Diploma Thesis

State of the Art in Service-Oriented Architecture: Current Advances and Approaches to its Implementation Achieving Distributed Business Process Integration

by

Hendrik Mueller
Born on October 11th, 1982 in Burgstäedt, Germany

Dresden University of Technology
Department of Computer Science
Institute for Computer Networks

Supervised by:

*SAP Labs India Pvt. Ltd. Bangalore, India:
Rajeev Warrier, Vice President NetWeaver Development*

*Dresden University of Technology, Dresden, Germany:
Dr.-Ing. Iris Braun
Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill*

Started on June 1st, 2006
Submitted on November 30th, 2006
TASK DESCRIPTION FOR DIPLOMA THESIS

Surname, First Name: Müller, Hendrik
Major: Media Computer Science
Student Number: 2856386
Topic: State of the Art in Service-Oriented Architecture:
      Current Advances and Approaches to its Implementation
      Achieving Distributed Business Process Integration

GOAL DESCRIPTION

Service-oriented architecture (SOA) as a growing concept for distributed software systems, particularly intended to be used within business environments, opens new ways for business process integration. Nowadays, complex business processes are spread across enterprise borders. Thus, companies using a variety of enterprise resource planning software need to be integrated in a flexible and open way to achieve even automated processing. Current technologies within the context of SOA could help to satisfy these needs. So far, a lot of standards have been established during the last few years, but there are also controversies associated with different ways to realize the complete architectural model. First developments towards a common implementation model of SOA have been started during the end of 2005. The goal of this thesis is an in depth state-of-the-art analysis of existing and emerging standards and technologies in the context of SOA. Also, the focus on business processes should be contemplated. Based on those current advances, different approaches to SOA implementation shall be elaborated illustratively by means of a rough prototype. A sample business scenario with a distributed process will be used as the base for a comparison between these approaches.

MAIN POINTS OF INTEREST

- Discussion of process integration within a business context;
- Complete overview of state-of-the-art technologies and standards associated with SOA, both in an abstract and solution specific way;
- Presentation of current approaches to SOA implementation (including SAP's approach);
- Conception of a simple business scenario and finally an example application which leads to a rough prototypical implementation using different approaches; and
- Analysis and elaboration of obtained results.

External Organization: SAP Labs India Pvt. Ltd., Bangalore
External Supervisor: Rajeev Warrier, Vice President NetWeaver Development
Internal Supervisor: Dr.-Ing. Iris Braun
Responsible Professor: Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill
Institute: Institute for Systems Architecture
Chair: Chair of Computer Networks
Beginning on: June 1st, 2006
To be submitted on: November 30th, 2006

Signature of the responsible professor
Service-Oriented Architecture (SOA), a growing concept for distributed software systems, was particularly intended for use within business environments. SOA opens new avenues for complex business processing spread across enterprise borders, revolutionizing business process integration. It provides a flexible and elegant basis for the integration of disparate enterprise resource planning software used by companies. This also helps facilitate automated processing. The answer to the realizations of all these needs lies in the numerous current technologies pertaining to SOA. Although a lot of standards have been established during the last few years, SOA remains steeped in controversies over the different ways to realize the complete architectural model. Common implementation models for SOA have been under development over the past two years.

The goal of this thesis is an in depth state-of-the-art analysis of existing as well as emerging standards and technologies relevant to SOA. Furthermore, the work closely examines business processes and their relationship to these technologies. Based on current advances, the different approaches to SOA implementation are elaborated and illustrated by means of a prototype. A hypothetical business scenario with a distributed process is used as the basis for a sample implementation and a comparison between these approaches.
Preface

This thesis was written during an internship at SAP Labs India Pvt. Ltd., Bangalore, India. The project was financially supported through a scholarship of the international internship program, “Join the Best 2006,” initiated by MLP AG, a German financial consulting company.

SAP AG, founded in 1972 in Mannheim, Germany as “SAP Systems Analysis and Program Development,” is currently one of the world’s leading providers in business software solutions. SAP software helps enterprises of all sizes around the world to improve customer relationships, enhance partner collaboration and create efficiencies across their supply chains and business operations. Their solutions address more than twenty-five industries, including high-tech, retail, financial services, healthcare as well as working with the public sector. Today, SAP stands for “Systems Applications Products in Data Processing” and its headquarters are located in Walldorf, Germany. SAP employs over 36,000 employees worldwide. With subsidiaries in more than fifty countries, the company is listed on several stock exchanges, including the Frankfurt and New York Stock Exchanges under the symbol “SAP”. SAP Labs India as a major subsidiary was founded in Bangalore in 1998. With currently over 3,500 employees, it is the fastest growing SAP development center worldwide.

The work was carried out under the kind supervision and guidance of Mr. Rajeev Warrier, vice president for SAP NetWeaver Development at SAP Labs India. The work was directed by the efforts of Dr.-Ing. Iris Braun, working with Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill at the Institute of Computer Networks at Dresden University of Technology.

I am greatly indebted to Tarun Ramakrishna for his help and guidance, without which this thesis would not have been possible. Also, I would like to thank Devika Narain and Georg Koester for their help in writing this thesis.

1 see http://www.saplabs.co.in, last checked 24/11/2006
2 see http://www.jointhebest.info, last checked 24/11/2006
3 see http://www.mlp.de, last checked 24/11/2006
5 see http://www.tu-dresden.de/, last checked 24/11/2006
# Contents

1 Introduction
   1.1 Objectives .................................................. 2
   1.2 Thesis Outline .................................................. 2

I Fundamentals

2 Business Process Management
   2.1 Business Processes .............................................. 6
   2.2 Business Process Management ...................................... 7
   2.3 Business Process Integration ..................................... 9

3 Service-Oriented Architecture (SOA)
   3.1 An Introductory SOA Example .................................. 12
   3.2 Evolution of SOA .............................................. 14
      3.2.1 History of SOA .......................................... 14
      3.2.2 SOA Contributions ......................................... 15
   3.3 SOA Reference Model ........................................... 17
      3.3.1 Definition of SOA ........................................ 17
      3.3.2 SOA Services ............................................ 19
   3.4 SOA Stack ..................................................... 22
   3.5 Benefits of Using SOA .......................................... 22

II Key Technologies Realizing SOA

4 Process Modeling Notations
   4.1 Petri Nets .................................................. 27
   4.2 UML Activity Diagram ......................................... 28
   4.3 Event-Driven Process Chain (EPC) .............................. 29
   4.4 Yet Another Workflow Language (YAWL) .......................... 31
   4.5 Business Process Modeling Notation (BPMN) .................... 32

5 Web Services Technologies
   5.1 XML as the Base Technology .................................... 38
   5.2 Transport Layer ............................................... 39
   5.3 Messaging Layer ............................................... 40
      5.3.1 SOAP ................................................ 40
      5.3.2 Web Services Addressing (WS-Addressing) .................. 42
   5.4 Description Layer ............................................. 43
      5.4.1 Web Services Description Language (WSDL) ............... 43
List of Figures

1. Parts of the business process life cycle addressed within BPM. ...................................... 8
2. Need for process integration. .................................................................................................. 9
3. SOA Example: A possible distribution of actors. ................................................................. 13
4. SOA Example: A more technical view on the company distribution. ................................. 13
5. The evolution of the structure of companies and the applied software architectures. ............. 14
6. The abstract concept map of the SOA reference model (see MacKenzie et al. 2006). ............ 18
7. The layered SOA stack. ......................................................................................................... 23
8. A simple Petri net model showing basic structural elements. ............................................. 27
9. A simple UML 2.0 Activity Diagram showing basic structural elements. .......................... 29
10. A simple EPC showing its basic structural elements. ......................................................... 30
11. A simple YAWL example showing its basic structural elements. ....................................... 31
12. The BPMI.org hourglass. ..................................................................................................... 32
13. General process of engaging a Web service. ......................................................................... 37
14. Web service technologies in the overall SOA stack. ............................................................ 38
15. Message path between several SOAP nodes. ....................................................................... 42
16. Isolation of the business process from the a set of services. ............................................... 50
17. The evolving WS-Security Stack. .......................................................................................... 52
18. Distinction between orchestration and choreography for business processes. ..................... 57
19. Timeline showing the development of the most important composition languages. ............. 58
20. Collaboration between service implementations interfaced by a WSCI description. .............. 64
21. A high-level view of the ESB (see Keen et al. 2004). ......................................................... 71
22. Pieces of the JBI Environment. ............................................................................................ 77
23. SCA Set of Specifications. .................................................................................................... 80
24. SCA Assembly Model: Component. ..................................................................................... 81
25. SCA Assembly Model: Composite. ....................................................................................... 82
26. SCA Assembly Model: System. ............................................................................................ 82
27. SCA Policy Framework: Policy Intent types. ........................................................................ 85
28. SDO’s disconnected graph pattern. ...................................................................................... 88
29. Partners of the OSOA Collaboration. .................................................................................... 88
30. Screenshot of the IBM WebSphere Integration Developer. .................................................. 89
31. SCA Composite for the Phonebook. ..................................................................................... 96
32. Entity-Relationship Diagram of the Phonebook Data Model. ........................................... 96
33. SCA Composite for the EDM Project. .................................................................................. 101
34. Entity-Relationship Diagram of the Educational Data Model (EDM). ............................... 102
35. SCA Composite for the Phonebook Webservice. ................................................................. 103
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>SCA Composite for the EDM Project Webclient.</td>
<td>104</td>
</tr>
<tr>
<td>37</td>
<td>SCA Composite for the EDM Project Standalone Swing client.</td>
<td>105</td>
</tr>
<tr>
<td>38</td>
<td>SCA System realizing several applications.</td>
<td>106</td>
</tr>
<tr>
<td>39</td>
<td>Web Interface of SCA Phonebook System Use Case.</td>
<td>107</td>
</tr>
<tr>
<td>40</td>
<td>Web Interface of SCA EDM Project Use Case.</td>
<td>107</td>
</tr>
<tr>
<td>41</td>
<td>J2SE Standalone Swing Interface of SCA EDM Project Use Case.</td>
<td>109</td>
</tr>
</tbody>
</table>
# List of Tables

1. Standards organizations involved in current SOA developments (see Erl 2005). . 16
2. Terms used in OASIS’ Reference Model for SOAs (see MacKenzie et al. 2006). . 20
3. BPMN core modeling elements: Flow Objects (see BPMI.org a. OMG 2006). . . 33
4. BPMN core modeling elements: Connecting Objects (see BPMI.org a. OMG 2006). 34
5. BPMN core modeling elements: Swimlanes (see BPMI.org a. OMG 2006). . . . . 34
6. BPMN core modeling elements: Artifacts (see BPMI.org a. OMG 2006). . . . . . 35
7. Terms used in the SCA Assembly Model (see Beisiegel et al. 2006a). . . . . . . 83
List of Source Code Listings

1. Structure within XML documents. ........................................... 39
2. The basic structure of a SOAP message with sample attributes. .......... 41
3. Sample header block with WS-Addressing SOAP binding for a request message. .. 43
4. Core structure of a WSDL 1.1 document. .................................. 45
5. Core structure of a WSDL 2.0 document. This content is conform to Listing 4. 46
6. UDDI: Rough definition of a business entity and a technical model. ....... 49
7. WS-MetadataExchange request asking for policy information. ............... 49
8. WS-MetadataExchange response returning a metadata reference. ............ 49
9. Basic structure of a BPEL description for business processes. ............. 60
10. A Java snippet activity performing a simple value manipulation. .......... 62
11. WSCI structure that would be included into the WSDL description. ......... 64
12. Structure of a simple WS-CDL document. .................................. 66
13. The Phonebook Data Service Java interface. ................................ 97
14. The Phonebook Data Service Java implementation. ........................ 98
15. Phonebook Data Service Component Configuration. ......................... 98
16. Send SMS Service Reference Configuration. ................................ 98
17. The Phonebook Service Java interface. ..................................... 99
18. The Phonebook Service Java implementation. ................................ 99
19. Phonebook Service Component Configuration. .............................. 100
20. Phonebook Service Configuration. ............................................. 100
21. Project RMI Service Configuration. ........................................... 103
22. SOAP Request from the Phonebook Composite to the remote Web service. 108
23. SOAP Response from remote Web service to the Phonebook Composite. 109
24. DAS configuration for Phonebook database. ................................ 116
25. DAS configuration for EDM database. ....................................... 116
26. WSDL file describing the Phonebook Web service. .......................... 118
27. WSDL file describing the EDM Project Web service. ......................... 120
28. Phonebook Composite configuration file. .................................... 127
29. EDM Project Composite configuration file. ................................... 127
30. Phonebook Webclient Composite configuration file. ........................ 129
31. EDM Project Webclient Composite configuration file. ....................... 129
32. EDM Project Swing Client Composite configuration file. .................... 130
1 Introduction

Evolution is a key principle in the survival of any entity. Whether material or conceptual, anything that does not adapt and grow, eventually perishes. With the advent of the communication revolution, the world became a smaller place. Newer possibilities began to exist with increased connectivity. The face of businesses was rapidly changing. In the evolution of the new kinds of businesses, there was a distinct emergence of a well defined set of philosophies beyond the actual realms of work. At this level, the intricacies of processes were defined discretely in units which when combined represented business processes. With the development of these discrete units, there came the need to study and control their interactions. This had repercussions on the competitive dynamics in the business world. Adapting to the changing business environment became crucial. This meant that serving the entire value chain would bring great rigidity into the system and was thus no longer an acceptable strategy. To ensure survival, the various tasks involved in producing a single product or service had to necessarily be spread over different discrete units, sometimes spanning companies across the entire globe. Each of these would specialize in a dedicated task, according to the best available options. At this point, the true nature of globalization emerged as businesses in a chain would be selected anywhere in the world according to their area of excellence. This, however, posed another problem. Specific access of each of these globally interacting units was limited and restrictive. The issue became a major drain on productivity, delayed effective communication, and shot up the overall costs.

Another important fact to consider is that the Internet was establishing itself as the gateway to the future. Every field of business that wanted to survive the information revolution had to necessarily adopt Information Technology (IT). Then however, after IT established itself as an indispensable part of modern-day businesses, the difficulties with its integration into the actual essence of the business became significant. The need thus arose for synchronization between IT with its final business objectives. Several solutions were attempted since this necessity was recognized. Amongst the most recent and most popular of these is Service-Oriented Architecture (SOA). It became the technology that could weave IT into conformance with businesses, thus achieving one of the ultimate goals for any business, automation.

