. 8). The ESB acts as a host for services and abstracts the

implementation details from the service consumer. It also provides mediation to ensure messages are routed and delivered to the appropriate services (Josuttis, 2007). According to Manes (October 2007), most ESBs exhibit the following features:

- Service-oriented middleware: ESBs model application endpoints as services.
 - Standards compliance: ESBs are more standards compliant than previous generations of EAI technology, and in particular, they all support multivendor interoperability using the WSF.
 - Virtualization of service agents: ESBs provide service containers that virtualize a service and insulate the application code from its protocols, invocation methods, message exchange patterns (MEPs), quality of service (QoS) requirements, and other infrastructure concerns. Beyond these basic characteristics, ESBs are a remarkably diverse and disparate bunch of products.
- (p. 8)

4.2.2 Interoperability

ESB's in contemporary SOA make services accessible to applications through Web Services (Manes, October 2007, p. 11). The use of a standard protocol enables generic interoperability with the services. There are many protocols that can be used to facilitate interoperability such as CORBA, Java Remote Method Invocation (RMI) (Josuttis, 2007). However, the most commonly adopted protocol in contemporary SOA is SOAP based web services (Erl, 2005).

4.2.3 Implementation Scenario

In an SOA implementation environment, a common implementation scenario could include developing a service in an integrated development environment (IDE). If the development environment is integrated with the ESB the service can be deployed to the ESB with very little effort (Krafzig, 2004). A

successful deployment results in a service that can be made available to any organization with internet connectivity.

4.3 Business Process Management (BPM)

According to Josettis (2007), “services are typically parts of one or more distributed business processes”. Defining business processes that reflect the day to day operations of an organization is the starting point of SOA. Services are the lowest-level activities of a decomposed business process (Josettis, 2007). Breaking business processes into granular, reusable services is a design step that can be performed in a BPM tool (Josettis, 2007). BPM provides a means to design and model the business processes. However, BPM goes beyond design and modeling to include a method for executing, monitoring and optimizing processes. A wikipedia (October 29, 2007) definition of BPM is:

Business Process Management (BPM) is a field of knowledge at the intersection between management and information technology, encompassing methods, techniques and tools to design, enact, control, and analyze operational business processes involving humans, organizations, applications, documents and other sources of information.

Although some vendors have complete BPM software solutions, many organizations are likely to assemble the components that make up BPM (Howard, 2007, p. 7). According to Chris Howard (2007) of The Burton Group, “the business process management (BPM) infrastructure is a collection of technologies and concepts that facilitate the construction and execution of

composite business processes” (p. 14). Figure 2 illustrates the major components that make up the BPM infrastructure.

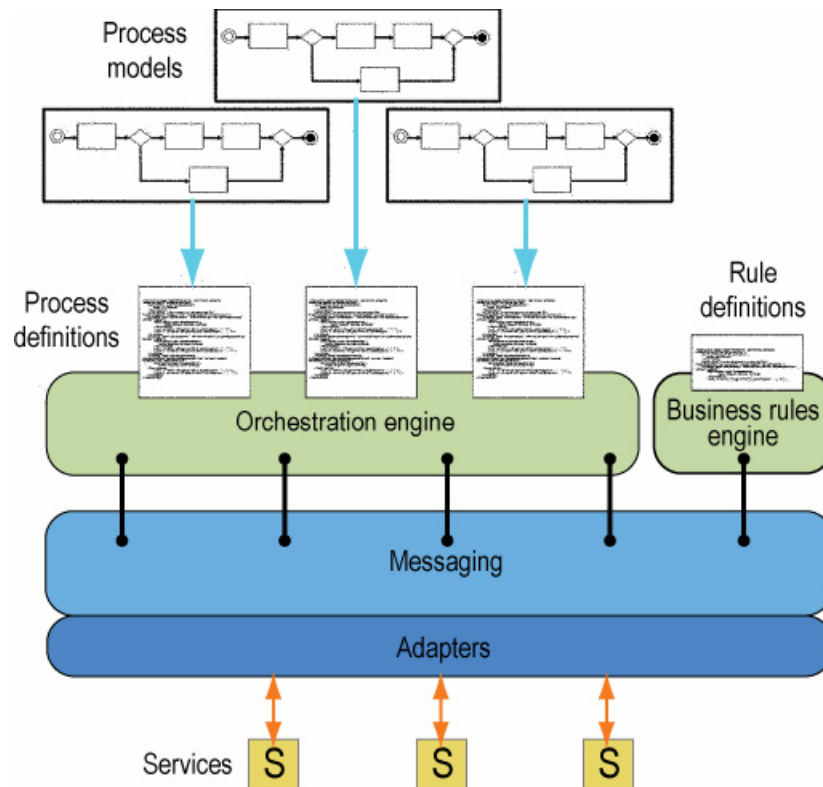


Figure 2 – Core BPM Infrastructure Components (Howard, 2007 p. 17)

The design components consist of a process designer and a process modeler. The process designer is a modeling tool that facilitates a graphical representation of the process flows needed to carry out the business processes (Howard, 2007, p. 15). Once the processes have been modeled they are translated into run time code that can be executed in the orchestration engine (Howard, 2007, p. 15).

The messaging component represents a logical view of the ESB (Howard, 2007, p 19). Figure 2 illustrates how the orchestration engine interacts with the ESB to utilize services within the process definitions. The ESB is sometimes

considered a separate infrastructure component for SOA, but some vendors offer it as a component of the BPM infrastructure (Howard, 2007, p. 7). Figure 2 depicts how “the adapter layer of the architecture provides connectivity to services on the network” (Howard, 2007, p. 19). These adapters enable the ESB to abstract the services from their underlying implementations. Depending on the vendor, the business rules engine may be part of the orchestration engine, or it may be a separate component (Howard, 2007, p. 18). A business rules engine is used to integrate detailed rules within the workflow (Howard, 2007, p. 18).

Another component of BPM that is not depicted in Figure 2 is a Business Activity Monitor (BAM). BAM provides a way for organizations to monitor and measure their business processes. “BAM processes collect data from the various sources and surface it to operations manager or business analysts” (Howard, 2007, p. 20). Many SOA vendors implement BAM software solutions as web based dashboards that integrate with the orchestration engine to collect data about each business process.

BPM is not an absolute requirement to implement a successful SOA (Howard, 2007, p. 8). However, it takes planning and diligence to define the behavior of an organization in terms of its processes (Josettis, 2007). An organizations highest probability for a successful SOA implementation is to “practice BPM in iterations of design, execution, analysis and refinement” (Howard, 2007, p. 13). These activities facilitate best practices by providing a means for defining processes in operational terms so that services can be accurately created to meet business needs (Josettis, 2007).

4.4 Business Process Execution Language (BPEL)

One of the goals of SOA is to provide a way to orchestrate services from multiple sources to build business applications (Juric, 2006). The Business Process Execution Language (BPEL) provides this capability by using a standard process integration model. IBM (2007) defines BPEL as follows:

An XML-based language for the formal specification of business processes and business interaction protocols. BPEL extends the Web Services interaction model and enables it to support business transactions. It is the result of a cross-company initiative between IBM, BEA and Microsoft to develop a universally supported process-related language.

According to Howard (2007), BPEL is a component of BPM (p. 10). It has been purposely separated into this section because most of the vendors offer it as a separate software product including Oracle and IBM. The basic idea is that once a library of services has been created and registered to an organization's ESB, these services can then be assembled into business processes using BPEL (Juric, 2006). This is commonly referred to as orchestration (Jusuttis, 2007). Erl (2005) explains that "Orchestration is more valuable to us than a standard business process, as it allows us to directly link process logic to service interaction within our workflow logic".

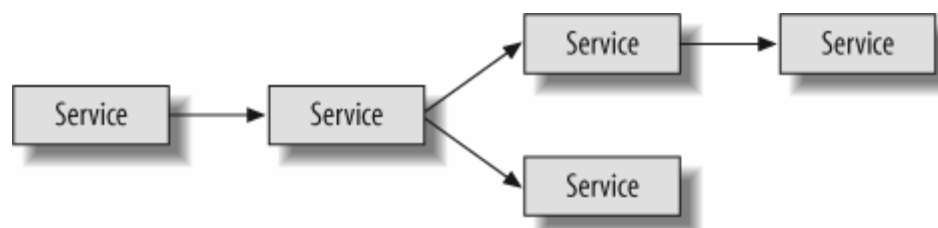


Figure 3 – Business Process Realized as Services (Jussuttis, 2007)

The power of this capability becomes clearer when one considers that not only services from within an organization are available for orchestration, but partner organizations services can be included in the business processes (Juric, 2006). This makes it possible to build business processes from a potentially unlimited number of services that are scattered across organizations all over the world.