Those who have analyzed SOA in detail, vouch for its indisputable potential as a cornerstone technology. Its immense conceptual power arises from its utilization of the entire set of functions in a business process, distributed in the form of services. This modularity infuses an unusual amount of flexibility into the process, since services may be independently displaced, replaced, or reused at the discretion of the developer. It also implies that the services combine with any underlying technology, free from the constraints of any one language.

Another interesting aspect of SOA is its emphasis on standards. In order to form a universal perception and ensure a common path for its evolution, SOA has been standardized. It also inculcates previously established standards into its implementation. SOA itself, however, is merely an abstract concept which may be realized through more concrete models at a lower level of abstraction. The model that specifically defines what SOA represents is the Programming
1 Introduction

Model. This, along with several other models and standards complete the goals SOA seeks to achieve.

The abstraction of SOA is a strength in that it serves to define a problem at a higher level of abstraction. The hierarchy of the architecture then permits the logic to trickle down through the layers before it is hard-wired into the system. This gives organizations the enormous freedom of creating specialized teams that concentrate on different levels of implementation. Overall, SOA is the next big breakthrough on the horizon of global business integration.

1.1 Objectives

This thesis addresses readers from the fields of both business and information technology. It does not require any particular pre-requisite knowledge although the reader must preferably have some insight into IT and business technicality. The thesis aims to bring across to the reader the most current technological breakthroughs in business integration viz. SOA.

In this thesis, we consider three primary objectives. The first being a study of the nature of business processes and how the need for implementation of SOA arose. Thereafter, the author clearly elucidates the concept of SOA to the reader. Second, this thesis aims to present an in depth state-of-the-art analysis of technologies and their collective utilization, which creates a multi-attributed, complex system like SOA. The thesis is organized in a chronological manner with respect to the emergence of technologies, for the benefit of the reader. This approach serves to guide the reader through the numerous standards associated with SOA so that these may be better appreciated. Finally, the emphasis lies on the functioning of the entire system and its internal coordination. In other words, different programming models that realize SOA are studied in greater detail culminating in a prototypical implementation. In the thesis, greater prominence is given to the cutting-edge breakthroughs in SOA, many of which will not be released in the market for months and even years to come. It thus hopes to project the future impact of SOA. To practically demonstrate several aspects of SOA and to highlight the advantages and disadvantages of an early implementation, the thesis includes an SCA prototype implementation. SCA, as one of these programming models, appears to be an outstanding choice in the author’s opinion and is thus used for the implementation mentioned above.

1.2 Thesis Outline

The remainder of the thesis is structured according to the aforementioned objectives. The entire thesis is divided into three parts which each pertain to the three objectives mentioned.

Within the first part, “Fundamentals”, we first discuss the fundamental aspects of business processes, their management, and their integration. It explains the reasons for which the development of SOA became necessary. Thereafter, we discuss the basic concepts of SOA. Following an introductory example of an SOA, we turn to its origins, its repercussions and its emergence as a cornerstone technology. The next discussion focuses on the reference model which forms the underlying foundation of SOA. Here we also introduce the importance of the reference model by providing a universal definition of SOA concepts. The section also delves into further technicality by explaining the layered structure of the architecture, known as the SOA Stack. The review of fundamentals concludes as we briefly discuss the benefits of SOA and draw attention to its importance from a business perspective.
In the next part “Key Technologies Realizing SOA”, we scrutinize the complexities of SOA implementation and various diverse mechanisms relevant to it. The main theme here is the detailed presentation of the numerous standards which when appropriately combined, lead to the realization of an SOA. After the integration of IT into business, communication between the business analysts and the IT professionals was initially difficult. This was due to a disparity in the work profiles of the two parties and the lack of a common language to communicate in. Thus the need for standardized modeling notations arose. We discuss these models in the “Process Modeling Notations” chapter. Thereafter, we proceed to a detailed description of Web service technologies, which form one of the most universally-used, integral technologies of the SOA concept. The order of the details of the specific technologies discussed follows the order of the SOA Stack, mentioned earlier. This detailed description of Web service technologies does not alone suffice. Since, service composition is a distinctively important aspect related to SOA, we discuss it in a separate chapter.

In the final part, “SOA Implementations”, we delineate the different approaches to SOA implementation. To be more specific, we discuss the concepts of ESB, Java EE 5, and SCA, with respect to their application to SOA. We then look at a sample SCA implementation of the described notions on SOA, in order to support its theoretical description with a practical application. SCA was used for this purpose because its future potential promises to deliver the most, compared to other solutions. Finally, the last chapter summarizes the past dealings and future prospects in store for SOA.

Following this, the thesis culminates in a user-friendly glossary of key words and acronyms for the reference of the reader. For further reference, there exists a detailed bibliography and index for all the relevant content of the thesis.
Part I

Fundamentals
2 Business Process Management

In the last decade of the twentieth century, business processes came under serious scrutiny. The next step in the business process evolution came in the form of Business Process Reengineering (BPR) introduced by Hammer a. Champy (1993). Soon thereafter, Business Process Management (BPM) became an indispensable part of effective modern day business strategy. Earlier, BPM was solely regarded as a tool for modeling, in time however, it extended to inculcate other management activities. At the same time, IT specialists started to integrate the business process paradigm into software solutions, which were then referred to as Business Process Management Systems (BPMS). In the 1980s, IT was only concerned with data management without analyzing business activity automation, however, they subsequently started to develop advanced application management tools. Enterprise Resource Planning (ERP) systems were the first of their kind to exhibit the capacity to model complete enterprise processes. These developments gave rise to a heterogeneous, distributed, n-tier architecture, which has since undergone several enhancements. Further improvisation of BPM and process integration is still in progress.

First, in this chapter, we briefly review the definitions and characteristics of processes, especially business processes. Subsequently, we look at BPMSs and components of BPM which lead to a discussion about process integration issues. After this brief business insight into BPM and process integration, in the following chapters, we focus on the IT perspective.

2.1 Business Processes

A process is a structured and controlled sequence of activities viz. a chain of events, which describes how given inputs are transformed into outputs. Activities are nonseparable actions viz. parts of the process that exclude any decision making and thus need not be split any further. Inputs represent the prerequisites for the execution of the activities. Outputs are clearly defined outcomes or results. Processes always rely on a certain infrastructure, like machines, means of transportation, or IT.

The first developments in business processes began in the 1930s. Business analysts began to realize that a dedicated organizational plan results in an isolation from the customer interface. The focus on quality then, led to the development of a process-oriented business concept.

A lot of formal definitions exist for a “business process”. A really simple approach by Bider (2000) describes a process as a “set of partially ordered activities intended to reach a goal.” A much more comprehensive definition is given in Hammer a. Champy (1993): “A business process is a collection of activities that takes one or more kinds of input and creates an output that is of value to the customer. A business process has a goal and is affected by events occurring in a the external world or in other processes.” However, all definitions share some common features:

- Business processes are driven by outside agents and customers.
2.2 Business Process Management

- They are designed to achieve a commercial goal.
- They produce something of value for the organization itself, its stakeholders, or its customers.

Examples include “receiving orders”, “invoicing”, “shipping products”, but also “employee data updates”. Business processes are generally understood as large processes composed out of several subprocesses that may cross functional, departmental, and/or organizational boundaries. Subprocesses are separate processes with their own goals, owners, activities, inputs, and outputs and may be included into several processes. Thus, one process may use other business processes or may be part of another process. Business processes are usually carried out collaboratively by a group of people and a set of computers.

Business processes may be classified in the following way:

- **Management processes**, in which higher-level, strategic operations are defined.
- **Operational processes**, in which the value generation or delivery to the customer is defined. Such processes represent a part of the core business. An example is the “delivery of goods”.
- **Supporting processes**, in which activities and flows assisting the core processes are defined. Examples are “accounting” and “recruiting”.

Another classification pertains to the relevance of the process to office or production:

- **Office processes**, in which people are participants in a social system to achieve a given goal.
- **Production or machine processes**, in which the focus is on the activities being performed and not on the final goal. Such activities are usually performed by machines.

The remainder of this work does not exclusively discuss any type of business process. Several aspects of control, management, and integration of business processes are discussed in the following section.

### 2.2 Business Process Management

**BPM** addresses how organizations may identify, model, develop, deploy, manage, and analyze their business processes; this includes processes that involve IT systems and human interaction. BPM aims at modeling business processes which are subsequently developed further by the IT department which provides the infrastructure to control and execute them. Besides that, the automation and streamlining of business processes increases employee productivity and reduces operational costs. Furthermore, it increases corporate agility and flexibility by explicitly separating process logic from other business rules. By doing so, it represents business processes in a form that is easy to change as business processes change. Finally, BPM reduces development costs and effort by employing a high-level, graphical programming language that allows business analysts and developers to quickly build and update IT systems within a particular problem domain.

**BPM** represents a business process life cycle (see Figure 1) with the following central activities:
Business Activity Monitoring (BAM) may be viewed as an additional part of the life cycle. It focuses on delivering real-time information on the health of a company’s business processes, to its executives. BAM products typically appear as a graphically-rich presentation layer, backed with sophisticated event-correlation technology. It relies on timely notification of business-related events and performs customizable analysis of those events to determine the efficiency of the business processes within the company.
Most BPMSs only provide a process modeling tool which allows processes to be defined graphically; however, a complete BPMS may yet serve all activities of the business process life cycle. They hence represent a generic software system driven by an explicit process design, that enact and manage operational processes. BPMSs may be used to achieve process integration, which is described ahead. A BPMS either uses services to connect applications to perform business operations or sends messages to human actors requesting them to perform certain tasks which require a human attribute. A prerequisite of effective BPMS operations is that the underlying software must be constructed according to the principles of SOA.

2.3 Business Process Integration

Today’s business processes are modeled through large software solutions, BPMSs, as mentioned before. These enterprise environments are composed of multiple, independently-developed information systems that must be coordinated and enabled to support standards-based, cross-enterprise communication. Applications manufactured by different vendors, legacy systems, and new systems need to be integrated to achieve a common process goal. Also, integration amongst various business units becomes important for BPMS to meet the requirements of today’s complex business processes. Such business units may possibly be situated within the same organization, within various subsidiaries, or even within external organizations like suppliers, customers, and other business partners (see Figure 2). Hence, independent applications and processes need to be integrated and “woven” together to achieve the goal of business process automation.

![Figure 2: Need for process integration.](image)

Ideally, process integration provides a distinct separation between the definition of the process, the execution of the process, and the implementation of the individual functions in the application. This separation allows the application functions to be reused and rearranged within many
different processes. Process integration software provides the infrastructure for the automation of business processes spanning different applications and/or crossing company boundaries. The term to describe the integration traversing several applications is Application-To-Application (A2A). Business-To-Business (B2B), on the other hand, describes the integration of several independent organizations. In the following chapters, we will discuss software architectures, models, and standards facilitating such an integration in greater detail.
3 Service-Oriented Architecture (SOA)

The increasing pace of evolution of business requirements and the need for higher revenues and cost optimization require companies to align their IT organizations more closely with their business requirements. The development of optimally and operationally integrated business processes represent the main goal of this convergence. This integration challenge demands a technology that can successfully combine the needs of business and IT into a viable solution that not only makes efficient and effective use of the IT infrastructure, but is also flexible and adaptive enough to keep step with continual changes in the organization’s business process and business model.

The concept of service orientation offers an advanced framework for integrated distributed systems that directly address the requisite business needs. Generally, service orientation represents a distinct approach for modularizing large problems. In other words, the logic required to solve large problems may be better constructed, executed, and managed if it is broken down into small, manageable units. Today’s business processes, as mentioned earlier, represent similar problems that may be solved through a decomposition approach. This decomposition is also imperative because business processes usually span several boundaries, with the Internet being the most basic underlying infrastructure. It thus necessitates the integration of various applications using different technology platforms and data models.

Moving to SOA, we see the introduction of a common understanding of subtasks or core functionalities as services. The use of this concept of services makes the object in question, independent of the platform on which the actual application runs as well as independent of the implicit technologies. Functionalities are wrapped as public, independently-described services that present the possibility to flexibly compose complete distributed and automated business processes as well as to reuse them within several processes.

Applying SOA to realize complete business processes is currently the most popular way to realize BPM; SOA merely represents the IT side of BPM. Currently, industry leaders, like IBM, Oracle, and SAP, are engaged in bringing the solutions of this new software architecture into effect, since they realize its potential for contemporary business processes.

The SOA concept is suitably elucidated by Bieberstein et al. (2005) from both a business and a technical point of view as:

From a business point of view, an SOA may be expressed as a set of flexible services and processes that a business wants to expose to its customers, partners, or internally to other parts of the organization. In this context, these same services may be recombined and supplemented to support changes to or an evolution of business requirements and models over time. From the technical point of view, SOA defines software in terms of discrete services, which are implemented using components that may be called upon to perform a specified operation for a specific business task. The SOA concept evolves the existing software concept of a function – a specific piece
3 Service-Oriented Architecture (SOA)

of code that performs one particular task – to include the notion of a contract, a
technology-neutral but business-specific representation of the function.

The remainder of this chapter describes this new software architectural concept in an abstract
way. First, a simple example explains the need to apply SOAs. Later, we look at the evolution
of SOA. Thereafter, we conclude our discussion with a recent SOA reference model published
by the Organization for the Advancement of Structured Information Standards’ (OASIS) and a
presentation of business benefits.

3.1 An Introductory SOA Example

In order to introduce SOA, let us consider a hypothetical business process. It elucidates the
practicality and requirement of SOA as an underlying software architecture. This example
describes the common process of an end customer ordering a technical product, like a laptop,
a camera, or an audio player, using the Internet. Such a business process may involve a huge
number of different actors or companies, that may be spread across the globe (see Figure 3). These actors involve the following:

- An end user with the intention to buy a new technical product.
- An online shop provider offering a huge selection of products.
- A business allowing payment and money transfer through the Internet.
- An enterprise, with a storehouse, selling one set of products.
- An assembly unit putting single components together.
- Several factories manufacturing these single components.
- A shipping company having offices spread worldwide.

The Internet serves as the implicit infrastructure allowing all of these entities to communicate
with each other in order to realize a complete business process. Since the IT of each of the
involved enterprises might utilize different technologies and applications, a common ground must
be found to integrate these distinct applications, platforms, and technologies. The following is
a sample process demanding such a technological linking of several entities.

Let us consider the scenario where the process may be initiated by a customer in India, who would
like to buy a new portable audio player. After some online research for the cheapest retailer, an
online shop with its headquarters in Germany, is selected by the customer. The customer logs on
to the required web page, selects the base product and customizes it. Then, an automated check
on the availability of this specific product is initiated by the online shop. If available, the process
will take a simple path and immediately announce the delivery date after contacting the shipping
company. In our case, however, the audio player is not available and thus cannot be directly
sold from stock. And thus, the request will be forwarded to the assembly unit located in, say,
China. Immediately, this request will be analyzed to identify all components that are necessary
to assembly the final product. The availability will also be checked with the suppliers. The
whole inquiry originally initiated by the end customer in India will be processed synchronously.
It results in the generation of a delivery date after all requirements have been met, including
the final delivery duration quoted by the shipping company. Now, the customer accepts this
offer, submits it, and enters his payment details which are processed by a separate e-commerce
company located in, say, the USA. At this point, the whole process is ready to start: The online
store orders the audio player from the US company; They forward this request to the assembly
unit in China which orders the components from the manufacturers and finally assembles them
after they have been shipped to the assembly unit; The final product is picked up in China by
the shipping company and then transported to the end customer in India. During the complete
execution cycle, a lot of information will be exchanged between the participating companies.
Payment, accounting, shipment, and several other issues will be performed automatically with
least human interaction possible to allow a fast and cost-effective process execution. Also, during
the entire time, it must be possible for each participant, including the end customer, to track
the order. In addition, security and other Quality of Service (QoS) issues need to be carefully
addressed at each step.

Such a complex business process may only be carried out with automation through the use
of the Internet as a basic messaging infrastructure. Also, participating enterprises must set up
contracts and agreements on the kinds of messages that are being exchanged in order to allow the
automated processing of the messages. These factors call for a common technology to provide a basis for all operations. SOA provides such a technology, that encapsulates functionalities from different companies, irrespective of the technology used, into services. Such services may then be accessed universally through a common interface definition. Figure 4 represents a more technical version of the network shown in Figure 3.

The example is designed to delimit the basics of SOA and exhibit its qualities as a flexible, distributed software architecture, as well as, its fast automated processing capabilities.

3.2 Evolution of SOA

The rise of SOA and its establishment as a fundamental technology in business environments, is analyzed here as two separate facets. The first describes historical developments while the second facet deals with several recent contributors viz. standards organizations and vendors.

3.2.1 History of SOA

Business organizations have undergone tremendous changes during the last decades. Wilkes (2002) organizes these changes roughly into three phases (see Figure 5). In the 1980s and earlier, enterprises pursued an organization-focused approach, where a company was clustered into vertical, isolated divisions. Wilkes (2002) refers to these divisions as “stovepipes”. This partitioning matured to horizontal business-process-focused structures in the late 1980s and 1990s. The parts were then referred to as “tunnels”, because the business process simply tunnels several function units. At the beginning of the new millennium, enterprises focused on extended supply chains including enterprises, customers, and other partners. Thus, organizations began to shape as an ecosystem, where one “blob” flexibly connected all units of a process.

![Figure 5: The evolution of the structure of companies and the applied software architectures.](image)

The evolution of business organizations also had certain repercussions on the related IT aspects of business. Stovepipes were implemented with a monolithic architecture. In this architecture, application logic, processing, data, and user interfaces reside in the same system. Such systems are often referred to as mainframe computers. Connected clients were “thin” and interacted directly with the bulky, central server through synchronous and asynchronous message exchange. This architecture, however, clearly lacked accessibility by multiple users. At the time, the whole system was only being used within the organization and integration was achieved only through
3.2 Evolution of SOA

simple paper and file transfer. The use of mainframes led to low-integrity, isolated applications. Later, in the mid 1980s, the IT architecture and distribution altered into more structured, decentralized systems, since automated data processing became increasingly necessary in several departments. This, in turn, created more flexibility in systems offering specific capabilities to satisfy specific needs.

With increased business process focus within organizations in the late 1980s, one-tier monolithic systems were replaced by a more advanced two-tier client-server architecture. This new approach introduced the concept of delegating logic and processing duties to individual workstations, resulting in the birth of the “fat” client. The server back then only served as a database and performed data-related logic and processing. A scalable set of clients communicates with the server using one connection each for synchronous data exchange. In conclusion, integration was then achieved through messaging, database replication, and data warehousing; however, client-server systems remained a tightly-coupled technology and applications were used internally.

Later, three-tier and even other multi-tier or n-tier architectures replaced the previous ones. In a three-tier architecture, the user interface, the functional process logic viz. the business rules, and the database are developed and maintained as independent modules. This separation bears the advantage, that any of the three tiers may be upgraded or replaced independently as requirements or technologies change. In a higher level multi-tier architecture, an application is executed by even more distinct software agents. Using these architectures, applications were still utilized internally in the enterprise.

In the mid-to-late 1990s, the World Wide Web (WWW) began to emerge as the basis of interaction amongst several businesses. Multi-tiered client-server architectures began to incorporate the Internet technology. Most significant was the replacement of the custom software client with the browser, a “thin” client. Such clients do not require the installation of any logic and perform presentation-related processing only. Hence, clients and servers became distributed and functionalities were modularized. Distributed, decentralized Internet architectures then represented the de facto computing platform for custom developed enterprise solutions. Such applications were not limited to internal use anymore, but were prepared for customer and partner access, forming large networks or “blobs”, as named by Wilkes (2002). The structure, spanning several companies, introduced certain problems, because it involved entities on disparate platforms and varying technologies. One of the answers of this crisis may be found in the employment of service orientation. Here, the requirements are identified and then encapsulated into standardized services. Preexisting solutions may also be included easily by wrapping them in services. This may be seen as a higher level of abstraction, since the services are made available through a unified description. Services may be composed to actualize more complex business processes. The service, in this context, may be reused by any other appropriate business process. At this point of time, SOA was introduced as a new software architecture paradigm. Integration was then achieved in real time through the Internet and loosely-coupled technologies.

The evolution has far from ceased. Many standard organizations and a large set of vendors are still improving it. We look at these organizations in the next section.

3.2.2 SOA Contributions

Even today, radical developments are changing the face of SOA to tune it to growing enterprise needs. A key to SOA’s widespread acclaim lies in its vendor-neutral standards. We first discuss the three most important standard organizations: World Wide Web Consortium (W3C), OASIS,
and Web Services Interoperability (WS-I). Subsequently, we will look at major vendors that contribute to SOA.

The W3C was formed in 1994 and has currently approximately 400 members worldwide. Their goal is to further advance the WWW by redefining fundamental standards that improve online business and information sharing. In the SOA domain, W3C’s role has been primarily as a standards body responsible for specifications that provide core and generic functionality.

The OASIS was founded in 1993, when it was still referred to as SGML Open. In 1998, the name was changed to OASIS. It as essentially a non-profit, international consortium driving the development, convergence, and adoption of e-business-related standards, with more than 5,000 participants from over 600 organizations and individual members in 100 countries. The goal of OASIS is to support specific industries and to foster trade and commerce between e-business-enabled enterprises by enforcing standards and specifications.

The WS-I, formed in 2002, currently has around 200 members. As a group dedicated to the interoperability concerns of disparate platforms, the WS-I does not produce technology standards. Instead, this organization provides profile documents that establish a tried and tested collection of standards. Compliance to these profiles guarantees organizations with environments that support a level of industry-standard interoperability.

An overview summarizing these three standard organizations and also naming its prominent deliverables related to SOA are given in Table 1. Most of the mentioned standards and specifications are described in chapter 5.

<table>
<thead>
<tr>
<th>W3C</th>
<th>OASIS</th>
<th>WS-I</th>
</tr>
</thead>
<tbody>
<tr>
<td>Established</td>
<td>1994</td>
<td>1993 as the SGML Open, 1998 as OASIS</td>
</tr>
<tr>
<td>Membership</td>
<td>400</td>
<td>600</td>
</tr>
<tr>
<td>Overall Goal</td>
<td>To further the evolution of the Web, by providing fundamental standards that improve online business and information sharing.</td>
<td>To promote online trade and commerce via specialized Web services standards.</td>
</tr>
</tbody>
</table>

Table 1: Standards organizations involved in current SOA developments (see Erl 2005).

Though standard organizations have their own culture and philosophies regarding the method of implementing their standards, they are all heavily influenced by the commercial market. Even though these organizations exist as independent entities, their membership includes most of the major software vendors. These vendors comprise a significant portion of the contributors who
actually develop the standards. Some of the companies that have participated in the standards development processes include Microsoft, IBM, BEA Systems, SAP, Sun Microsystems, Oracle, Tibco Systems, Hewlett-Packard, and Canon. Forming alliances allows vendors to join forces and attain common goals. Generally, the lifespan of an alliance is centered around the development cycle of a particular specification. Such collaborations result in suitable extensions and other futuristic technologies.

An individual or an organization does not own or control SOA. Since SOA has evolved from proprietary platforms into an architecture that promotes and supports open standards and vendor-neutral protocols, it is likely to remain an important architecture for as long as it has the support of major software vendors. This is because the benefits of SOA may only be realized as long as it continues to receive the global acceptance it does now.

3.3 SOA Reference Model

In August 2006, the OASIS published a committee specification called the “Reference Model for Service Oriented Architecture 1.0” (see MacKenzie et al. 2006). The reference model was intended to form a basis for the development of standards and specifications supporting SOAs. It belongs to a high level of abstraction viz. it is not directly tied to any standards, technologies, or concrete implementations. It helps to understand the relations among underlying entities and might also be used for training purposes or to define SOA. Another important objective of this reference model is to introduce a common vocabulary and understanding of SOA to be used amongst business analysts, technology specialists, researchers, and educators. The specification was approved as an OASIS standard in October 2006. In the remainder of this chapter, we evaluate the most important issues of this specification.

3.3.1 Definition of SOA

The SOA reference model defines this software architecture in the following way:

Service Oriented Architecture is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations. (see MacKenzie et al. 2006)

We now look at the concepts involved in the reference model. This particular set of concepts may be represented in the form of a map as shown in Figure 6. Principal concepts are depicted in dark blue and big bubbles. Light-blue bubbles represent minor concepts. A line between two concepts stands for a relationship between the two. The concept to which an arrow points may be interpreted as dependent in some way on the concept from which the line originates. The concept map in Figure 6 elucidates the dependencies between single concepts. These will be dealt with subsequently.

Capabilities or functionalities are created by people or organizations in order to solve a problem, or at least to support the solution of a problem. On the other hand, the needs of a person or an organization may be met by the capabilities of another entity. Applying this concept to the world of distributed computing, the requirements of one component may be met by capabilities of another component. These components may belong to different owners, but this
does not necessarily imply a one-to-one relationship. Sometimes a capability meets several needs or otherwise one need may utilize numerous capabilities in order to solve a problem. The granularity may vary from simple to complex. Hence, “the perceived value of SOA is that it provides a powerful framework for matching needs and capabilities and for combining capabilities to address those needs.” (see MacKenzie et al. 2006)

The three key concepts describing the SOA paradigm, depicted in the reference model, are Visibility, Interaction and Real World Effect:

- **Visibility** represents the “eyes” of the entities with needs and capabilities. Visibility enables the needy entity to “see” where their requirements may be fulfilled. Therefore, features of the capability must be portrayed in a way that is universally accessible and understandable. Visibility enables the matching of needs and capabilities both ways.

- **Interaction** is the process of using a capability. This is typically done by the exchange of messages. Executing an interaction leads to a real world effect.

- **Real world effects** are the results of interactions. These results may also represent a set or a series of effects. These effects may be classified as public and private actions. Private actions, as the name suggests, are unknown to other parties. Public actions may be invoked, usually resulting in change of the shared state. This shared state often persists between those involved in the current interaction. The expected real world effect of using
a capability is important as it forms the basis for deciding whether a capability meets a need or not.

The aforementioned concepts apply to services in the same way as they apply to SOA. The terms “loosely-coupled” and “coarse-grained” or even “Web services” are often used with regard to SOAs. The Reference Model for Service Oriented Architectures does not use these terms since they are too specific and without any metrics for an abstract discussion. Although Web services are usually involved in the implementation of SOAs, they are merely a means to concrete implementation; thus, Web services are too solution-specific to be part of a reference model (see MacKenzie et al. 2006).

### 3.3.2 SOA Services

This section gives a deeper insight into services described on an abstract and general level. The dynamics of services in terms of the aforementioned concepts of visibility, interaction, and real world effect as well as the description of services itself are being considered in this section. Services are defined in the following way, within the reference model:

A service is a mechanism to enable access to a set of one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description. A service is provided by one entity – the service provider – for use by others, but the eventual consumers of the service may not be known to the service provider and may demonstrate uses of the service beyond the scope originally conceived by the provider. (see MacKenzie et al. 2006)

Entities, such as people and organizations that offer capabilities, act as service providers. Those whose needs are addressed by these capabilities are known as service consumers.

From a consumer’s point of view, the most integral part of a service is the service interface. This is the public part required to determine if the service indeed serves the consumer. Typically, the rest of the service implementation is invisible to the consumer. Once the service has been invoked, it leads to certain consequences: the effect may be classified as the return of a response, the change of the shared state, or a combination of the both.

The three fundamental concepts introduced for SOA above also apply for services in a similar manner. A clear understanding of the essential nature of services is critical to our discussion. The definition of visibility requires that a consumer and a provider “see” each other. Preconditions for visibility are awareness, willingness, and reachability. The parties must be aware of each other, they must have an ambition, and they must be able to interact. Visibility and hence, interactions are made possible only if these prerequisites are fulfilled. In the service context, visibility may be realized through service descriptions. Such descriptions contain necessary information for interactions with the service, such as inputs, outputs, associated semantics, and conditions. The service description also contains information about what may be accomplished when that specific service is utilized. The description empowers service consumers with the ability to decide whether the service is suited to their needs. Furthermore, the service description makes apparent all requirements of the service provider that are being fulfilled by the consumer.