Many of the vendors support the use of BPEL in an integrated development environment (IDE) (Juric, 2006). Some commonly used IDE's for SOA development include Eclipse and JDeveloper (Juric, 2006). The IDE provides the BPEL developer component to orchestrate services into a business process (Juric, 2006). The developer selects from a list of standard process activities to define the actions. The IDE also provides a list of services that a developer can choose from to include in the process (Juric, 2006). These services can have a local ESB as the source, or they can use partner links that reside on servers elsewhere (Juric, 2006). As a best practice, UDDI registries are the ideal place to look for all services (Erl, 2007, p. 372). Whether they are internal or services from partner organizations, if they are registered in a UDDI, the BPEL developer will have information to ensure that the services are appropriate for their business process.

An important point here is that these IDE's enable a developer to create BPEL processes graphically, but behind the scenes, they are producing XML (Juric, 2007). That means that BPEL is easily transferred from one environment to another without any proprietary ties. Once the business processes have been

defined using BPEL, they are registered to the orchestration engine where they can be executed (Howard, 2007, p. 15).

4.5 Web Services

The World Wide Web Consortium (W3C) (2007) defines web services as "a software system designed to support interoperable Machine to Machine interaction over a network". An interface described in a machine-processable format, specifically WSDL, is used to enable interaction with other systems using SOAP messages (W3C 2007). Web services are self describing messages that provide interoperability through standard protocols (Josuttis, 2007).

The core technologies behind web services are eXtensible Markup Language (XML) and HTTP. Two other standards are also generally accepted to be major contributors to web services. They are WSDL and UDDI. Nicolai Josuttis (2007) summarizes the standards that are generally accepted to make up web services as follows:

- XML is used as the general format to describe models, formats, and data types. Most other standards are XML standards. In fact, all Web Services standards are based on XML 1.0, XSD (XML Schema Definition), and XML namespaces.
- HTTP (including HTTPS) is the low-level protocol used by the Internet. HTTP(S) is one possible protocol that can be used to send Web Services over networks, using Internet technology.
- WSDL is used to define service interfaces. In fact, it can describe two different aspects of a service: its signature (name and parameters) and its binding and deployment details (protocol and location).
- SOAP is a standard that defines the Web Services protocol. While HTTP is the low-level protocol, also used by the Internet, SOAP is the specific format for exchanging Web Services data over this protocol.
- UDDI is a standard for managing Web Services (i.e., registering and finding services)

According to Josuttis (2007), the only key characteristic of web services is the employment of WSDL. “Everything else is optional. For example, you don’t have to use SOAP and HTTP to send service requests around” (Josuttis, 2007). There are other protocols that can be used, but as long as a WSDL is used, it can still be considered a web service. Although UDDI is an important aspect of managing Web Services, it is often forgotten because it plays only a secondary part as will be discussed later in this chapter (Manes, January 2007).

There is enough information on web services to fill volumes of books on the subject. The key concept to take away from this brief discussion on the topic is that most experts, including SOA architects, vendors and other experts agree that most appropriate way to implement contemporary SOA is with web services (Erl, 2005). Josuttis (2007) stresses this point by saying, “Web Services are widely regarded as the way SOA should be realized in practice”.

4.6 *Application Server*

There are many ways to support applications that use web services and other web based software. One of the key ingredients is a web server such as Apache (Newcomer, 2004). As internet traffic to increases, organizations must have a way to meet scalability demands (Erl, 2005). Web servers can fail to meet the needs of a consumer for any number of reasons. That brings to light a need to plan for increased availability. Security, manageability and performance are also service level requirements that require careful consideration (Sun, 2002, p. 8).

A large IT team committed to building the necessary software components from scratch can address these issues through many hours of design, coding development and testing. Alternatively, IT departments could take advantage of the thousands of man hours that have been dedicated by one of the many vendors by using a commercial off the shelf (COTS) application server. An application server is a central component of the SOA infrastructure for the major vendors, including IBM and Oracle (Manes, 2007, p. 21).

According to wikipedia (October 29, 2007), an application server “is a software engine that delivers applications to client computers or devices. Moreover, an application server handles most, if not all, of the business logic and data access of the application (a.k.a. centralization)”. “The basic functions of an application server can be described as hosting components, managing connectivity to data sources, and supporting different types of user interfaces, such as thin Web interfaces or fat client applications” (Krafzig, 2004).

There are scores of software development platforms, but the two most noted development platforms that facilitate the implementation of enterprise applications on the web are J2EE and .Net (Manes, 2007, p. 21). Oracle, IBM and many other vendors offer application server software for the J2EE platform. Microsoft is the only vendor for its proprietary offering of the .Net framework (Manes, August 2005). The application components of the .Net framework achieve the same core features of most of the J2EE application servers (Manes, August 2005). These core features address issues like availability, security,

manageability and performance, allowing developers to concentrate on building business services instead of infrastructure components.

4.7 Universal Description, Discovery and Integration (UDDI)

A central registry that allows providers to advertise services and consumers to discover those services is a key component of SOA. “Although a registry service isn't required to build and deploy services, it is required to manage and govern the ensuing SOA environment” (Manes, January 2007). A wikipedia (November 2, 2007) definition of UDDI is:

Universal Description, Discovery and Integration (UDDI) is a platform-independent, XML-based registry for businesses worldwide to list themselves on the Internet. UDDI is an open industry initiative, sponsored by OASIS, enabling businesses to publish service listings and discover each other and define how the services or software applications interact over the Internet.

A registry provides a central repository that houses information about the services. There are many vendors that offer UDDI compliant service registries (Manes, 2007, p. 31). These registries often integrate with IDE's that allow designers and developers to discover existing services. “A registry service sits at the intersection of design, development, discovery, provisioning, and management of services” (Manes, Jan 2007).

A typical commercial registry offering consists of two major components, a registry repository and a registry interface. The registry repository is often referred to as a data store or meta-data repository and is usually implemented as

a database (Manes, January 2007, p. 11). Figure 4 shows the basic architecture of a registry.

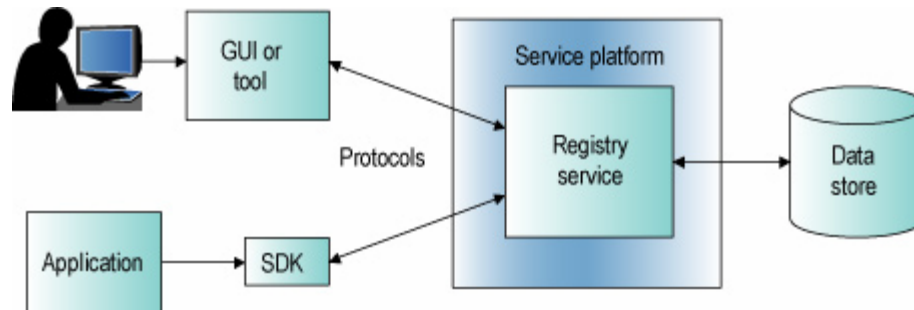


Figure 4 – Registry Components (Manes, January 2007)

The data store in Figure 4 represents the registry repository that stores the data about each service. The registry service provides a graphical user interface that allows service data to be entered and edited (Manes, January 2007, p. 11). It may be implemented as a web or client application. Some of the major IDE's provide plug-ins that allow designers and developers to interface with registries (Manes, January 2007, p. 11). The overall goal of these two components is to enable the following:

- Governance and lifecycle management: Ensures that services conform to corporate principles and best practices, and manages and coordinates service lifecycle stages, such as release engineering, provisioning, utilization, and versioning
 - Virtual system of record: Provides a single point of reference that enables disparate products to exchange information pertaining to the SOA environment
 - Discovery: Finds services and service metadata
- (Manes, January 2007)

Registries contribute to governance by providing a central point of reference for all available services (Erl, 2005). This central point of reference helps organizations to manage services to ensure that they are developed with

purpose and discipline (Erl, 2005). This reduces the risk of duplicating services that already exist and organizes the services to ensure that they live up to their objectives.

4.8 Summary

Recent technology developments have created a lot of enthusiasm around SOA as an IT solution (Josuttis, 2007). These advances in technology have the industry leading vendors converging toward similar implementation components that will enable organizations to realize SOA (Manes, June 2005, p.6). Although there is still some debate over what the components are and how they should be implemented, the industry is starting to see the formation of concrete implementation components (Manes, 2005, p. 6). The components that have been generally recognized by the industry leaders include the following:

- Business Process Model (BPM)
 - Business Process Execution Language (BPEL)
 - Business Activity Monitoring (BAM)
 - Web Services
 - Application Server
 - Enterprise Service Bus (ESB)
 - UDDI/Registry
- (Manes, January 2007)

The major vendors offer software to address each of these infrastructure components for their respective SOA software solutions (Manes, January 2007, p. 21). Although the general trend seems to have the industry focused on these components, there are many ways to realize SOA that may or may not incorporate them (Manes, January 2007, p. 21). As SOA matures the implementation details are becoming more practical and clear.

5 Chapter Five: Analysis and Design of SOA

Careful consideration and diligence must be given to the design phase of a SOA system in order to achieve benefits like “reuse of existing IT assets; quicker development of new software; simplified, integrated, and standardized IT portfolios” (Finneran, 2006). Design of SOA components can be more complex than traditional systems because the designers must take into account possible uses that are not present at design time (Erl, 2007, 255). The design must provide a clear vision of how software components can be divided into services that address both functional requirements and system requirements (Erl, 2007, 255).

Imagine the complexity of building a reusable software component with the goal of making it generic enough that it can accommodate the needs of future business partners while addressing the current needs of present business partners (Erl, 2007, p. 212). All the while, these same components must adhere to system requirements such as security, performance, scalability manageability and extensibility (Cade, 2002, p. 7). How does the designer know if the service will be able to accommodate the functional needs of business partners that don't even exist yet? This chapter is dedicated to revealing best practice design that maximizes the potential benefits identified with SOA.

5.1 SOA Analysis and Design Overview

The overall goal of the analysis and design phases of a SOA system is to create a plan that will facilitate development of a library of services (Erl, 2007, p. 393). Once the library of services has been created, they can be used and

reused by a potentially infinite number of business processes (Erl, 2007, p. 393). The process is very similar to that of traditional IT systems. First, analysis is performed to define business requirements. Once the business requirements are fully understood, design of the services can begin. There are two general categories of business services, task-centric and entity-centric services (Erl, 2005). The artifacts that result from the analysis and design phase will vary depending on the organizations preferences. However, the following two sections discuss generally accepted modeling sources for task-centric and entity-centric services.

5.1.1 Task-centric Services

Task-centric services “contain operations that relate to a particular task within the context of a process” (Erl, 2005). The source of a task-centric service is usually a use-case or a Business Process Management (BPM) model. The clear trend in the industry is in favor of deriving services from a BPM. According Erl (2005), the “advent of BPM has resulted in an industry-wide flurry of process modeling and remodeling activity”. The major software vendors include a BPM component in their SOA offerings further fueling the trend (Manes, June 2005, p. 23).

The starting point is usually to identify all of the business processes that an organization uses to carry out its tasks. These business processes are often expressed as process workflows (Erl, 2005). Once the processes have been identified, they are broken down into their functional components with the ideal of

maximizing reuse. The primary focus is on identifying services that need to be built and determining what logic should be included in each service (Erl, 2005).

5.1.2 Entity-centric Services

“Entity-centric business services generally are produced as a part of a long-term or on-going analysis effort to align business services with existing corporate business models” (Erl, 2005). In contrast to task-centric services, entity-centric services focus first on the entities or objects in a system and their relationships with other entities. The tasks or functions are logically grouped into their corresponding entities. Entity-relationship diagrams or object models are typical sources for entity-centric services (Erl, 2005).

Although the industry is gravitating toward task-centric services, Erl (2005) explains that “when compared to task-centric services, entity-centric services significantly increase the agility with which service-oriented processes can be remodeled”. Entity-centric services “inherent generic nature makes them highly reusable by numerous business processes”. On the other hand, task-centric services are created with a single business process in mind which can lead to them becoming coupled to that process. When the business logic changes, “the context under which the services are used and composed may change as well. This may invalidate the original grouping of service operations and could result in the requirement for a redesign and redevelopment effort” (Erl, 2005). Erl (2005) goes on to explain that “a series of entity-centric services composed by a parent orchestration service layer establishes a desirable SOA, promoting a high degree of agility and accurate business model representation”.

5.2 Design Fundamentals

Learning from experiences is one of the most essential values in any endeavor (Erl, 2007, p. 104). Examining what works and what does not work from failures and successes of the past can help ensure that future projects are more likely to succeed if one follows design fundamentals that previously worked while avoiding or modifying those that failed (Erl, 2007, p. 255). Following design fundamentals, or principles is especially important for a SOA effort because it can be more complicated than a traditional system (Erl, 2007, p. 255). Thomas Erl (2007) illustrates this point with the following statement:

When moving toward a service-oriented architecture, principles take on renewed importance primarily because the stakes are higher. Instead of concentrating on the delivery of individual application environments, we usually have a grand scheme in mind that involves a good part of the enterprise. (p. 104)

SOA design fundamentals should form guidelines that support service-oriented computing benefits. According to Erl (2007), these benefits are as follows:

- Increased Intrinsic Interoperability
 - Increased Federation
 - Increased Vendor Diversification Options
 - Increased Business and Technology Domain Alignment
 - Increased ROI
 - Increased Organizational Agility
 - Reduced IT Burden
- (p. 104)

Application of SOA design fundamentals should be geared toward supporting these goals. The following sections are intended to examine design fundamentals that are effective in realizing these goals.

5.2.1 Service Granularity

According to Erl (2007), the term granularity “is most commonly used to communicate the level of (or absence of) detail associated with some aspect of software program design” (p. 114). When designing a service, there are several factors to consider including the following types of granularity:

- Service granularity refers to the functional scope of the service as a whole, as defined by its functional context.
- Capability granularity refers to the functional scope of a specific capability.
- Data granularity refers to the volume of data exchanged by a service capability.
- Constraint granularity refers to the level of detail to which validation logic is defined for a particular parameter or capability within the service contract.

(Erl, 2007, p. 118)

The functional granularity of a service has a direct impact on the reusability of the service (Erl, 2007, p. 280). In order to meet the design goals of SOA, diligence and careful consideration must be put into the logic that will implement the functional requirements. A design goal is to build services with a functional context that allows it to be reused by other business processes (Erl, 2007, p. 280). Too much logic would most likely reduce the level of reusability, while too little logic could make the service too impractical (Erl, 2007, p. 277).

In contemporary SOA, where web services are used, data and constraint granularity are both usually built into the XML documents that are defined in the

SOAP messages (Erl, 2007, p. 117). The data that a service receives or returns is passed within an XML document. Since all of the data is validated and passed within an XML document, the data and constraint granularity would be considered course grained (Erl, 2007, p. 117). Each of the remaining sections of this chapter consider granularity as a major concern in proper design of services.

5.2.2 Service Contracts

Service contracts are an essential part of service design because they describe the interface to a service. This interface is intended to facilitate interaction with the service. According to Thomas Erl (2007), “a contract for a service (or a service contract) establishes the terms of engagement, providing technical constraints and requirements as well as any semantic information the service owner wishes to make public” (p. 126). For example, a web service contract could be made up of a WSDL definition, an XML schema and a WS-Policy description (Erl, 2007, p. 127).

The overall purpose of a service contract is to consistently provide a description of the service and technical requirements for its use. A service contract may consist of both technical and non-technical documents, but there must be some form of technical contract when there are two software components that need to interact (Erl, 2007, p. 128). Erl (2007) emphasizes the importance of careful design of contracts by saying, “much of service-orientation is dedicated to ensuring that service contracts establish a balanced expression of a service's purpose and capabilities in support of reuse and other key strategic goals of service-oriented computing” (p. 129).

5.2.2.1 Standardization of Contracts

It probably comes as no surprise that as a best practice, formal, standardized service contracts should be established during the design phase. One important consideration in standardization of service contracts is naming of expressions. Figure 5 illustrates how naming standards in a web service can help clarify the meaning of a service.

```
Listing #1
<message name="GetInvoiceRequest">
  <part name="InvoiceCriteria"
    element="bus:GetInvoiceRequestType"/>
</message>
<message name="GetInvoiceResponse">
  <part name="InvoiceDocument"
    element="bus:GetInvoiceResponseType"/>
</message>

Listing #2
<message name="GetInvoiceRequest">
  <part name="RequestValue"
    element="bus:InvoiceNumber"/>
</message>
<message name="GetInvoiceResponse">
  <part name="ResponseValue"
    element="bus:Invoice"/>
</message>
```

Figure 5 – Contract Naming Standard (Erl, 2007, p. 133)

In listing #1, the part name and element names are unclear and misrepresent the purpose of the service. In contrast, listing #2 has naming standards that more accurately describe the function and data in the service. This will reduce the chances of confusion as to the purpose and use of this service. “Because effort is made to consistently clarify the meaning of each service, reuse opportunities for those with an agnostic context are more easily identified” (Erl, 2007, p.133).