Interaction may be defined as performing actions that directly involve a service. Usually, this is done through the exchange of messages. Other ways of interaction include those that may be initiated through a change in the shared state. To enforce appropriate interactions, two
### Table 2: Terms used in OASIS’ Reference Model for SOAs (see MacKenzie et al. 2006).

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action Model</td>
<td>The characterization of the permissible actions that may be invoked against a service.</td>
</tr>
<tr>
<td>Awareness</td>
<td>A state whereby one party has knowledge of the existence of the other party. Awareness does not imply willingness or reachability.</td>
</tr>
<tr>
<td>Behavior Model</td>
<td>The characterization of (and responses to, and temporal dependencies between) the actions on a service.</td>
</tr>
<tr>
<td>Capability</td>
<td>A real-world effect that a service provider is able to provide to a service consumer.</td>
</tr>
<tr>
<td>Execution context</td>
<td>The set of technical and business elements that form a path between those with needs and those with capabilities and that permit service providers and consumers to interact.</td>
</tr>
<tr>
<td>Information model</td>
<td>The characterization of the information that is associated with the use of a service.</td>
</tr>
<tr>
<td>Interaction</td>
<td>The activity involved in making use of a capability offered, usually across an ownership boundary, in order to achieve a particular desired real-world effect.</td>
</tr>
<tr>
<td>Policy</td>
<td>A statement of obligations, constraints or other conditions of use of an owned entity as defined by a participant.</td>
</tr>
<tr>
<td>Process Model</td>
<td>The characterization of the temporal relationships between and temporal properties of actions and events associated with interacting with the service.</td>
</tr>
<tr>
<td>Reachability</td>
<td>The ability of a service consumer and service provider to interact. Reachability is an aspect of visibility.</td>
</tr>
<tr>
<td>Real world effect</td>
<td>The actual result of using a service, rather than merely the capability offered by a service provider.</td>
</tr>
<tr>
<td>Semantics</td>
<td>A conceptualization of the implied meaning of information, that requires words and/or symbols within a usage context.</td>
</tr>
<tr>
<td>Service</td>
<td>The means by which the needs of a consumer are brought together with the capabilities of a provider.</td>
</tr>
<tr>
<td>Service Consumer</td>
<td>An entity which seeks to satisfy a particular need through the use capabilities offered by means of a service.</td>
</tr>
<tr>
<td>Service description</td>
<td>The information needed in order to use, or consider using, a service.</td>
</tr>
<tr>
<td>Service Interface</td>
<td>The means by which the underlying capabilities of a service are accessed.</td>
</tr>
<tr>
<td>Service Provider</td>
<td>An entity (person or organization) that offers the use of capabilities by means of a service.</td>
</tr>
<tr>
<td>Shared state</td>
<td>The set of facts and commitments that manifest themselves to service participants as a result of interacting with a service.</td>
</tr>
<tr>
<td>Visibility</td>
<td>The capacity for those with needs and those with capabilities to be able to interact with each other.</td>
</tr>
<tr>
<td>Willingness</td>
<td>A predisposition of service providers and consumers to interact.</td>
</tr>
</tbody>
</table>

models are introduced in the reference model: the information model and the behavior model. The information model characterizes the information exchanged with a service with regard to
structure and semantics. Since knowledge of the structure, for instance form and data types, is not adequate to completely describe the interpretation of the data, semantics must also be included for a comprehensive data analysis. For instance, the ordering of products comprises two vital aspects: the intent to buy products and the information of the specific items included in the order. On the other hand, the behavior model describes the actions offered by a service and the process details pertaining to interaction with a service. The behavior model may hence be further divided in an action model and a process model. While the action model expresses the literal actions of a service, the process model characterizes the temporal aspects of an interaction. Temporal aspects are intimately related to the orchestration of services that set up a flow of executions viz. to realize complex business processes. Although orchestration is an important aspect in SOAs, the reference model is not absolute with respect to the process model in the current version.

The real world effect, as the name implies, is the result of one interaction or even a complete set of interactions with a service in the real world. For instance, a flight ticket is booked with the purpose to secure a seat in the right airplane. The consumer of the service anticipates the effect of using the service and so does the provider. Internal actions are private and by definition, both consumer and provider are oblivious to them. The involved parties share a certain information during the interaction, referred to as the shared state. The modifications of the shared state is effected by the actions accomplished by service consumers and service providers (see MacKenzie et al. 2006).

There are other concepts to support this view on dynamic aspects of services: service descriptions, policies, and contracts related to services and execution contexts.

The necessary information for using a service is represented in its service description with the purpose to facilitate the necessary interaction and visibility. This is of value, particularly when the involved parties reside in domains of different ownership. Therefore, any service description must use a standard and widely accepted format, containing information on:

- availability and reachability of the service,
- functions performed by the service,
- constraints and policies regarding the service,
- compliances of policies, and
- interactions with the service in order to achieve the desired objectives.

These represent a set of critical information which helps a service consumer evaluate the suitability of a specific service. The service reachability provides the consumer with decisive metadata, like the location of the service or the access protocol to the service. In addition, it may include a status on the current availability of that service. Service functionality contains the functions and the real world effects that represent results after service invocation. Its formulation must be easily comprehensible to every potential consumer. A service description also accounts for policies for prospective consumers to decide whether the service conforms to the consumer’s constraints. For effective interaction with services, certain information is indispensable; this is directly addressed by the existence of the information model. We have already discussed the information model above with respect to dynamic aspects of services. We will now turn our focus to two issues pertaining to the SOA reference model viz. policies and contracts related to services as well as the execution context of services.
A policy outlines constraints or conditions on the usage, the deployment, and the description of an entity, in this case, of a service. The policy definition is determined by the owner of the entity. A policy assertion mostly concerns the realization of a service. The policy conductor takes care of the compliance, thus policy enforcement becomes imperative. An unenforceable constraint may be considered as a “wish”. The service description is a mutual point of contact binding service participants and the service policy. A contract, on the other hand, represents an agreement between two or more parties. The contract may be arrived at by methods, that may not be part of an SOA or could be arrived at during a service interaction, that may be part of the SOA.

The execution context with regard to service interactions paves the path between those with needs and those with capabilities. The application of the concepts of visibility, policies, contracts and interaction, to the actual and individual situation involving a specific service, is described by the execution context. This helps distinguish equal services. From a provider’s point of view, the execution context grants individuality to each interaction (see MacKenzie et al. 2006).

Till now, we have discussed SOA in an abstract manner without any association to a specific instance. Used definitions of terms related to SOA, indicated in the above text using italic letters the first time they occur, are summarized in Table 2. It represents an extract from MacKenzie et al. (2006).

### 3.4 SOA Stack

The various concepts involved in SOA implementation give rise to its high level layered architecture shown in Figure 7. The composition of layers, collectively referred to as a “stack”, delineates the underlying techniques relevant to each layer. The lowermost layer, the transport layer, contains mechanisms required to move service requests from the service consumer to the service provider and to transport responses back. The next layer consists of the messaging layer wherein the message framework is defined as agreed upon by the service consumer and service provider. The next logical step involves the description of the service, providing an insight into the nature of the service, the manner in which it may be invoked and its subsequent output. This standardized service description is held in a registry which serves as a beacon for future discovery. The layer is thus referred to as the discovery layer. This is followed by the business process layer which composes several core services to complete business processes rendering them available as composite services. All layers above the transport layer should preferably share a common conceptual basis to enhance the collective usability of the schema. Issues pertaining to the quality of service vertically transgress into each layer. This ensures effective implementations of policy, security, reliability, transaction, and management aspects in each layer.

Standards and specifications, that may be used to fulfill each layer’s objective will be presented in greater detail in chapter 5.

### 3.5 Benefits of Using SOA

The use of SOA has several benefits which help an organization successfully adapt to the dynamic business landscape today. One rudimentary goal of an enterprise is to be resilient to changing business conditions. It is also necessary to maintain a competitive advantage with regard to cost effectiveness. The following listing, discusses both business and technical benefits. This list
3.5 Benefits of Using SOA

Figure 7: The layered SOA stack.

is not exhaustive and does not imply that previous methods did not contribute to any of the mentioned benefits.

- **Leverage existing structures:** Existing IT investments may be used in conjunction with SOA, because SOA provides a layer of abstraction that wraps these existing assets as services that deliver business functions. Organizations may then continue to receive value out of existing resources instead of rebuilding from scratch. Thus, the cost and effort of integrating legacy and contemporary solutions is lowered.

- **Reusability:** Services are built to fulfill immediate application-level requirements while still supporting a degree of reprocessing by future potential requesters. This decreases the development effort and thus reduces costs.

- **Flexibility, fast time-to-market:** With core business services exposed in a loosely-coupled manner, the services may be used easily and combined based on business needs. This also allows flexible and efficient rewiring once the business requirements change. The usage of SOA also leads to rapid development of new business services. Furthermore, it allows an organization to adapt quickly and reduce time-to-market.

- **Manageability:** Integration is performed through the specification of a service, not with its implementation. Hence, integration of various services spanning boundaries becomes more manageable since their complexities are isolated.

- **Transparency:** SOA is not bound to a certain set of technologies, yet it fosters the application of several public standards thus increasing transparency. The development of new standards and the improvement of existing standards is backed by an emerging industry.

- **Business and IT alignment:** SOA provides business executives with a clear understanding of what IT does and its value. A closer association of IT and business services may justify IT investments with greater ease.

The SOA boom has begun and giant business software vendors are competing for parts of this new market. Today, SOA features as a top priority for major software organizations. “Suddenly, the major enterprise vendors are all deciding that they will own SOA and make it the foundation of their own development and partner ecosystem.” (see Hurwitz 2005) According to a prediction of Gartner, “by 2011, extensible and pluggable modularity will be the prevailing internal architecture of platform technologies from application servers and portals to integration suites,
business process management suites and APSs (application platform suites) (0.8 probability).” (see Thompson et al. 2005)
Part II

Key Technologies Realizing SOA
4 Process Modeling Notations

There exist a large number of technologies that utilize applications in the domain of SOA. In the last few years, in particular, there has been extensive research and development to further the extent of standardization. In this chapter, we will describe usual process modeling notations with a focus on business processes. Subsequently, we look at several Web services technologies in the order of the previously described SOA stack, are presented in the chapter 5. Since the composition of core functionalities into complete processes is a fundamental concept within SOA, its current service composition standard are discussed in chapter 6.

To describe and model processes, notations, as a set of symbols, are required for better communication amongst people. In business processes, during the interaction between business analysts and software developers, the use of special notations becomes necessary. As a consequence, many notations exist that vary in their domains, stakeholders, expressiveness, and levels of formalization. The domains range from business processes to workflows to software processes; the set of stakeholders includes business analysts, system architects, and software developers. Regarding the level of formalization, notations may be classified into informal, semi-formal, and formal models as, informal textual descriptions, semi-formal graphical visualizations, and formal representations to support code generation.

Early process visualization methods contain Entity Relationship (ER) diagrams, organizational charts, function trees, and flow charts. With an ER model, a part of the real world can be described through a diagram composed of entities, roughly representing objects, and relationships defining the semantic connection between several entities. Lines connecting these entities and relationships are usually equipped with cardinalities representing the amount of participating entities in one relationship. Both entities and relationships posses several attributes. Since the nature of ER models does not allow the description complex processes, it is, however, suitable for defining data structures. It is thus mainly utilized for database design and occasionally used for business process visualization. The next process visualization method, is an organizational chart. This method only depicts the structure of an organization, usually representing hierarchies within a company. It contains relationships between objects and data flows, but there is no way to represent sequential and parallel activities. It is, therefore, not appropriate for use in the description of processes. Although function trees are the simplest of all methods to model processes, it is also the most limited. It is a diagram depicting functional dependences within a system. It shows how a task may be repeatedly divided into several subtasks and thus, following it in reverse, leads to processing the main task. Another schematic representation of a process is given through flowcharts which are commonly used in business environments. A flowchart, which treats a process as a succession of activities, typically begins with one start symbol and completes with several end symbols, usually depicted as ovals. Arrows between processing steps are represented as rectangles indicating the flow of control. Inputs and outputs are drawn as parallelograms, whereas, conditions are portrayed as diamonds.

Because of their simplicity and lack of standardization, the mentioned models have been replaced by more precise notations, including Petri nets, Unified Modeling Language (UML) activity
diagrams, Event-Driven Process Chains (EPC), Yet Another Workflow Language (YAWL), and Business Process Modeling Notation (BPMN). The selection of an appropriate process modeling notation of a given modeling task is left at the user’s discretion. Petri nets are the most formal, computational way to describe processes and UML activity diagrams depict an object-oriented programming model. For these reasons, both directly address the software developer. On the other hand, EPC, YAWL, and BPMN are designed to narrow the disparity between enterprises and information technology. Out of all these, BPMN is presently the most promising and popular notation amongst leading vendors, however, all of the other four mentioned standards will also be discussed in the following sections.

### 4.1 Petri Nets

Petri networks represent a formal, mathematical model for discrete distributed systems. With this graphical modeling language, systems and transformation processes may be designed. Petri nets were defined by the mathematician Carl Adam Petri in his PhD thesis in 1962.

A Petri net is a bipartite, directed graph composed out of Places and Transitions, which are connected through directed Arcs. Places represent a stopping point in a process, usually the attainment of a milestone and are drawn as circles. A Transition is a rectangle symbolizing an event or an action. Arcs are arrows linking a Transition with a Place or vice versa. Direct connections between two Places or two Transitions are not permitted. Arcs may possibly have weights or costs assigned. In order to model the dynamic behavior of Petri nets, the concept of Tokens was introduced. Therefore, each Place has a certain capacity and may hold a certain number of tokens. They are represented as black dots residing in a Place, never in a Transition.

![A simple Petri net model showing basic structural elements.](image)

**Figure 8:** A simple Petri net model showing basic structural elements.
and are moved through the network upon its execution. Collectively, the set of tokens in a Petri net show the current state of a process. The initial distribution of tokens in a Petri net is referred to as a *Marking*. A path, modeled as a Petri net, always starts and ends in a Place. Branching is possible in a many-to-many way allowing one Place/Transition to split into multiple Transitions/Places, which again may join into a single Place/Transition. See Figure 8 for a simple Petri net portraying the described structural elements.