Another design consideration is standardization of service data representation. Since contemporary SOA utilizes web services for passing data to and from business services, the focus of this topic is on standardizing web service contracts. When processes in an inventory of services have differing XML schema representations, there must be a mechanism for transforming the data from one format to another (Erl, 2007, p. 133). This can be accomplished by transforming the data with XSLT style sheets (Josuttis, 2007). However, transformations should be minimized because it is very inefficient to execute transformation logic at runtime (Erl, 2007, p. 140). Standardizing message data representation formats will result in fewer transformations. Erl (2007) explains this by explaining that standardization of service data representation, “results in more efficient and simplified interoperability, where runtime message transformation is avoided when two services share data based on a common XML schema (p. 141).

5.2.3 Loose Coupling

Coupling between software components allows the exchange of information for the purposes of automation of an IT system (Erl, 2007, p. 165). There will always be a need for some degree of coupling in any system that is modular in nature. Erl (2007) describe the level of coupling as follows: “A measure of coupling between two things is equivalent to the level of dependency that exists between them” (p. 165). Traditional IT systems commonly created tightly coupled systems. This resulted in inflexible systems and very low reuse opportunities.

A design principle for SOA systems is to minimize the dependencies between services. “When there are fewer dependencies, modifications to or faults in one system will have fewer consequences on other systems” (Josuttis, 2007). There are several types of negative coupling such as dependencies on specific technologies; however, the primary concern during the SOA design phase is in decoupling business services. Erl (2007) explains the design challenge of instilling the principle of loose coupling as follows:

When trying to determine suitable levels of service coupling, our goal is to position the service as a continually useful and accessible resource while also protecting it and its consumer from forming any relationships that may constrain or inhibit them in the future. (p. 167)

During the design of a SOA system loose coupling can be realized by abstracting the service contract from the business logic. Erl (2005) supports this by saying, “Loose coupling is achieved through the use of service contracts that allow services to interact within predefined parameters”. This ensures that any future code modifications will not impact the interface that is recognized by the service consumers (Erl, 2007, p. 167). It is also very important to carefully divide the business logic into services that have enough functionality to be practical, but not so much that they include redundancies with other services. This principle reduces redundancy and maintenance costs and enables orchestration by centralizing logic into reusable services (Erl, 2007, p. 508).

5.2.4 Abstraction

Abstraction, as it relates to SOA, is described by Erl (2007) to mean hiding “information about a program not absolutely required for others to effectively use that program” (p. 212). The functional purpose of a service should be the only concern of the service consumer. The logic, platform, or other proprietary details of a service should not be published as part of the service contract. “The primary reason for us to share less information about what a service encapsulates is so that we can make changes without affecting consumer programs that are already using the service” (Erl, 2007, p. 235).

By hiding the proprietary details and ensuring that interaction between services is performed only through service contracts, this principle promotes loose coupling and therefore flexibility (Erl, 2007, p. 237). The level of abstraction during the design phase should be considered for each service. Careful consideration should be given to what is published in the service contracts. The less detail that is published, the more flexibility the service owner will have in evolving the service over time (Erl, 2007, p. 238).

5.2.5 Reusability

The concept of building IT assets that can be reused is arguably the most important design principle in SOA (Erl, 2007, 259). It may also be the most difficult to achieve (Erl, 2007, 255). Building a software product for a single purpose is a relatively straight forward endeavor. Building one that can be reused has many rewards, but also increases the complexity significantly (Erl, 2007, p. 255). Once that software component is in use by one or more

consuming processes, dependencies are formed (Erl, 2007, 256). Changes can no longer be made without analyzing the impact on all consumers of the component. Erl (2007) expands on the challenges introduced with reusable software components as follows: “A reusable program may furthermore require a hosting environment capable of fulfilling increased availability and scalability requirements” (p. 256).

“Pursuing Service Reusability requires us to position logic so that it is as neutral or agnostic as possible to its surrounding environment” (Erl, 2007, p. 268). A service should be designed where possible to provide capabilities that are not specific to a single business process. This can be achieved by identifying redundant logic across all known business processes. Once identified, the redundant logic should be extracted into a service of its own. Well-designed service “provide the most repeated value in a given inventory” (Erl, 2007, p. 269).

5.2.6 Autonomy

Designing each service as a standalone component is a highly stressed goal of SOA. According to Erl (2007), “autonomy, in relation to software, represents the independence with which a program can carry out its logic” (p. 295). Isolating the logic of a service from outside influence enables it to maintain control of its own functional paths. “The result of achieving enhanced autonomy in software programs is increased reliability and predictability due to the increased independence and isolation in which the programs operate” (Erl, 2007, p. 294).

Once a service consumer starts utilizing a service in a production environment, it is bound that services contract. This has the effect of limiting how the service evolves because the service must honor the existing contract. A well designed service should ensure that “service contracts are designed in alignment with each other to avoid overlap of expressed functionality” (Erl, 2007, p. 301). During the design phase, service logic should also be analyzed for redundancy. All redundant logic should be pulled into a single service where possible. Erl (2007) explains that the practice of service normalization facilitates “a well aligned (and streamlined) service inventory, and because redundancy is avoided, the overall quantity of required services (and therefore the overall size of the inventory) is also reduced” (p. 303).

5.3 Summary

Determining exactly what services should consist of, and how they should interact with other services is a crucial step in delivering a successful SOA (Erl, 2007, p. 104). There are two generally accepted approaches to consider when getting started in the analysis and design of a SOA. Organizations must first consider whether to organize components by their processes, known as task-centric services or by entities, known as entity-centric services (Erl, 2007).

In either case, there are several design fundamentals that are intended to support the goals of SOA. The fundamental concerns of the design and analysis phases of a service include the following:

- Service Granularity
- Service Contracts
- Loose Coupling
- Abstraction

- Reusability
- Autonomy
(Erl, 2007)

Consistent application of each of these fundamentals during the analysis and design phases offers the highest probability for a successful SOA (Erl, 2007, p. 105). The ultimate goal of diligently applying these fundamentals is to ensure that the overall benefits of SOA are realized (Erl, 2007, p. 105).

6 References

- (1) Brown, Paul C, (2007). Succeeding with SOA: Realizing Business Value Through Total Architecture. Pearson Education, Inc. (Safari Tech Books Online).

Paul C. Brown is a Principal Software Architect at TIBCO Software, Inc. His model-based tool architectures underlie a diverse family of applications, including distributed control systems, process control interfaces, internal combustion engines, and NASA satellite missions. Extensive design work on enterprise-scale information systems led him to recognize that service-oriented architectures inherently structure both business processes and information systems, which, in turn, led him to the concept of Total Architecture.

- (2) Carter, Sandy. (2007). The New Language of Business: SOA & Web 2.0. IBM Press. (Safari Tech Books Online).

Sandy Carter has a Bachelor of Science degree in math and computer science from Duke University and an MBA from Harvard. Sandy Carter's background in technology and business gives her the unique ability to author this book that addresses the changing business landscape and how organizations can capitalize on this shift. As an IBM Vice President of Marketing, Sandy Carter taps into her fluency in eight programming languages and meshes it with her award winning marketing prowess to drive worldwide strategic initiatives for one of IBM's biggest bets: SOA. Her track record in strengthening brands through savvy marketing campaigns and strategic acquisitions is evidenced by her past successes including IBM's On Demand and WebSphere e-Commerce campaigns as well as ensuring IBM's SOA initiatives consistently earn third party validation and top leadership rankings by analysts and pundits alike.

- (3) Cade, M., Simon, R., (2002), Sun Certified Enterprise Architect for J2EE Technology Study Guide. Sun Microsystems Press.

Mark Cade has 15 years of experience as a software engineer. Currently, he works at the Sun Microsystems Java Center as a Senior Java Architect, where he has extensive experience creating architectures for J2EE solutions for Fortune 500 companies. Cade is lead developer and assessor of the Sun Certified Enterprise Architect for J2EE exam.

Simon Roberts has worked in the software development and training industries for nearly twenty years. He is a team lead on the programmer and

developer certification projects. Roberts is co-author of *The Complete Java 2 Certification Study Guide* with Philip Heller and Michael Ernest, and *The Java 2 Developer's Handbook* with Philip Heller.

- (4) Eeles, Peter, (2006). What is a software architecture. IBM. Retrieved March 21, 2007 from:
<http://www-128.ibm.com/developerworks/rational/library/feb06/eeles/>

Peter Eeles works for Rational Software, part of the IBM Software Group, and has spent much of his career architecting and implementing large-scale, distributed systems. He is based in the UK, and is Worldwide Community of Practice Lead for Model-Driven Development, assisting organizations in their adoption of the Rational Unified Process and the Rational toolset in architecture-centric initiatives. He is coauthor of "Building J2EE Applications with the Rational Unified Process" (Addison-Wesley, 2002), coauthor of "Building Business Objects" (John Wiley & Sons, 1998) and a contributing author to "Software Architectures" (Springer-Verlag, 1999). He is a regular speaker at conferences worldwide

- (5) Erl, T. (2005). Service Oriented Architecture: Concepts, Technology and Design. Prentice Hall PTR. (Safari Tech Books Online).

Thomas Erl is the world's top-selling SOA author, Series Editor of the "Prentice Hall Service-Oriented Computing Series from Thomas Erl" (www.soabooks.com), and Editor of The SOA Magazine (www.soamag.com). With over 80,000 copies in print world-wide, his books have become international bestsellers and have been formally endorsed by senior members of major software organizations, such as IBM, Microsoft, Oracle, BEA, Sun, Intel, SAP, and HP. His most recent title, "SOA: Principles of Service Design", was released in 2007 and his fourth book, "SOA Design Patterns", is scheduled for publication in 2008.

Thomas is also the founder of SOA Systems Inc. (www.soasystems.com), a company specializing in SOA training, certification, and strategic consulting services with a vendor-agnostic focus. Through his work with standards organizations and independent research efforts, Thomas has made significant contributions to the SOA industry, most notably in the areas of service-orientation and SOA methodology. Thomas is a speaker and instructor for private and public events, and has delivered many workshops and keynote speeches. Papers and articles written by Thomas have been published in numerous industry trade magazines and Web sites, and he has delivered Webcasts and interviews for many publications, including the Wall Street Journal. For more information, visit www.thomaserl.com.

- (6) Erl, T. (2007). SOA Principles of Service Design. Prentice Hall PTR.

- (7) Finneran, John, (09/18/2006). SOA: The New Software Bazaar. The Motley Fool. Retrieved on October 1, 2007 from:
<http://www.fool.com/investing/value/2006/09/18/soa-the-new-software-bazaar.aspx>

John Finneran is a consultant, investment analyst, and writer specializing in the financial value of technology.

- (8) Haddad, Chris, (2005). Building the Business Case for Service-Oriented Architecture Investment. The Burton Group.

Chris Haddad is a Vice President and Service Director at The Burton Group. His emphases include: Web services, J2EE, .NET, WS-*, XML Schema, service-oriented architecture, open source, portals, enterprise information integration, model-driven development

Chris has over 17 years of experience architecting and managing development initiatives, and advising on product strategy for Web service companies such as Grand Central, Flamenco Networks (Digital Evolution), Computer Associates (formerly Adjoin), Securant, Employeease, Jamcracker, and TRX.

Primary Distinctions: An expert in the design of service-oriented architecture and business class Web service infrastructure. Current event chair of the IASA Interop City events, director of the XML Atlanta Users Group, member of the Fawcette Publications editorial board. Chaired the Atlanta Web Services SIG and IBSi Technology Committee, a business service provider consortium forging standards relating to application integration. Has worked on Web services infrastructure for the Apache Axis and Employeease EConnect projects. Granted committer status on the Apache Axis project in 2002. Author, speaker, and technical editor of courseware, presentations, articles, and books on SOA, Web services, and model-driven development.

- (9) Howard, C., (2007). BPM Infrastructure Technology and Standards. The Burton Group.

Chris Howard is a Vice President and Service Director at The Burton Group. His emphases includes: Languages, platforms and frameworks, modeling and domain-specific languages, business process management, composite applications, .NET, user experience design, organizational dynamics, enterprise architecture initiatives

Since 1991, Chris has been actively engaged in the technology industry in a variety of consulting roles, including Former Vice President of Information Delivery and Payment Systems at U.S. Bank, with responsibility for .NET strategy and adoption, J2EE and .NET development standards and support,

branch channel renewal and university research affiliations. Consulting projects include developing and implementing the MIDI specification for the Roland Corporation, multimedia production, global project leadership and enterprise application design. Led a cross-campus research project with Stanford University to study the impact of behavioral economics and consumer motivation on application design in high trust environments.

Primary Distinctions: Frequent speaker for IT industry conferences and universities including Cambridge, Harvard and Stanford Universities. Held teaching and faculty positions at McGill University, the University of British Columbia and the University of Cincinnati. Participated in the development of the MIDI specification. Served as an industry advisor to the Microsoft Patterns and Practices community. Listed in the "Canadian Who's Who" since 1990 for his interdisciplinary achievements.

- (10) IBM. (2007). IBM WebSphere Studio: On-line Glossary. Retrieved on November 2, 2007, from:
<http://publib.boulder.ibm.com/infocenter/adiehelp/index.jsp?topic=/com.ibm.w.sinted.glossary.doc/topics/glossary.html>

- (11) Josuttis, Nicolai, (2007). SOA in Practice. O'Reilly. (Safari Tech Books Online).

Nicolai Josuttis, who wrote *The C++ Standard Library* and *C++ Templates*, both for Addison-Wesley, has worked as a systems architect and technical manager. Recently, he spent two years rolling out a SOA at an international phone company. Nicolai presents SOA tutorials at several conferences, and has been speaking on the subject for multiple years.

- (12) Juric, Matjaz B., (2006). Business Process Execution Language for Web Services. Packt Publishing. (Safari Tech Books Online).

Matjaz B. Juric holds a Ph.D. in computer and information science. He is Associate Professor at the University of Maribor. In addition to this book, he has coauthored *Professional J2EE EAI*, *Professional EJB*, *J2EE Design Patterns Applied*, and *.NET Serialization Handbook*, published by Wrox Press. He has published chapters in *More Java Gems* (Cambridge University Press) and in *Technology Supporting Business Solutions* (Nova Science Publishers). He has also published in journals and magazines, such as *Java Developer's Journal*, *Java Report*, *Java World*, *Web Services Journal*, *eai Journal*, *theserverside.com*, *OTN*, and *ACM journals*, and presented at conferences such as OOPSLA, Java Development, XML Europe, OOW, SCI, and others. He is a reviewer, program committee member, and conference organizer. Matjaz has been involved in several large-scale object technology projects. In cooperation with IBM Java Technology Centre, he worked on

performance analysis and optimization of RMI-IIOP, an integral part of the Java platform.

- (13) Krafzig, D., Banke K., Slama D. (2004). Enterprise SOA: Service-Oriented Architecture Best Practices. Prentice Hall PTR. (Safari Tech Books Online).

DIRK KRAFZIG has twelve years experience in enterprise IT, including a full decade in project management and distributed system design for large-scale insurance industry projects. He is currently the product manager for a standard insurance suite with FJA in Germany.

KARL BANKE has extensive experience with Deutsche Bank as a software architect and consultant for distributed systems, Web services, and Java. He now serves as the general manager for iternum GmbH, Frankfurt, Germany.

DIRK SLAMA is the strategic advisor for Allianz Venture Partners. He has ten years in enterprise IT, including stints with Bankgesellschaft Berlin and IONA, and as CEO of Shinka Technologies.

- (14) Kruchten, P., (2003). Rational Unified Process, The: An Introduction, Third Edition. Addison Wesley Professional. (Safari Tech Books Online).

Philippe Kruchten is professor of software engineering in the department of electrical and computer engineering of the University of British Columbia, in Vancouver, Canada. He joined UBC in 2004 after a 30+ year career in industry, where he worked mostly in with large software-intensive systems design, in the domains of telecommunication, defense, aerospace and transportation. Some of his experience is embodied in the Rational Unified Process (RUP) whose development he directed from 1995 till 2003, when Rational Software was bought by IBM. RUP includes an architectural design method, known as "RUP 4+1 views". His current research interests still reside mostly with software architecture, and in particular architectural decisions and the decision process, as well as software engineering processes, in particular the application of agile processes in large and globally distributed teams. He is a senior member of IEEE CS, member of ACM and INCOSE, the founder of Agile Vancouver, and a Professional Engineer in British Columbia. He has a diploma in mechanical engineering from Ecole Centrale de Lyon, and a doctorate degree in informatics from Ecole Nationale Supérieure des Télécommunications in Paris.

- (15) Manes, A. T, (Jan 2007). Registry Services: The Foundation for SOA Governance. The Burton Group.

Anne Thomas Manes is a Vice President and Research Director at The Burton Group. Her emphases includes: Service-oriented architecture, web

services, XML, governance, superplatforms, application servers, Java, J2EE, .NET, application security, data management

Background: Former Chief Technology Officer at Systinet, a SOA governance vendor (now part of HP). Director of Market Innovation in Sun Microsystem's software group. Manes' 28-year industry background also includes field service and education at IBM Corporation; customer education at Cullinet Software; product management at Digital Equipment Corporation; chief architect at Open Environment Corporation; and research analyst with Patricia Seybold Group.