The execution of a Petri net requires tracing the movement of tokens through the net guided by the following rules. A Transition is enabled if each input Place, a Place that links to the Transition, has at least one token. The result is that a Transition is able to act on these input tokens, which is referred to as a *Firing*. When a Transition fires, it consumes one token from each input Place and generates a token for each of its output Places which are Places that the Transition links to. By combining the basic elements and following the described execution rules, parallel and sequential processing using several control structures may be modeled.

To facilitate the modeling of large and complex processes, the classical Petri net, as described above, was extended with additional constructs. The problem of indistinguishable tokens in classical models was solved by adding colored tokens representing tokens with different attributes. This extension introduces features like conditional branching and guards. Furthermore, since some processes could not be modeled with a single net, an hierarchy extension was introduced. Nets may unitized and one net may then initiate subnets enabling a Transition. Another construct adds time to Petri nets. A time stamp for each token indicates the point in time from which the token will be available. The complex nature of Petri nets led to the evolution of relatively simpler notations which we will discuss ahead.

### 4.2 UML Activity Diagram

Contrary to the formal method adopted by Petri nets, activity diagrams adopt a semi-formal notation for process modeling. It mainly targets software engineers and architects who design operational workflows of systems. An activity diagram may be seen as a flowcharting technique as well as a special type of a state diagram. Activity diagrams are standardized within UML. Its current version 2.0 (see OMG 2005) uses Petri net token semantics to model activities.

A state diagram is a visualization of the behavior of a single entity. It shows a set of event-triggered transitions that moves the entity from state to state. Such state machines define object behavior precisely, however, they lack features required for a business process notation. Activity diagrams deal with the flow of control and data amongst actions involved in a process.

The functional elements of a UML 2.0 Activity Diagram are *Activity Nodes*. There are three kinds of nodes: *Action Nodes*, *Object Nodes*, and *Control Nodes*. Action nodes, which are atomic processing units or activities, are drawn as round-cornered rectangles (see Figure 9 for graphical representations) with attached pre- and post-conditions. Object nodes represent objects and data. Control nodes are used to structure *Transitions*, *Control Flow*, and *Object Flow*. For alternate flow modeling, *Decision* and *Merge Nodes*, shown as diamonds, are utilized. A Decision node ensures that the guards select exactly one path. Merge nodes then simply forward all incoming tokens to its outgoing edge. Each activity has at least one input and one output path, which are transitions drawn as arrows from the previous or to the following activity. Tokens, representing both control and data objects, are used to pass control from one activity to another. Activities accept tokens from their input edges and generate tokens at their output edges. The *Start State* of a complete process is depicted as a filled circle; *Stop States* are
4.3 Event-Driven Process Chain (EPC)

drawn as filled circle with an outer circle. **Fork Nodes** are used for concurrent execution whereas **Join Nodes** are used for merging concurrent branches. Fork nodes copy incoming tokens and pass them to all outgoing edges; join nodes offer tokens to the outgoing edges based on certain conditions. A rounded border around the whole model is used to frame a complete process and depict its activity parameters as object nodes. Furthermore, a **Swimlane** is a way to group activities performed by the same actor in an activity diagram or to group activities in a single thread.

Figure 9: A simple UML 2.0 Activity Diagram showing basic structural elements.

The presented elements, though sufficient, have their limitations in the modeling of complex processes. The addition of advanced elements and constructs may be used to improve the modeling capabilities.

4.3 Event-Driven Process Chain (EPC)

**EPCs** represent another kind of a semi-formal flowchart approach for process modeling, which enjoyed better success in the business world by being used for modeling, analyzing, and redesigning of business processes. Its syntax and semantics differ from the UML Activity Diagram, they are both, however, comparable in their expressive power. The EPC notation was developed in a team led by Prof. Scheer supported by SAP and was published in January 1992 (see Keller et al. 1992).

In this modeling language, two basic objects are defined which are connected through directional lines in a one-to-one relation. Objects alternate between **Events** and **Functions** forming an ordered graph. Events are passive elements that describe under which circumstances under which a function may be triggered or may represent the results of the previous function. In the notation, events are depicted as hexagons (see Figure 10). Functions, on the other hand, are the active elements in EPCs responsible for modeling the activities within a process. These represent the transformation from an initial to a resulting state and are always commenced by an event.
Functions are drawn as rounded rectangles (see Figure 10). Functions alone may have cost and time associated with them. Between events and functions, various Control Flow connectors may be used allowing alternative and parallel execution of processes. Also, Logical Connectors, like conjunction (AND), disjunction (OR), and exclusive disjunction (XOR) may be applied. Minor elements of the EPC notation include the Information Object and the Organization Unit. Information or resource objects, drawn as rounded rectangles, portray objects in the real world which may act as input data serving as the basis for a function or output data produced by a function. The connection between the two is undirected and referred to as Information flow. Organization units, depicted as ellipses, determine which person or organization within the structure of an enterprise is responsible for a specific function. The link between a function and an organization unit is referred to as Organization Unit Assignment.

![Figure 10: A simple EPC showing its basic structural elements.](image)

As a result, the EPC graph contains information about any event that initiates a function or any event created by a function. An event created by a function may act as the trigger or activator for another function, thus setting up a cascade or a chain of events. Each chain begins with at least one initial event and also ends with at least one final event. To structure large EPC graphs representing complex processes, an hierarchical arrangement using subgraphs may be employed. Such subgraphs represent a special event in the model, for which purpose, the Process Path element is available.

In order to include relationships involving this business process and other systems, additional
elements had to be inculcated into the classical EPC model. This made the presentation of the information regarding supporting systems, utilized data, and generated output possible. This also allows for other information objects, such as databases, that influence certain functions or may be influenced by them, to be included.

A major strength of EPC is its simplicity and its capability to model important features of a business process at the same time. It is mainly used in SAP and Architecture of Integrated Information Systems (ARIS) products, suggesting its suitability for ERP and business analysis.

4.4 Yet Another Workflow Language (YAWL)

YAWL is another process modeling notation designed on the basis of Petri nets. It was originally developed by researchers at Eindhoven University of Technology and Queensland University of Technology in 2002. The team led by Prof. van der Aalst analyzed existing workflow management systems and workflow languages and formed workflow patterns necessary for business processes (see van der Aalst et al. 2003). Based on these results, YAWL was developed to comprehensively support all of the twenty workflow patterns (see van der Aalst a. ter Hofstede 2005).

The developers took Petri nets as a starting point. To overcome the limitations of Petri nets, YAWL has been extended with features to facilitate patterns involving multiple instances, advanced synchronization patterns, and cancellation patterns. Moreover, YAWL allows hierarchical decomposition and handles arbitrarily complex data. Figure 11 shows the most important modeling elements of YAWL. The most outstanding aspect of YAWL is its ability to model OR-joins, composite tasks, directly connected transitions, and the removal of tokens. Though based on Petri nets, YAWL represents a completely new language with independent semantics.

![Figure 11: A simple YAWL example showing its basic structural elements.](image)
4.5 Business Process Modeling Notation (BPMN)

As mentioned before, BPMN, as a process modeling notation, enjoys significant support in the industry. Currently, it has been implemented by more than 35 companies, including BEA Systems, IBM, IDS Scheer, Infosys, Intalio, Oracle, and SAP.

An initial draft of BPMN was released to the public in November 2002 by the Business Process Management Initiative (BPMI.org). Later, in May 2004, the first version of a BPMN specification was released. Because of the overlap in the work of the BPMI.org and the Object Management Group (OMG), they announced the merger of all BPM activities in June 2005. They combined forces to form a group called the “Business Modeling and Integration Domain Task Force”. Hence, a final adopted specification of BPMN version 1.0 was released in February 2006 (see BPMI.org a. OMG 2006).

BPMN is a graphical flowchart-like language to model business processes, especially developed to fulfill the needs of business environments. Though this specification addresses business analysts, it also narrows the gap between them and software engineers. This is obtained through the provision of a formal mapping from BPMN to Business Process Execution Language (BPEL) (see section 6.2), a strong process execution language widely-used during the technology implementation phase. This mapping finally enables the execution of BPMN. This issue is often referred to as the “BPMI.org Hourglass” (see Figure 12. Like YAWL and other process modeling languages, BPMN uses the token concept throughout and thus is also based on the powerful notation of Petri nets.

Within the BPMN end-to-end model, three basic types of sub-models exist (see BPMI.org a. OMG 2006):

- private or internal business processes,
- abstract or public processes, and

---

BPMN is a graphical flowchart-like language to model business processes, especially developed to fulfill the needs of business environments. Though this specification addresses business analysts, it also narrows the gap between them and software engineers. This is obtained through the provision of a formal mapping from BPMN to Business Process Execution Language (BPEL) (see section 6.2), a strong process execution language widely-used during the technology implementation phase. This mapping finally enables the execution of BPMN. This issue is often referred to as the “BPMI.org Hourglass” (see Figure 12. Like YAWL and other process modeling languages, BPMN uses the token concept throughout and thus is also based on the powerful notation of Petri nets.

Within the BPMN end-to-end model, three basic types of sub-models exist (see BPMI.org a. OMG 2006):

- private or internal business processes,
- abstract or public processes, and

---

1 see http://www.bpmi.org, last checked 24/11/2006
2 see http://bmi.omg.org, last checked 24/11/2006
4.5 Business Process Modeling Notation (BPMN)

- collaboration, or global processes.

Private business processes are those that are native and limited within the purview of a specific organization. Generally, this type was known as workflow processes. On the other hand, public business processes only represent interactions between a private business process and others processes instead of showing internal details. Only activities that communicate play a role in this sub-model. Far more universal, a collaboration model, depicts all interactions between multiple organizations, focusing on message exchange. Again, only communicating activities are visible. Each of these three models is represented in a specific format within BPMN.

A graph created on the basis of the BPMN specification is usually referred to as a Business Process Diagram (BPD). It is built on a set of essential language constructs which are further categorized into four groups: Flow Objects, Connecting Objects, Swimlanes, and Artifacts.

Flow Objects represent the main graphical elements that define the behavior of a business process. These act as nodes in the BPD. The core elements that belong to this category are Events, Activities, and Gateways. See Table 3 for more detailed descriptions and the notation of each of these Flow Object types.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>An Event is something that “happens” during the course of a business process. These Events affect the flow of the process and usually have a cause and a result. Events are circles with open centers to allow internal markers to differentiate different triggers or results. Events are categorized by the stage at which they occur in the process: Start, Intermediate, and End (see the figures to the right, respectively).</td>
<td><img src="image" alt="Event" /></td>
</tr>
<tr>
<td>Activity</td>
<td>An Activity is a generic term for work that company performs. An Activity may be atomic or non-atomic, thus compound. The types of activities that are a part of a process model are: Process, Sub-Process, and Task. Tasks and Sub-Processes are rounded rectangles. Processes are either unbounded or a contained within a Pool.</td>
<td><img src="image" alt="Activity" /></td>
</tr>
<tr>
<td>Gateway</td>
<td>A Gateway is used to control the divergence and convergence of Sequence Flow. Thus, it will determine branching, forking, merging, and joining of paths. Internal Markers will indicate the type of behavior control.</td>
<td><img src="image" alt="Gateway" /></td>
</tr>
</tbody>
</table>

Table 3: BPMN core modeling elements: Flow Objects (see BPMI.org a. OMG 2006).

Flow Objects are linked to each other or to other information objects through Connecting Objects. There are three basic types of such objects: Sequence Flows, Message Flows, and Associations. See Table 4 for more detailed descriptions and the notation of each of these types.
### 4 Process Modeling Notations

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence Flow</td>
<td>A Sequence Flow is represented by a solid line with a solid arrowhead. It is used to show the order that activities will be performed in a process.</td>
<td><img src="image" alt="Sequence Flow" /></td>
</tr>
<tr>
<td>Message Flow</td>
<td>A Message Flow is represented by a dashed line with an open arrowhead. It is used to show the flow of messages between two participants that are prepared to send and receive them. In BPMN, two separate Pools in the diagram will represent the two participants (e.g., business entities or business roles).</td>
<td><img src="image" alt="Message Flow" /></td>
</tr>
<tr>
<td>Association</td>
<td>An Association is represented by a dotted line with a line arrowhead. It is used to associate data, text, and other Artifacts with Flow Objects. Associations are used to show the inputs and outputs of Activities.</td>
<td><img src="image" alt="Association" /></td>
</tr>
</tbody>
</table>

Table 4: BPMN core modeling elements: Connecting Objects (see BPMI.org a. OMG 2006).

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pool</td>
<td>A Pool represents a participant in a process. It is also acts as a graphical container for partitioning a set of Activities from other Pools, usually in the context of B2B situations.</td>
<td><img src="image" alt="Pool" /></td>
</tr>
<tr>
<td>Lane</td>
<td>A Lane is a sub-partition within a Pool and will extend the entire length of the Pool, either vertically or horizontally. Lanes are used to organize and categorize Activities.</td>
<td><img src="image" alt="Lane" /></td>
</tr>
</tbody>
</table>

Table 5: BPMN core modeling elements: Swimlanes (see BPMI.org a. OMG 2006).

The primary modeling elements, Flow and Connecting Objects, may be organized into separate visual categories in order to illustrate different functional capabilities and responsibilities. BPMN supports this through Swimlanes which are available as two main constructs: Pools and Lanes. See Table 5 for more detailed descriptions and the notation of each of these two types.

BPMN was designed to allow modelers and modeling tools some flexibility in extending the basic notation and in the provision of additional contexts, suitable to a specific modeling situation. For such purposes, Artifacts may be used. Any number of Artifacts may be added to a diagram. The current version of BPMN predefines only three types of BPD Artifacts, which are Data Objects, Groups, and Annotations. See Table 6 for more detailed descriptions and the notation of each of these three types.

Overall, BPMN represents a useful, descriptive means of modeling business processes despite a few restrictions to this notation. For example, Message Flows may only be modeled amongst different Pools. Also, the two levels of Swimlanes might be too limited for larger organizations. On the other hand, the assumptions that the model is based on may be too simple for effective
<table>
<thead>
<tr>
<th><strong>Element</strong></th>
<th><strong>Description</strong></th>
<th><strong>Notation</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Object</td>
<td>Data objects are considered Artifacts because they do not have any direct effect on the Sequence Flow or Message Flow of the process, but they do provide information about what activities require to be performed and/or what they produce.</td>
<td><img src="image" alt="Name" /></td>
</tr>
<tr>
<td>Group</td>
<td>A grouping of activities that does not affect the Sequence Flow. The grouping may be used for documentation or analysis purposes. Groups may also be used to identify the activities of a distributed transaction that is shown across Pools.</td>
<td><img src="image" alt="Group" /></td>
</tr>
<tr>
<td>Annotation</td>
<td>Text Annotations are a mechanism for a modeler to provide additional information for the reader of a BPMN diagram.</td>
<td><img src="image" alt="Annotation" /></td>
</tr>
</tbody>
</table>

Table 6: BPMN core modeling elements: Artifacts (see BPMI.org a. OMG 2006).

handling of Data Objects. These problems are now being addressed in recent improvisations.
5 Web Services Technologies

As we discussed in the previous chapter, SOA represents an abstract architectural concept that allows systems to be built using loosely-coupled components. These components must be described, discovered, and composed in a universal way. Web services and ancillary technologies, initiated in 2000, became the main approach towards realizing SOA. Although there are other means, this particular approach had the enthusiastic support of the IT industry.

Earlier, companies used ad hoc methods to take advantage of the basic Internet infrastructure. With the advent of Web services, a systematic and extensible framework for A2A communication and interoperability came into existence. This allowed compatibility across varied environments. Web service technology is widely accepted by the IT industry since it is built on existing Web protocols and based on open XML standards. Due to the “open” nature of its standardization and development, Web service technologies possess an inherent advantage over traditional solutions like Common Object Request Broker Architecture (CORBA)\(^1\). Industry-wide organizations, such as W3C, OASIS, and OMG, have always addressed this model adequately.

Within the W3C, the “W3C Web Services Architecture (WSA) Working Group”\(^2\) was formed in 2002 with the goal to lay out a coherent architecture by producing architectural documents and advising W3C regarding work in the Web services area. It defines Web services as follows:

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards. (see Booth et al. 2004)

In conformity with the achievements of the “W3C WSA Working Group”, we will now describe the process of engaging a Web service (see Booth et al. 2004). For each process, there is at least one pair of a provider entity and a requester entity needed. The purpose of the provider is to supply a functionality. While the Web service represents an abstract notion of this functionality, the concrete implementation that sends and receives messages is its provider agent. This agent must be provided. A requester is an entity that wishes to make use of the provider’s Web service. It will use a requester agent to exchange messages with the provider’s agent. The modalities of this message exchange are documented in a Web service description, which defines message formats, data types, transport protocols, and transport serialization formats. Also, the semantics of a Web service as the shared expectation regarding the behavior of the service must

\(^1\)CORBA is a standard for a software component model, created and controlled by the OMG. It defines Application Programming Interfaces (API), communication protocol, and object/service information models to enable heterogeneous applications written in various languages running on various platforms to interoperate. CORBA therefore provides platform and location transparency for sharing well-defined objects across a distributed computing platform. The first developments on CORBA began in October 1991. It is currently available in version 3.0.2 published in December 2002.

be agreed upon by both sides. In most situations, a *discovery service* is used by the requester to locate suitable Web service candidates.

![Diagram](image)

**Figure 13:** General process of engaging a Web service.

We now proceed to discuss the general process of engaging a Web service shown in the following listing and in Figure 13. Depending on the scenario, additional tasks may be inculcated into the presented steps. These steps are fundamental to engaging a Web service, although their exact order may not be predetermined (see Booth et al. 2004).

1. The requester and the provider must become known to each other or at least the initiator must become aware of the other party. Only in rare cases, the provider initiates the process within some “push” or subscription scenarios. In most cases, the process will be initiated by the requester. The requester entity may then obtain the provider agent’s address directly from the provider entity or the requester entity may use a discovery service to locate the suitable service. The following steps are required for working with discovery services:

   a) The provider may publish both the Web service description and an associated functional description of its service within a registry, an index, or a peer-to-peer system that may possibly be used by requester entities.

   b) The requester entity supplies criteria to the discovery service to select a Web service description based on its associated functional description.

   c) The discovery service (see subsection 5.5) returns one or more Web service descriptions (see subsection 5.4) that meet the specified criteria.
2. The requester entity and provider entity agree on the service description and semantics that will govern the interaction between the requester agent and the provider agent. This may be achieved through direct communication, take-it-or-leave-it contracts, or standardization by an organization.

3. Both the service description and the semantics must be input to, or implemented in, both the requester agent and the provider agent before the two agents are able to interact.

4. The requester agent and provider agent exchange messages on behalf of their owners (see subsection 5.3).

During the past few years, many Web service standards were proposed by standard organizations and groups of individual vendors. To get an overview, it is important to structure these standards into individual domains. Figure 14 shows the SOA stack with the current standards and specifications that will be further elaborated in the following subsections.

![Web service technologies in the overall SOA stack.](image)

**Figure 14: Web service technologies in the overall SOA stack.**

### 5.1 XML as the Base Technology

Since all Web services technologies are built heavily on a core set of Extensible Markup Language (XML) specifications, the problem of finding the underlying key technology requirement has been solved. The XML metalanguage provides a flexible data format which is standardized, platform and programming language independent, and extensible. XML data is self-descriptive and represented in Unicode, which makes it possible to store characters from every language. For Web services, it provides a clear separation between the definition of the service and its execution. Since the developer may then focus on data types and structures without considering the implementation, XML also eases system development and integration. For these reasons, the Web service domain employs XML for purposes such as description, discovery, data transfer, and security.
The core concept of XML is based on tags within angled brackets. Elements, attributes, descriptions, and a document have to be defined for every new XML language (see Cowan a. Tobin 2004; Bray et al. 2004). An element is a name construct containing a set of attributes and some “children”. Attributes themselves, represent a pair of a name and a value. It is necessary for one document to contain exactly one structured element. The basic XML structure is depicted in an abstract example in Listing 1.

```xml
<?xml version="1.0"?>
<documentElement>
  <element1>Some Description</element1>
  <element2 attribute1="attrValue">
    <subElement1 attribute2="attrValue">Some Description</subElement1>
  </element2>
</documentElement>
```

Listing 1: Structure within XML documents.

Using these concepts, new languages may be defined through a set of elements, determined by their structure, attributes, and a set of permissible attribute values and element contents. The document structure of that new language must conform to the rules described in a Document Type Definition (DTD) or XML Schema. While DTD was used in the early days of XML, XML Schema has superseded this objective in a more flexible manner. All Web services technologies, which we will discuss in the following sections, have associated schemas written in XML Schema.

Another base technology in XML is “Namespaces in XML”, with which it is possible to combine various XML vocabularies unaffected by overlapping elements. Each element and attribute name is then scoped to name spaces making them unique. Extensible Stylesheet Language (XSL) is used to define the format of an XML document. It includes two parts: XSL Formatting Objects (XSL-FO) and XSL Transformations (XSLT). XSL-FOs are XML applications used to describe the layout of for example a Web page in terms of blocks, graphics, and horizontal lines. Usually, XSL-FOs are not created directly, but instead XSLT style sheets define a mapping from XML to XSL-FO. In addition to these XML standards, XML Path Language (XPath) is a widely-used addressing mechanism to identify elements and attributes within XML documents.

### 5.2 Transport Layer

Message transport technologies form the foundation of a WSA. In this layer, service requests must be moved from the service consumer to the service provider and service responses must be transported back. To transfer these Web service messages, ubiquitous Web protocols such as Hypertext Transfer Protocol (HTTP) or Secure Hypertext Transfer Protocol (HTTPS), Java Messaging Service (JMS), Simple Mail Transfer Protocol (SMTP), or Web Services Reliable Messaging Protocol (WS-ReliableMessaging) may be used.

HTTP is a widely adopted, open W3C standard. It may be used on many different system types and does not conflict with protocol firewalls. It is, however, stateless and less reliable. HTTPS uses the same syntax, but adds an additional encryption and authentication layer before HTTP. The use of message middleware to exchange messages amongst Java applications is eased through JMS. This is more reliable and makes asynchronous requests possible, yet it is not usually adopted. The protocol that ensures reliability of electronic mail transfer is known
as SMTP. Finally, WS-ReliableMessaging is a framework for interoperability between different reliable transport infrastructures (see section 5.7.2). Of all the used transport protocols for Web services, HTTP outweighs all by far.

## 5.3 Messaging Layer

The messaging layer comprises the underlying communications mechanism for Web services upon which more abstract layers may be built. The input and output, to and from the conventional Web services are shuttled using the previously described protocols. To integrate widely distributed Web services, a standardized message framework must be defined. This ensures that all services use the same message format.

Today's technologies and standards within this layer are SOAP and Web Services Addressing (WS-Addressing). SOAP is a message format with which information may be wrapped to be exchanged between Web services. WS-Addressing gives SOAP a set of additional headers to route these messages. We look at both standards in the following subsections in further detail.

### 5.3.1 SOAP

The development of SOAP was started by Microsoft in 1999. Microsoft then collaborated with IBM, DevelopMentor, UserLand Software, and Lotus to develop a revised specification, SOAP version 1.1 which was published in April 2000. One month later, it was submitted to the W3C XML Protocol Working Group\(^3\). At the time, the meaning of the acronym “SOAP” was ambiguous. In version 1.1, it was referred to as “Simple Object Access Protocol” but also widely interpreted as “Service Oriented Architecture Protocol”. With the publication of the SOAP version 1.2 recommendation in June 2003, “SOAP” was declared a proper name and was no longer treated as an acronym. This recommendation consisted of a primer (see Mitra 2003), a definition of an XML-based messaging framework (see Gudgin et al. 2003a), and optional components (see Gudgin et al. 2003b). In the aforementioned documents, the W3C defines SOAP version 1.2 in the following way:

SOAP Version 1.2 (SOAP) is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that may be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics. (see Gudgin et al. 2003a)

The format focuses on flexibility, extensibility, and simplicity. Using SOAP to transfer information between a sender and a receiver, the Web services may be accessed through a loosely-coupled infrastructure no matter on which implementation technology and network protocol the transmission relies on. This specification delivers four core capabilities, which will be addressed in detail in the following paragraphs:

1. a standardized message structure based on XML,
2. a processing model about how a service should process the message,
3. a mechanism to bind SOAP messages to different network transport protocols, and

4. a way to attach non-XML encoded content to SOAP messages.

A SOAP message is an XML document that is constructed out of three distinct elements. The root element of the XML document is an envelope `<env:Envelope>`, which contains an optional header `<env:Header>` and one mandatory body `<env:Body>` (see Listing 2). The header contains meta information which paves a way to information in SOAP messages that are not business message payload. In this way, SOAP may be easily extended in an application-specific manner. Within each `<env:Header>` tag, several header blocks may be defined. Header blocks contain rules that define the message routing along a message path. The mandatory body element holds the primary content of the message viz. the business information, the syntax for which is not defined within SOAP. A few sub-elements like fault information `<env:Fault>` are placed within the body tag as well.

```xml
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <p:headerBlock1
      xmlns:p="http://example.com"
      env:role="http://example.com/Log"
      env:mustUnderstand="true">
      ...
    </p:headerBlock1>
    <q:headerBlock2
      xmlns:q="http://example.com"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      env:relay="true">
      ...
    </q:headerBlock2>
    <r:headerBlock3
      xmlns:r="http://example.com">
      ...
    </r:headerBlock3>
  </env:Header>
  <env:Body>
    ...
  </env:Body>
</env:Envelope>
```

Listing 2: The basic structure of a SOAP message with sample attributes.

SOAP also extends the processing model. It describes the actions that may be taken by a SOAP node on receiving a SOAP message. A SOAP node is a software agent that transmits, receives, processes, and/or relays a SOAP message. There are two basic node types: a sender and a receiver. A particular sender node acts as the original sender which performs the initial transmission of a message. Nodes that both receive and transmit messages are called intermediaries. The final destination of a message is the ultimate receiver which processes the message. A collection of nodes is called a message path (see Figure 15). Between two nodes, different message exchange patterns are possible; the simplest one is a one-way message. Synchronous and asynchronous request/response communications and even long-running conversations where the message exchange continues for an extended period of time may be modeled using SOAP. To summarize, the SOAP processing model describes a three-stage handling of SOAP messages. First, the node should check the received message for its syntactical conformity. Second, the message header should be processed depending on the assumed role of the SOAP node. The role tag `env:role` defines who should deal with the header block. This optional attribute may be set to three different values: `none`, `next`, and `ultimateReceiver`. If the header block is
target to a node with the role `none`, no actions should take place. If the value is set to `next`, the header block is destined to intermediary nodes, which forward that part. A header block with an `ultimateReceiver` role or an absent role definition is only intended for nodes assuming the ultimate receiver; this node then begins to process the message body. See Figure 15 for a sample path. Beside these role definitions, a header block may be assigned for mandatory or non-mandatory processing using the `env:mustUnderstand` tag. When it gets processed, it will be removed from the outbound message. The third and final stage in this model is the processing of the body information, which may be done only by the ultimate recipient. The body itself contains information that is not structured by SOAP. It may either include a complete business document which after processing will return another document or a Remote Procedure Call (RPC). The RPC lists a function name to be called and reverts the return values.

![Figure 15: Message path between several SOAP nodes.](image)

SOAP bindings specify rules for carrying a SOAP message on top of an underlying network protocol. The connection models to be used include HTTP and SMTP, which were described in the previous subsection. Since SOAP is completely independent of the transport mechanism in use, it is flexible in selection of the appropriate protocol up to runtime. The mechanism may even change between different nodes.

SOAP fulfills SOA’s need for communication between independent units. It abstracts the physical communication framework and establishes a standard message structure with an extensible header section.