Primary Distinctions: Named one of the 50 most powerful people in networking in 2002 by Network World. Listed among the "Power 100 IT Leaders," by Enterprise Systems Journal. A frequent speaker at trade shows and InfoWorld, JavaOne, and RSA conferences. She has also authored numerous articles in trade publications. Member of Web Services Journal editorial board. Authored "Web Services: A Manager's Guide," published by Addison-Wesley, 2003. Participated in web services standards development efforts at W3C, OASIS, WS-I, and JCP. Anne earned a BA in Economics at Wellesley College.

- (16) Manes, A. T., Monson-Haefel, R., Niski, J., Robison, L., Howard, C. (2007). VantagePoint 2007-2008: Build for Today, Architect for Tomorrow. The Burton Group.
- (17) Manes, A. T, (August 2005). The Microsoft Superplatform: Setting the Bar in the Superplatform Arms Race. The Burton Group.
- (18) Manes, A. T, (2005). VantagePoint 2005-2006 SOA Reality Check. The Burton Group.
- (19) Manes, A. T, (2006). SOA Runtime Infrastructure. The Burton Group.
- (20) Manes, A. T, (July 2006). Service-Oriented Architecture Enterprise Roadmap. The Burton Group.
- (21) Manes, A. T, (2007). SOA Runtime Infrastructure. The Burton Group.
- (22) Manes, A. T, (Oct 2007). Enterprise Service Bus : A Definition. The Burton Group.
- (23) Mauhmoud, Q. (2005). Service-Oriented Architecture (SOA) and Web Services: The Road to Enterprise Application Integration (EAI). Sun Microsystems. Retrieved on September 27, 2005 from: <http://java.sun.com/developer/technicalArticles/WebServices/soa/>

Qusay H. Mahmoud holds a PhD in Computer Science from Middlesex University (UK), and an M.Sc. in Computer Science and a B.Sc. in Data Analysis, both from the University of New Brunswick (Canada). Qusay is the author of two books: *Distributed Programming with Java* (Manning Publications, 1999) and *Learning Wireless Java* (O'Reilly, 2002), and editor of two books: *Middleware for Communications* (Wiley, 2004) and *Cognitive Networks* (Wiley, 2007). He is a Senior Member of the IEEE, and Associate Editor of the *ACM Transactions on Internet Technology*. Qusay has contributed dozens of Java articles to Sun Developer Network.

- (24) Mehul, J. (07/24/2007). SOA Reusability: Shrinking the Lag between Business and IT. java.net. Retrieved on August 21, 2007 from: <http://today.java.net/pub/a/today/2007/07/24/soa-reusability-shrinking-lag-time.html#service-reuse>

Mehul J. is a writer, researcher, and software engineer working at the forefront of the IT and business worlds. Mehul has worked in the SOA/EAI industry for more than six years, a field that he is passionate about, and has an extensive experience in SOA, EAI, integration architectures, business process management (BPM), and rule based engines. Mehul's 10 years of experience includes working with business and IT users across the globe mainly from North America, Europe, and South Asia wearing hats of technical lead, solution consultant, and trainer. Mehul has also excelled into Internet, J2EE, XML, and internationalization related technologies, tools, and products.

- (25) Newcomer, E., Lomow, G. (2004). *Understanding SOA with Web Services*. Addison Wesley Professional. (Safari Tech Books Online).

In the role of Chief Technology Officer at IONA, Eric Newcomer is responsible for IONA's technology roadmap and the direction of IONA's Orbix E2A e-Business Platforms as relates to standards adoption, architecture, and product design.

Eric has 24 years experience in the computer industry, including more than 15 years at Digital Equipment Corporation/Compaq Computer (from April 1984 through November 1999), where he held a series of positions of increasing responsibility in management and technical roles within the commercial software engineering division, including most recently as manager of the Compaq COM+ Expertise Center. Eric received his B.A. in American Studies from Antioch College, with a minor in computer science.

Greg Lomow is a senior architect and developer with fifteen years of experience building applications and distributed systems for the banking and financial services industry as well as mentoring development teams on the

effective use of object-oriented technology. He received the PhD in Computer Science from the University of Calgary.

- (26) Edwards, Mike. (07/27/2007). Open SOA. Retrieved on October 15, 2007 from: <http://www.osoa.org/display/Main/Home>

- (27) Patrick, Paul. (2005). Impact of SOA on Enterprise Information Architectures. BEA Systems, Inc.

Paul Patrick is the Chief Security Architect for BEA System, Inc. He was the architect of BEA's (and earlier Digital Equipment Corporation's) ObjectBroker CORBA ORB, co-architect of WebLogic Enterprise (now Tuxedo), and has been the security architect for WebLogic Server.

- (28) Robison, L., (Aug 2007). Application Rationalization: Burning Fat and Building Muscle. The Burton Group.
Lyn Robison specializes in B2B commerce on the Microsoft platform. Since 1998, he has worked as a programmer and as a technical lead on numerous projects that produced successful B2B applications. He is a speaker at technical conferences such as WinDev and WinSummit, and teaches courses for developers on B2B commerce. He is also the author of *Teach Yourself Database Programming with Visual C++ 6 in 21 Days* (Sams, 1998).

- (29) Schwalbe, Kathy. (2004). Information Technology Project Management Third Edition. Thomson Course Technology.

Kathy Schwalbe is an associate professor at Augsburg College in Minneapolis and teaches project management, problem solving for business, systems analysis and design, information systems projects, and electronic commerce. Kathy worked for 10 years in industry before entering academia in 1991. In addition, Kathy is an active member of PMI (Project Management Institute). Kathy earned her Ph.D. at the University of Minnesota, her MBA at Northeastern University, and her B.S. in mathematics at the University of Notre Dame.

- (30) Siple, Roger., (2005), Breaking Down the Siloi Wall: Realizing the Vision of SOA. Sun White Paper. Retrieved on February 5, 2008 from: http://www.soahub.com/Architecture/Breaking_Down_Silo_Walls.pdf

Roger Sippl has over 25 years of senior operations and chief executive experience with technology companies. He was founder of Informix Software, Inc. in 1980. Under his direction, Informix pioneered SQL relational databases, 4GL application development tools, and OLTP database technology and is now a part of IBM. Sippl was co-founder and chairman of The Vantive Corporation, a CRM leader and successful public company now part of PeopleSoft. In 1993, he

founded and was chairman of Visigenic Software, helping pioneer distributed computing and application servers in enterprises. Visigenic was acquired by Borland.

Roger Sippl is also a founding partner of Sippl Macdonald Ventures and during the nineties, Sippl invested privately in several successful software companies, including Illustra (acquired by Informix), Broadvision, SupportSoft and Red Pepper (acquired by PeopleSoft). In 2002, Sippl founded, and is currently chairman of, Above All Software, the leading development software for composite applications that enable service-oriented architecture. He earned a B.S. in computer science from the University of California at Berkeley.

- (31) Sun Microsystems Inc., (2002), Adaptable Architecture: Best Practices for Meeting Dynamic IT Requirements. Sun White Paper. Retrieved on October 14, 2005 from:
http://www.sun.com/service/sunps/architect/sun_architecture_whitepaper.pdf
- (32) Sullivan, John (2006). SearchNetworking.com. Definitions. Retrieved February 28, 2007, from:
http://searchnetworking.techtarget.com/sDefinition/0,,sid7_gci211796,00.html
- (33) Van der Vlist, E. (2002). XML Schema: The W3C's Object-Oriented Descriptions for XML. O'Reilly. (Safari Tech Books Online).

Eric van der Vlist is a consultant and contributing editor for XML.com (<http://xml.com>) and writer of the O'Reilly books "XML Schema" and "RELAX NG". He has created and maintains XMLfr (<http://xmlfr.org>), a French portal dedicated to XML and is the editor of the ISO/DSDL Part 10 specification in progress: Validation Management. Eric is a seasoned software engineer and active contributor to XML and XSL mailing lists. He is one of the authors of the RSS 1.0 proposal and the creator of examplotron. He has an engineer degree (B.Sc..) from the Ecole Centrale de Paris.

- (34) W3C. (2007), World Wide Web Consortium: Web Services Glossary. Retrieved on October 24, 2007 from: <http://www.w3.org/TR/ws-gloss/>

7 Appendix

Josuttis (2007) compiled the following glossary. This glossary lists the most common specific SOA terms.

Glossary

See [WS-Glossary] for another glossary of the Web Services community.

2PC (two-phase commit)

An approach for maintaining consistency over multiple systems. In the first phase, all backends are asked to confirm a requested change so that in the second phase the commitment of the updates usually succeeds. In accordance with the principles of loose coupling, in SOA compensation is usually used instead of 2PC.

Activity

Possible term for one step in a business process. In the context of SOA, an activity is typically implemented by a service.

Agent

A Web Services term for participant, which is the general term for a consumer or provider.

Architecture

According to [BassClementsKazman03], the architecture of a computing system is the structure or structures of the system, which comprise software (and hardware) components, the externally visible properties of those components, and the relationships among them.

Asynchronous communication

A form of communication where there is a measurable time interval between the sending and receiving of the content of any message. Message-oriented middleware is typically implemented based on this concept by introducing message queues that queue (persist) messages sent by a system until they are accepted by the receiving system(s).

Asynchronous communication is a form of loose coupling because it avoids the requirement that the sender and receiver of a message must be available at the same time.

Asynchronous request/response

Another name for the request/callback message exchange pattern.

Backend

A system that maintains the data and/or business rules of a specific domain. Usually, it provides a specific role or has a specific responsibility in a system landscape. In SOA, a backend is usually wrapped by some basic services.

Basic service

Common term for services that provide basic business functionalities of a single backend. These services are usually part of the first set of services that wraps or hides implementation details of a specific backend. Basic services can be data-driven or logic-driven. They are the base for composing higher services such as composed services and process services.

BPEL

Business Process Execution Language. An XML-based language used to orchestrate services to composed services or process services. The resulting services are Web Services.

IDEs allow you to create BPEL files using graphical user interfaces. Engines allow you to run (and debug) services implemented with BPEL.

BPM

See business process management and business process modeling.

BPMN

Business Process Modeling Notation. A graphical notation for business processes maintained by the OMG.

Bus

An abstract software pattern used to transfer data between multiple systems. In contrast to the hub and spoke pattern, it uses a federation of components that all follow a common policy or protocol to send, route, and receive messages.

Business process

A structured description of the activities or tasks that have to be done to fulfill a certain business need. The activities or tasks might be manual steps (human interaction) or automated steps (IT steps).

Business processes might be managed (see business process management) and implemented using modeling notations such as BPMN or EPC or execution languages such as BPEL.

Some people differentiate between workflows and business processes by stating that business processes describe more generally what has to be done while workflows describe how activities or tasks should be carried out.

Business Process Execution Language

See BPEL.

Business process management (BPM)

A general term that refers to all activities carried out to manage (i.e., plan, implement, document, observe, and improve) business processes.

Business process modeling (BPM)

According to [BloombergSchmelzer06], "a set of practices or tasks that companies can perform to visually depict or describe all the aspects of a business process, including its flow, control and decision points, triggers and conditions for activity execution, the context in which an activity runs, and associated resources."

Business Process Modeling Notation

See BPMN.

Choreography

A way of aggregating services to business processes. In contrast to orchestration, choreography does not compose services to a new service that has central control over the whole process. Instead, it defines rules and policies that enable different services to collaborate to form a business process. Each service involved in the process sees and contributes only a part of it.

CMMI

Capability Maturity Model Integration. An approach to categorize and improve the product and software development processes of organizations. CMMI is an extension of SW-CMM (formerly just called CMM), which deals with the aspects of software development. Part of it is a model to categorize the maturity of an organization by different levels ("initial," "managed," "defined," "quantitatively managed," and "optimizing").

Compensation

An approach for maintaining consistency over multiple systems. In contract to 2-phase commit, compensation doesn't update all the backends synchronously. Instead, it defines compensating activities to be performed in the event that not all corresponding updates of different systems succeed (regardless of whether the updates are performed sequentially or in parallel). As a consequence, this approach leads to looser coupling of systems; however, it might require more effort to implement. BPEL has direct support for compensation.

Composed service

Common term for services that are composed of basic services and/or other composed services.

Consumer

General term for a system that has the role of calling ("consuming") a service (which is offered by a service provider). Another term used for this role is (service) requestor.

Contract

The complete description of a service interface between one consumer and one provider. It includes the technical interface (signature), the semantics, and nonfunctional aspects such as service-level agreements.

Sometimes a contract is also called a "well-defined interface."

CORBA

Common Object Request Broker Architecture. An OMG standard that allows remote access to objects of different platforms. Although its initial purpose was to provide an infrastructure to access distributed objects,

CORBA can be used as a SOA infrastructure by focusing on its concept of Object by Value (OBV).

Domain

A definable (business) area or scope that plays a specific role and/or has a specific responsibility. In SOA this might be a company, a division, a business unit, a department, a team, or a system.

Domain-specific language (DSL)

A specific graphical or textual notation for a meta model. It allows you to specify the concrete behavior of a model in a precise, condensed, readable, and complete form.

EAI

Enterprise Application Integration (sometimes also just called Enterprise Integration, or EI). An approach to integrate distributed systems such that they use a common infrastructure (middleware and/or protocol). With this approach, for each system it is enough to provide and maintain only one adapter to the infrastructure, instead of a specific adapter for each of the systems with which it communicates.

The infrastructure might use a bus or hub and spoke approach.

SOA can usually be described as an extension of EAI that provides the technical aspect of interoperability. For this reason, the concepts of EAI can be considered as being a major part of or even the same as an enterprise service bus.

EDA

See Event-driven architecture.

Enterprise Application Integration

See EAI.

Enterprise service bus (ESB)

The infrastructure of a SOA landscape that enables the interoperability of services. Its core task is to provide connectivity, data transformations, and (intelligent) routing so that systems can communicate via services. The ESB might provide additional abilities that deal with security, reliability, service management, and even process composition. However, there are

different opinions as to whether a tool to compose services is a part of an ESB or just an additional platform to implement composed and process services outside the ESB.

In addition, while tool vendors tend to define an ESB as something to buy, you might also consider a standard such as Web Services to be an ESB because, conceptually, they define all that is necessary to provide interoperability between systems (without the need to buy some specific hardware or software).

An ESB might also be heterogeneous, using various middleware and communication technologies.

You can consider EAI solutions as (part of) an ESB.

EPC

See Event-driven process chain.

ESB

See Enterprise service bus.

Event

A notification sent to a more or less well-known set of receivers (consumers). Usually, the receivers of an event have to subscribe for a certain type of event (sent by a certain system or component). Depending on the programming or system model, the systems sending the events (the providers) might or might not know and agree to send the events to the subscribing receivers.

You can consider events as part of the publish/subscribe message exchange pattern.

Event-driven architecture (EDA)

A software architecture pattern promoting the production, detection, consumption of, and reaction to events. Some consider EDA to be an extension of or complement to SOA; others consider EDA to be part of the SOA approach (a special message exchange pattern where the service provider sends a message to multiple consumers).

Event-driven process chain (EPC)

A graphical notation for business processes, mainly promoted by SAP.

Fire and forget

Another name for one-way messages (a message exchange pattern where a service sends a message without expecting a response).

Frontend

A system that initiates and controls business processes by calling the necessary services. That is, it acts as a service consumer. A frontend might be a system with human interaction or a batch program.

Governance

In general, a term that describes the task of "making sure that people do what's right." In SOA, governance is about architectural decisions, processes, tools, and policies.

HTTP

HyperText Transfer Protocol. The fundamental protocol of the World Wide Web. In a secure form (using SSL transport-layer security), it is called HTTPS.

Hub-and-spoke

An abstract software pattern used to transfer data between multiple systems. In contrast to the bus pattern, it uses a central component that coordinates all communication between senders and receivers.

IDE

Integrated development environment. A (usually graphical) project-oriented environment for the development of specific software.

Idempotency

The ability of services to deal with messages that are delivered more than once so that redeliveries do not have unintended effects.

In an unreliable network, if you don't receive a confirmation, you don't know whether a message was delivered (it is possible that the receiver processed the message and its response was lost). If you send the message again (to be sure the message gets delivered), the receiver should be able to deal with this second message in such a way that it does not produce an effect different from that of receiving the message only once. For example, if the message is a request to add money to a

bank account, the receiver should add the money only once even if, for reliability reasons, the message was sent twice.

Interoperability

The ability of different systems to communicate with each other. Interoperability between different platforms and programming languages is a fundamental goal of SOA.

Note that standards do not necessarily ensure interoperability. For this reason, in the Web Services world a special organization called WS-I provides profiles to make the standards interoperable.

JMS

Java Message Service. The standard Java API for message-oriented middleware (MOM). Because it is only an API standard, it provides portability (allowing you to change the middleware while keeping the interfaces) but not interoperability (allowing you to use different MOM implementations).

Loose coupling

The concept of reducing the dependencies between systems.

There are different ways to decrease the tightness of coupling between systems, such as having different object models, using asynchronous communication, or using compensation instead of 2PC to maintain consistency. In general, loose coupling leads to more complexity. For this reason, in a specific SOA you have to find the right amount of loose coupling.

Maturity model

A model to categorize the maturity of an organization by different levels. Most famous are the Capability Maturity Model (CMM) and its successor, the Capability Maturity Model Integration (CMMI). Following this approach, many organizations have developed SOA maturity models.