### 5.3.2 Web Services Addressing (WS-Addressing)

A widely-used extension for the SOAP message header is WS-Addressing. Following some initial work, a core recommendation together with specifications for SOAP and other bindings were published by the “W3C Web Services Addressing Working Group” in May 2006. WS-Addressing “provides transport-neutral mechanisms to address Web services and messages” (see Gudgin et al. 2006). Hence, this gives senders and receivers a way of identifying message senders and receivers. It defines two constructs: message addressing properties and endpoint references. The latter is a way of encoding the addressing information necessary to reach service endpoints. It is a collection of abstract properties like, the service endpoint address, reference parameters facilitating a particular interaction, and finally the metadata describing the behavior, policies, as well as capabilities of the endpoint. The information within such an endpoint reference may be mapped to the header block information in the SOAP message. These message addressing properties are meant to identify a Web service endpoint and provide information necessary for bidirectional and asynchronous usage. It augments the message with abstract properties to

---

5.4 Description Layer

The messaging layer offers basic communication although it does not specify what messages must be exchanged to successfully interact with a service. Standards within the description layer fulfill that role. The metadata in this layer fully describes the characteristics of the services. This layer is fundamental in achieving loose-coupling, through which a main SOA requirement is implemented. Thus, the provider may publish its service without caring about the consumer’s technical platform or programming language. Web Services Description Language (WSDL) and the Web Services Policy Framework (WS-Policy) are the two major specifications pursuing these goals. WSDL is the most mature metadata description for Web services. It portrays a service in terms of what it does, were to locate it, and how to interact with it. WS-Policy provides a framework for defining and associating policies with Web service descriptions. Both standards will be depicted further in the following subsections.

5.4.1 Web Services Description Language (WSDL)

The first version of WSDL was released in September 2000, as a combination of IBM’s Network Accessible Service Specification Language (NASSL) and Microsoft’s Service Description Language (SDL). A slightly updated version 1.1 was published later that year and was submitted to the W3C in March 2001 (see Christensen et al. 2001). Following that, WSDL version 1.1 became widely used and thus the de facto standard for service description. In early 2002, the W3C Web Services Description Working Group\(^5\) was founded to further develop the WSDL specification. They finished their work in March 2006 with a candidate recommendation for WSDL version 2.0. It is expected that it will take a few years until this specification is widely deployed. Also, no further development may be foreseen, since the successors will be strongly built upon WSDL.

\(^5\)see http://www.w3.org/2002/ws/desc/, last checked 24/11/2006
Since WSDL version 1.1 is still widely used, both version 1.1 and 2.0 will be described in short subsequently.

In general, WSDL is XML-based, platform, programming language, and protocol independent, as well as extensible regarding multiple message formats and network protocols. It describes the public interface to a Web service, functional characteristics, and other crucial information. Protocol bindings and message formats required to interact with the Web service are delineated in the specification. To be specific, WSDL describes a service as a set of endpoints that operate on messages. First, supported messages and operations are shown abstractly; second, these are bound to a concrete network protocol and message format. There is a clear separation amongst what the service does, how the service is to be interacted with, and from where the service is offered. A client, that wants to invoke a Web service, reads its abstract WSDL description to extract its functionality. During development, this description may be used to generate a specific client code that conforms to the service requirements. Subsequently, the client may use the concrete protocol and format (in most cases SOAP and HTTP) to call one of the listed functions.

Both in the WSDL 1.1 and 2.0 specification, a Web service is described in two fundamental stages: one abstract and one concrete. While abstract definitions provide the application-level functionality, concrete descriptions solve site-specific matters. Upon scrutiny, the two versions are quite disparate. Within WSDL 1.1 (see Christensen et al. 2001), first, the vocabulary, the messages, and then the interactions have to be described abstractly. Data types are specified within the <types> element. For this purpose, external type systems, such as XML Schema Definition (XSD), are used. Then, the messages which are data sent to or received from a service, must be defined. A <message> tag represents one of such aggregations of message parts. Each <part> again is depicted in XSD. Interactions are abstract message exchanges. An <operation> tag represents a combination of input, output, or fault messages. The <portType> tag contains all operations that are collectively supported by one endpoint. Therefore, different message exchange patterns are possible: The one-way method involves a message sent to the service where no response is produced by the service. Another method, request-response, requires the service to receive a message and to send back a response. Yet another method, solicit-message, requires a service to send a message and to receive a response. Finally, notifications require the service to send a message and getting nothing in response. The concrete part of a WSDL description delineates the relation amongst the binding information, communication, and network protocols, and the address of the Web service. The <binding> element specifies the various operations, input, output, and fault messages of each port type within it. The <port> element describes the binding and the network address of a single endpoint. Groups of related ports are kept within the <service> tag. Listing 4 gives a rough structural overview of such a WSDL description.

Despite its phenomenal success and widespread use, WSDL 1.1 has several fundamental limitations. These have been addressed in the development of WSDL 2.0. In most ways, WSDL 2.0 emerged as a much simpler, less powerful, yet more usable service description language than WSDL 1.1. The main problems, that were faced, will be looked at subsequently. For instance, the <message> construct cannot handle a variable number of items to be received or sent. Also, presenting a choice of responses is not possible. To address these restrictions, WSDL 2.0 removes the <message> and <part> elements completely and recommends the direct usage of XML Schema constructs. Another improvement pertains to previously defined interfaces which may be extended for broader usage. WSDL 2.0 focuses less on SOAP and thus provides a more general view on document/literal style classes. Furthermore, since the <service> element did not clearly define how to describe a service with multiple interfaces, its definition was altered. The
previous message exchange patterns were revised which resulted in providing eight clearer sets. The definition of these patterns now also indicates the order of the messages in addition to the source and destination. In order to conform to spoken language, a few simple changes were inculcated: `<description>` substitutes for `<definitions>`, `<interface>` replaces `<portTypes>`, and `<endpoint>` fills the place of `<port>` within the `<service>` element. To summarize these changes, a service is described through the `<types>` element describing the kinds of messages that the service will send and receive, the `<interface>` element describing what abstract functionality the Web service provides, the `<binding>` element describing how to access the service, and the `<service>` element describing where to access the service (see Booth a. Liu 2006). The rough structure of WSDL 2.0 is shown in Listing 5, which expresses the same content as Listing’s 4 WSDL 1.1 structure. More details on the usage of WSDL 2.0 may be found in Chinnici et al. (2006).

5.4.2 Web Services Policy Framework (WS-Policy)

While WSDL provides a definition of the business interface of a Web service, it says nothing about how a service delivers its interface or what it expects from the user. The Web service provider enforces certain Web service policies that play an important role in communicating
necessary constraints and conditions to the service consumer. This may include information on security, authentication, transaction, and reliable messaging issues. See section 5.7 for more information on these QoS topics. In conclusion, service policies become a critical part of any service description augmenting the basic WSDL functional description with a statement of non-functional service behaviors. Depending on this data, a desired service may also be selected based on its support of certain privacy policies or security guarantees. The WS-Policy family extends WSDL with such information. The first publication of this policy, delineating WS-Policy and Web Services Policy Attachment (WS-PolicyAttachment), was released in 2002 and updated in September 2004. Subsequent revisions include version 1.2 by the W3C in April 2006.

WS-Policy inherently describes the grammar for expressing several policy alternatives and their composition as domain assertion combinations. An assertion “represents an individual requirement, capability, or other property of a behavior” (see Bajaj et al. 2006). WS-PolicyAttachment on the other hand, defines how to flexibly attach these policies to existing Web services.

To summarize, WS-Policy and WS-PolicyAttachment add a prominent quality to the description of Web services using WSDL.
5.5 Discovery Layer

A key benefit of using an SOA is the recycling of services and the composition of services solving specific business problems. To fulfill this requirement, developers must be aware of preexisting services and how to interact with them. In order to achieve this, explicit service description information viz. metadata and metadata management solutions have been developed. Metadata discovery, assuming publication by the service provider, may occur in two ways:

1. via a centralized registry, i.e. a Universal Description, Discovery, and Integration (UDDI) registry, or
2. via distributed discovery, i.e. Web Services Metadata Exchange (WS-MetadataExchange).

Universal Description, Discovery, and Integration (UDDI) provides a standardized process to publish and discover Web service information. It supplies a registry for metadata that gives references to all kinds of information, including WSDL documents, policies, etc. It also offers several powerful and flexible techniques for querying the registry. In contrary to UDDI mainly being used as a centralized registry, Web Services Metadata Exchange (WS-MetadataExchange) is the standard applied to directly inquire the endpoints of Web services which in turn returns its own metadata. This metadata may be the WSDL document, the policies, or anything else. We now discuss both specifications in the following two subsections.

5.5.1 Universal Description, Discovery, and Integration (UDDI)

In September 2000, IBM, Microsoft, and Ariba took a new initiative in proposing a Web service standard that would serve as a giant online repository for companies seeking business online. UDDI’s current version 3.0 from October 2004 has become an OASIS standard (see Clement et al. 2004).

As mentioned before, UDDI is a definition of the information provided about each Web service. At the same time, UDDI provides an aggregation service. These aspects render UDDI an extremely efficient standard. It may be likened to an automated online phone directory, like the Yellow Pages. UDDI also specifies protocols for accessing and updating a common registry. These queries are usually sent as SOAP messages. The registry may be queried at design time or dynamically at runtime, to find a Web service that suits the given requirements. When using UDDI during design time, the developer building the new application with the need to incorporate existing services may use a registry to find an appropriate service. The user sends a query to the UDDI registry, gets a reply with a list of suitable services, may refine the search criteria, and finally may use further descriptions, for instance WSDL to generate code artifacts necessary to invoke the service. On the other hand, through the use of UDDI during runtime, implementations are dynamically bound to a service. And hence, another key attribute of SOA is realized through these methods. Rather than tightly binding a client to a particular implementation, UDDI may be used to find an implementation of a certain service at runtime based on some specific search criteria. Evidently, UDDI is especially designed to be used with SOAP and WSDL.

There are three different types of UDDI repositories: public, inter-enterprise, and intra-enterprise.

---

6see http://www.uddi.org, last checked 24/11/2006
Public UDDI registries are universally accessible. A sample public implementation of the UDDI standard is the UDDI Business Registry\(^7\), whose nodes are managed by a group of vendors including IBM, Microsoft, and SAP.

Inter-enterprise UDDI registries share the control of the UDDI services between known cooperating business partners. It is usually more focused on the needs of one industry and supports confidence between providers and consumers.

Intra-enterprise UDDI registries represent a private internal repository. They allow much more control over which services are published and how they are described to be reused within the same enterprise.

To store relevant metadata about Web services, XML-based UDDI uses four primary data types: `<businessEntity>`, `<businessServices>`, `<bindingTemplates>`, and `<tModel>`. See Listing 6 for a roughly structured UDDI document. The root element is `<businessEntity>` representing one service provider. It contains information like names, simple descriptions, identifiers, and contract information. Two businesses may form an association through a `<publisherAssertion>`.

Each entity might own an arbitrary amount of `<businessServices>`. Each `<businessEntity>` and each `<businessService>` contain a unique identifier `<businessKey>` or `<serviceKey>` to possibly make a reference to them across the repository. Beside non-technical information about the service, like name, description, and categorization, the `<businessService>` element holds a list of `<bindingTemplates>`. They describe the technical information to access the service, most likely through a URI and/or email address. Each `<bindingTemplate>` represents one alternative access point around different endpoints, which is again uniquely identified through a `<bindingKey>`. The most important enclosed element is `<tModelInstanceDetails>` with several `<tModelInstanceInfo>` elements which contain the specification, that uses a reference to a technical model `<tModel>`. The `<tModel>` is used for a wide range of purposes in UDDI. According to the specification, it is used for value sets such as category systems and identification systems, for structured categorizations, for postal address formats, and for references to other documents. To effectively locate a service, the service has to be assigned to several categorizing schemes and taxonomies. Categorization is represented in the UDDI data model by the `<categoryBag>` element. Each classification is registered as a `<tModel>`.

There exists a formal mapping to use the information already provided by the WSDL description. WSDL’s `<portType>` or `<interface>` element is mapped to UDDI’s `<tModel>`, as well as WSDL’s `<binding>`. A `<service>` is represented as a `<businessService>`, a `<port>` or `<endpoint>` and is mapped to the `<bindingTemplate>` element. Beside WSDL, the previously described WS-PolicyAttachment Specification (see section 5.4.2) may be used with UDDI as well. It describes how to attach policies to each of the four main UDDI data types.

5.5.2 Web Services Metadata Exchange (WS-MetadataExchange)

While UDDI addresses scenarios relying on a central repository, WS-MetadataExchange is more likely to refer to those in which the requesters discover service endpoints dynamically at runtime. This bears the advantage of interaction customization fitting a particular requester, for example based on the identity of the other endpoint. This increase in flexibility accounts for varying QoS issues suitable for disparate target audiences. Furthermore, the requested metadata is always up to date. A service requester may ask the service provider directly for the required metadata; no third party registry may be involved. Overall, WS-MetadataExchange is an interaction protocol

\(^7\) see [http://uddi.ibm.com](http://uddi.ibm.com), last checked 24/11/2006
for the discovery and the retrieval of Web service metadata from a specific Internet address through querying the network for WSDL, XML Schema, and WS-Policy information. The first public draft was published by BEA Systems, IBM, Microsoft, SAP, Sun Microsystems, and others in February 2004 and later refined in September (see Ballinger et al. 2004).

To make use of this discovery technique, each service provider must implement a universally
accessible WSDL interface at its endpoint defined by WS-MetadataExchange to accept WS-
MetadataExchange messages. The location of the endpoint must be described using WS-
Addressing. Inquiries regarding specific metadata properties are sent as a SOAP message request
and parsed to extract the metadata in question. Finally, a reply message is formulated to provide
the required information. This metadata may then be embedded in the response, or it contains
a reference or location. A metadata reference directs to an endpoint where the metadata may
be retrieved. A metadata location contains a URI. Listings 7 and 8 show a simple metadata
request and its response. After such a metadata response has been received by the metadata
requester, a simple <wsx:Get/> request may be sent to retrieve the actual information from the
defined location.