MDSD

Model-driven software development. An approach where a significant amount of schematic code, which has the same structure but varies depending on the concrete situation, is generated out of an abstract model. In the context of SOA and this book, MDSD might also stand for model-driven service development.

Message

A chunk of data sent around as part of a service call. Message exchange patterns define typical sequences of messages to perform service calls.

Message exchange pattern (MEP)

A definition of the sequence of messages in a service call or service operation. This sequence includes the order, direction, and cardinality of the messages sent around until a specific service operation is done.

The most important message exchange patterns are one-way, request/response, and request/callback (asynchronous request/response).

For example, the request/response MEP defines that a consumer sends a request message and waits for the answer, which is sent by the provider as a response message.

Message-oriented middleware (MOM)

Middleware that is based on the concept of asynchronous communication. Examples are WebSphere MQ (formerly MQ Series) by IBM, MSMQ by Microsoft, Tibco Rendezvous, and SonicMQ.

Meta model

A description of a model. A meta model refers to the rules that define the structure a model can have. In other words, a meta model defines the formal structure and elements of a model.

Model

An abstraction. In SOA, a model is typically used to specify services. With the help of MDSD, you can generate different code and other artifacts out of it. The structure of a model is typically described with a meta model. For the model, there are typically one or more specific graphical or textual notations (sometimes called domain-specific languages, or DSLs) that allow you to specify the concrete behavior in a precise, condensed, readable, and complete form.

Model-driven software/service development

See MDSD.

OASIS

Organization for the Advancement of Structured Information Standards. An international not-for-profit computer industry consortium for the development, convergence, and adoption of e-business and Web Services standards. See "<http://www.oasis-open.org>."

OMG

Object Management Group. An international, not-for-profit computer industry consortium for the development of enterprise integration standards. OMG's standards include UML, MDA, and BPMN. See <http://www.omg.org>.

One-way

A message exchange pattern where a service sends a message without expecting a response. Another name for this pattern is fire and forget.

Orchestration

A way of aggregating services to business processes. In contrast to choreography, orchestration composes services to a new service that has central control over the whole process.

For Web Services, BPEL is a standard for orchestration, for which development tools and engines are available.

Participant

General term for a consumer or provider. Alternatively, in Web Services terminology, agent is used.

Policy

A general rule or guideline. In SOA, policies have an impact on the infrastructure (ESB), the provider(s), and the consumer(s). A policy might be a mandatory law (such as a required naming convention) or a goal (such as the maximum number of versions of a service in operation).

Process

A structured set of steps (activities or tasks) to carry out to fulfill a certain need or reach a certain goal.

Different processes are involved in SOA: the goal is to implement business processes. To do this, you must have processes to establish and manage solutions and services (solution lifecycles, service lifecycles, and so on). Also, on a meta level, you have the process of establishing SOA and SOA governance.

Process service

A service that represents a long-term workflow or business process. From a business point of view, this kind of service represents a macro flow, which is a long-running flow of activities (services) that is interruptible (by human intervention).

Unlike basic services and composed services, these services usually have a state that remains stable over multiple service calls.

Profile

In the context of SOA and especially Web Services, a profile is a set of standards, each of specific versions, combined with guidelines and conventions for using these standards together in ways that ensure interoperability.

Provider

General term for a (part of a) system that has the role of offering ("providing") a service, which might then be used/called by different consumers.

Publish/subscribe

A message exchange pattern where a service consumer subscribes to get a notification message from a service provider when a certain condition or state occurs or changes.

The subscription might happen at design time or at runtime. If the provider doesn't know the consumer, this pattern is the base of event-driven architecture, where the notification is an event.

Registry

Registries manage services from a technical point of view, unlike repositories, which manage services from a business point of view. Registries manage all the technical details necessary for using services at runtime (signatures, deployment information, and so on). Usually, a registry is considered to be a part of the infrastructure (ESB).

Repository

Repositories manage services and their artifacts from a business point of view. That is, they manage interfaces, contracts, service-level agreements, dependencies, etc., to help to identify, design, and develop services. Unlike for a registry, the service description should be independent of technical details and infrastructure aspects. That is, it should not be necessary to change a repository when a company switches to a new infrastructure (ESB).

Request

A message that is sent by a consumer as an initial message in most message exchange patterns (see request/response and request/callback).

Sometime this term is also used as a synonym for a service call.

Requestor

Alternative term for consumer (mainly used in the context of Web Services).

Request/callback

A message exchange pattern where a service consumer sends a request message but does not block and wait for a reply. Instead, it defines a callback function that is called later, when the response message sent by the service provider arrives.

Sometimes request/callback is called asynchronous request/response.

Request/response

A message exchange pattern where a service consumer sends a request message and expects an answer.

Usually the consumer blocks until the response message sent by the service provider arrives. Sometimes, however, blocking is not required. In that case, there is a separation between synchronous and asynchronous request/response. The latter is also known as the request/callback message exchange pattern.

Response

A message that is sent by a provider as an answer to a service request (see request/response).

Service

The IT realization of some self-contained business functionality.

Technically, a service is a description of one or more operations that use (multiple) messages to exchange data between a provider and a consumer. The typical effect of a service call is that the consumer obtains some information from and/or modifies the state of the providing system or component.

Services can have different attributes and can fall into different categories. The most famous categorization differentiates between basic services, composed services, and process services.

A service is usually described by an interface. The complete description of a service from a consumer's point of view (signature and semantics) is called a "well-defined interface" or contract.

Service-level agreement (SLA)

A formal negotiated agreement between two parties, which in the context of SOA are usually a service provider and a service consumer. For a specific subject, an SLA usually records the common understanding about priorities, responsibilities, and warranties, with the main purpose of agreeing on the quality of the service. For example, it may specify the levels of availability, serviceability, performance, operation, or other attributes of the service (such as billing and even penalties in the case of violations of the SLA).

Service-oriented architecture (SOA)

There are various definitions for SOA. Some specify only that it is an approach for architectures where the interfaces are services. However, in a more specific sense (and according to my understanding), SOA is an architectural paradigm for dealing with business processes distributed over a large and heterogeneous landscape of existing and new systems that are under the control of different owners.

The key concepts of SOA are services, interoperability, and loose coupling. The key ingredients of SOA are the infrastructure (ESB), architecture, and processes. The key success factors for SOA are understanding, governance, management support, and homework.

Note that Web Services is not a synonym for SOA; Web Services are one possible way of realizing the infrastructure aspects of SOA.

SOAP

SOAP is the basic protocol of Web Services. As an XML-based format, it defines the format of the header and body of a Web Services message.

Formerly the acronym stood for "Simple Object Access Protocol," but because SOAP was neither simple nor for objects or access, the term now stands for itself.

The protocol still allows different types of message exchange. The most commonly used is the document/literal wrapped pattern.

Software architecture

See architecture.

SSL

Secure Sockets Layer. A cryptographic protocol that provides secure communication over the Internet protocol HTTP (which is often called HTTPS then).

Task

Possible term for one step of a business process. In the context of SOA, a task is typically implemented by a service.

Two-phase commit

See 2PC.

UBR

The UDDI Business Registry, which was founded in 2000 with the intention of becoming a worldwide registry for public Web Services. However, the idea didn't work, and the UBR was switched off in 2006.

UDDI

Universal Description, Discovery, and Integration. A Web Services standard for registries. Initially designed for the UDDI Business Registry (UBR), it now serves as a standard for the technical management and brokerage of Web Services.

W3C

World Wide Web Consortium. An international consortium for the development of standards for the World Wide Web, which also develops SOA standards such as XML and SOAP. See <http://www.w3.org>.

Web Services

A set of standards that serves as one possible way of realizing a SOA infrastructure. Initially started with the core standards XML, HTTP, WSDL, SOAP, and UDDI, it now contains over 60 standards and profiles developed and maintained by different standardization organizations, such as W3C, OASIS, and WS-I.

Workflow

Similar to a business process, a description of the activities or tasks that have to be done to fulfill a certain business need.

Some people differentiate between workflows and business processes by stating that business processes describe more generally what has to be done while workflows describe how activities or tasks should be carried out.

WS

General abbreviation for Web Services. Also used as common prefix for Web Services standards.

WSDL

Web Services Description Language. An XML-based language that describes service interfaces from a technical point of view. Although it is a Web Services standard, WSDL can also be used for other infrastructures.

WS-I

Web Services Interoperability Organization. An open industry organization that standardizes Web Services standards as profiles to make them interoperable. See <http://www.ws-i.org>.

XML

eXtensible Markup Language. A human-readable general-purpose notation widely used for the description and exchange of data. Specific XML formats can be defined by and validated against an XML schema definition.

XML Schema Definition (XSD)

A language used to describe a set of rules to which a corresponding XML document must conform in order to be considered valid. It includes a set of basic data types.