5.6 Business Process Layer

In the previous subsections, we discussed, how Web services are described, how they may then
be registered and later discovered, and finally how messages may be exchanged between Web
services and clients. The invocation of a single Web service, however, only suffices to suit simple
business cases. To actualize more comprehensive and complex processes, which span several core
functionalities across various companies, another layer of abstraction is required. Within this
business process layer, single Web services may be connected to a complete process, as illustrated
in Figure 16. A business process then specifies the potential execution order of operations from
a collection of Web services. It also outlines which partners are involved and how they are
included in the business process. Furthermore, joint exception handling for collections of Web
services, etc. are specified. Since these service compositions are a key principle for realizing
SOAs, chapter 6 is exclusively devoted to this topic.

![Figure 16: Isolation of the business process from the a set of services.](image)

5.7 Quality of Service (QoS) Issues

QoS issues are a fundamental aspect in the domain of Web services, wherein the requirements
associated with the overall reliability of Web services are evaluated. It involves security, reliability
of message delivery, management, and the support for transactions. We look at it in further
detail in the following subsections.
5.7 Quality of Service (QoS) Issues

5.7.1 Security

In the world of e-business, there is a heavy exchange of data between a number of systems and companies. The “security issue” aims at enhancing the reliability of Web services, despite the impending threat of security attacks.

According to Endrei et al. (2004, page 146f), the following security requirements must be addressed in any architectural solution:

- **Identification:** The party accessing the resource is able to identify itself to the system.
- **Authentication:** There exists a procedure to verify the identity of the accessing party.
- **Authorization:** There exists a set of transactions the authenticated party is allowed to perform.
- **Integrity:** The information is not changed on its way.
- **Confidentiality:** Information cannot be intercepted en route.
- **Auditing:** All transactions are recorded so that problems may be analyzed.
- **Non-repudiation:** Both parties are able to provide legal proof to a third party that the sender did send the information, and the receiver received the identical information.

During the last few years, most Web services solely relied on transport level support for security functions. In this layer, secure transport channel technologies, like HTTPS, may be applied. HTTPS relies on Secure Sockets Layer (SSL), with which secure sessions are established. The service consumer and provider may authenticate themselves through self-assigned or agency-assigned certificates. Such security technologies only fulfill the security requirements of identification, authentication, integrity, and confidentiality. The level of function is significantly less than necessary for today’s distributed applications.

To satisfy the need for complete security, a family of security specifications covering most of the named security requirements had to be developed. Initially, Microsoft started to work on Web Services Security (WS-Security) in 2001. Since then, it has undergone several alterations through additional support by a wide range of companies, including IBM, SUN, VeriSign, and SAP. WS-Security’s current version 1.1 has been approved as an OASIS standard in February 2006 by the OASIS Web Services Security Technical Committee. WS-Security solves security issues in scenarios where intermediaries exist in long Web service interactions. It adds end-to-end security to the whole message path by restricting each intermediary to access only that data which is specifically destined to it. Also, the possibility integrating multiple systems with various security domains and technologies demands that translation and exchange of security information be equally available to all participating systems. Hence, WS-Security specifically secures multi-hop and multi-party Web service communications. The set of security specifications is arranged in a layered structure consisting out of WS-Security itself, WS-Policy, Web Services Trust Language (WS-Trust), Web Services Privacy (WS-Privacy), Web Services Secure Conversation Language (WS-SecureConversation), Web Services Federation Language (WS-Federation), and Web Services Authorization Language (WS-Authorization) (see Figure 17). Although WS-Security is an established standard, further specifications are still being built on it, as well as upon foundation technologies such as SOAP, WSDL, and XML.

---

The WS-Security specification (see Lawrence et al. 2006) serves as the basis for any other security specifications within that family. It provides a set of mechanisms to help the developer of Web services secure an exchange of SOAP messages. Specifically, it protects messages through the application of message integrity, message confidentiality, and single message authentication. Also, mechanisms to associate security tokens with SOAP messages are presented. It accommodates a variety of authentication and authorization technologies, including X.509 tokens\(^9\) and Kerberos tokens\(^10\). Message integrity is provided by leveraging “XML Signature” and “XML Encryption”. XML Signature makes it possible to securely verify the origin of a message through digital signatures leading to validation and non-repudiation. XML Encryption allows the ciphering of specific parts in an XML document.

WS-Policy, another specification out of the Web services security family, makes security policies available to service consumers (see subsection 5.4.2 for more details). WS-Privacy describes syntax and semantics for binding organizational privacy policies to Web services and instances of data in messages. Since “trust relationships” are essential for security, WS-Trust (see Anderson et al. 2005a) describes an extensible model for setting up and verifying such relationships. It is built on top of WS-Security defining additional primitives and extensions to issue, exchange, and validate security tokens. It also enables the issue and dissemination of credentials between the two trust domains.

WS-SecureConversation (see Anderson et al. 2005b) builds on the concepts of WS-Security, WS-Trust, and WS-Policy to provide secure conversations between services. It helps to establish and share security tokens among participating parties and allows Web services to set up and agree on which security servers they trust. WS-Federation (see Bajaj et al. 2003), also built on top of WS-Security and WS-Trust, describes how to establish and manage trust relationships in a heterogeneous federated environment. The purpose of WS-Authorization, as the last arising specification of the Web services security family, is to describe how access policies for a Web service are specified and eventually managed. It is designed to be flexible and extensible with respect to both authorization format and authorization language.

---

\(^9\)X.509 is a standard for public key infrastructure mainly specifying standard formats for public key certificates and a certification path validation algorithm. It is under the supervision of the International Telecommunication Union Telecommunication Standardization Sector (ITU-T).

\(^10\)Kerberos is a computer network authentication protocol which allows individuals communicating over an insecure network to prove their identity to one another in a secure manner. Kerberos prevents eavesdropping or replay attacks, and ensures the integrity of the data. Its designers at the Massachusetts Institute of Technology (MIT) aimed primarily at a symmetric key cryptography that requires a trusted third party.
5.7.2 Reliability

After security issues within the Web service domain, we now turn to reliable interactions between service providers and requesters which represent an equally important QoS characteristic. The presence of software components, systems, and/or network failures and thus unsuccessful completions of message exchange must always be assumed. Connections typically become unreliable when the connection between the service requester and provider breaks suddenly, when messages get delivered incompletely or multiply, or when messages are sent in the wrong order. It is the developer’s task to ensure reliable message delivery even when such failures occur. Several solutions for reliable messaging for Web services have been proposed, out of which WS-ReliableMessaging bears the most promise for broad adoption and standardization. WS-ReliableMessaging was jointly developed by IBM, BEA Systems, Microsoft, and TIBCO Software and originally published in March 2004. It was updated with additional support of Oracle and SAP as a OASIS committee draft in March 2006 (see Davis et al. 2006).

This specification defines protocols for establishing and maintaining a reliable interaction between a client and a service. It supports three delivery assurance types:

- **In-order delivery**: The messages are delivered in the same order in which they were sent.
- **At-least-once delivery**: Each message that is sent is delivered at least one time.
- **At-most-once delivery**: No duplicate messages are delivered.

Based on these basic assurances, more complex types like “exactly-once transference”, may be created through combination. Furthermore, message persistence and delivery status awareness are assured.

WS-ReliableMessaging is defined as a set of SOAP header extensions enabling the named assurances. The addressing aspect is factored out to WS-Addressing (see subsection 5.3.2). Within WS-ReliableMessaging, reliable messaging is enabled by a shared context for a set of messages, referred to as a sequence, to be delivered between the sending and receiving endpoints. Each message within such a sequence is assigned a unique number. The receiving endpoint acknowledges receipt of a message within a sequence by indicating the range of received messages. Since such an acknowledgment document contains the acknowledgment information for all the messages that have been received within one sequence, it does not require retransmission if this information sometimes fails to reach the sending endpoint. WS-ReliableMessaging’s processing model defines an abstract model for the protocol which conforms to the described concept.

Overall, WS-ReliableMessaging protocols comprehensively allow different operating and middleware systems to reliably exchange messages, thereby bridging different infrastructures into a single, logically complete, end-to-end model for Web service reliable messaging. Thus, it enables enterprises to extend Web services to support reliable B2B exchanges.

5.7.3 Transaction

Today’s business scenarios are constructed out of multiple Web services exchanging a large amount of messages. Applications enabling such extensive message exchange may be complex, spanning heterogeneous, loosely-coupled distributed systems that are prone to failure. This introduces significant reliability problems. To obtain a reliable outcome, the various services that contribute to the application must universally agree on an undisputed resolution. “Transaction” represents an approach fulfilling these contracting requirements. Traditionally, transactions must
have the following properties to prevent from application failure. They are often referred to as ACID properties.

- **Atomicity**: If the transaction is successfully completed, all of the application actions must take place; on the contrary, following an unsuccessful completion, none of the actions are permitted to occur.
- **Consistency**: Each transaction must produce a consistent result and perform valid state transformations on completion.
- **Isolation**: While a transaction executes, intermediate states have to stay invisible to other transactions. Thus, transactions must be executed serially.
- **Durability**: After a transaction has been successfully completed, the changes must be maintained and survive failure.

In order to facilitate these transaction properties and reliable applications within a Web service environment, several Web service transaction specifications were introduced in September 2003 by a consortium consisting of IBM, BEA Systems, Microsoft, Arjuna, Hitachi, and IONA. The current version is dated August 2005. This set of specifications contains an extensible coordination framework, described in the Web Services Coordination (WS-Coordination) (see Cabrera et al. 2005a) specification that delivers a general mechanism for the initiation and subsequent consent on the outcome of multi-party, multi-message Web service tasks. Protocols defined with WS-Coordination provide a context to identify Web service operations as part of a particular activity. They allow other Web services to register its interest in participating in a task. They also permit the selection of a specific coordination protocol to be used between the coordination service and any participating Web services at the completion of the activity. Specific coordination types or completion processing patterns, which extend a WS-Coordination description are depicted in Web Services Atomic Transaction (WS-AtomicTransaction) and Web Services Business Activity (WS-BusinessActivity). Developers may use either or both when building applications that require consistent agreement on the outcome of distributed activities. While WS-AtomicTransaction handles short-term activities with ACID properties (see Cabrera et al. 2005b), WS-BusinessActivity describes long-running business transactions where intermediate steps are visible to the participants (see Cabrera et al. 2005c).

### 5.7.4 Management

The manageability of Web services concerns the propensity to discover the existence, availability, and “health” of the Web services infrastructure, service registries, and Web service applications. Ideally, the management system should also be able to control and configure the infrastructure and components of the implemented SOA. The management of the underlying infrastructure and networks required to implement and run the Web service, is imperative to ensure a suitable management over the end-to-end Web services environment. It must also extend outside the enterprise to include integrated business partners, suppliers, and customers.

To address this field, OASIS created a technical committee called Web Services Distributed Management Technical Committee\(^ {11} \) in 2003. In the following September, a specification called Web Services Manageability (WS-Manageability) was submitted to that committee by IBM and others. HP, Sun, BEO, IONA, and others also submitted a specification called Web Services Management Framework (WSMF) to this committee. The committee then brought out its first

specifications on Web Services Distributed Management (WSDM) in February 2006. Overall, the field of Web services management is still in its infancy and requires further developments.
6 Services Composition

In the previous section, the discussion only pertained to single Web services. Such services are limited to fulfilling basic tasks only. But today, business processes require integration across enterprise boundaries. Cross-organizational connectivity addressing customer, supplier, and business partner interaction has become fundamental to business. This empowers these long-running interactions with the electronic advantage, revolutionizing speed and performance. To carry out complex business cases, available core services must be combined and arranged to suit their requirements and follow an explicit process model. The resulting service is then often referred to as a composite service.

Some business goals for implementing SOAs must especially be realized within the composition layer which will be revisited in this chapter. The technology to combine several core skills into large processes allows better alignment of the IT infrastructure with constantly changing aspects of business. The encapsulation and externalization of process logic from the actual service grants a business more flexibility in arrangement of process flows. It also allows rapid application development and lowers time-to-market by splitting up complex problems into independent sub tasks. The Total Cost of Ownership (TCO) will be decreased as well, since the sub tasks encapsulated within core services may be reused easily. Effective cost optimization and quality control may be realized by careful selection of partners and utilization of outsourcing opportunities; this is a step towards achieving the overall business goal effectively. Service composition thus becomes fundamental to larger enterprises.

For additional consideration of composite services or complex Web services, as they are referred to if the composition is based on Web services, a distinction must be made between orchestration and choreography. This will be examined further in the following section.

6.1 Orchestration vs. Choreography

The terms orchestration and choreography explain two different aspects of creating business processes from single services. They are often used interchangeably, but even if they overlap, they differ in detail. Peltz (2003) and Havey (2005) delineate the difference between them as:

Orchestration is the process of combining multiple services into a composite service to achieve a given goal. Thus, recursive composition of services is possible so that it will yield another service. It describes how services may interact at the message level, including the business logic (parallel activities and conditional branching logic) and the task execution order. Conditions under which different services are being engaged may also be specified. Orchestration refers to an executable business process that may interact with services that span applications and/or organizations to define a long-lived, transactional, multi-step process model. Furthermore, services are combined imperatively and shows how the process is implemented. Orchestration always represents control
6.1 Orchestration vs. Choreography

from one party’s perspective – the process is initiated, executed, and terminated by the same entity.

Choreography is generally proposed as a complementary technique to orchestration. On the other hand, it does not entail any orchestration at all. It is applicable to situations where multiple parties collaborate in a peer-to-peer manner and allow each involved entity to describe its part in the interaction. This model describes the relationship between services in a peer-to-peer environment. Specifically, it provides a global view on public message exchanges between collaborating parties regardless of the supporting platform or the programming model used. It already does basic validation on the message level. Message sequences get tracked among multiple parties and sources. In contrast to orchestration, choreography combines services declaratively. It is purely a description about current occurrences and thus may only be used to generate behavior but not to execute the process.

There are two reasons for separating the public and internal or private aspects of business process behavior (Newcomer a. Lomow 2004): “One is that organizations do not want to reveal their internal business processes and data management to their business partners. The other is that separating public processes from private processes provides the freedom to change private details of the internal process implementation without affecting the public business protocol.”

![Figure 18: Distinction between orchestration and choreography for business processes.](image)

Figure 18 concisely reviews the differences. While orchestration refers to an imperatively described and executable process, choreography is a declarative description to track message sequences between multiple collaborating sources. When talking about service composition, people typically refer to BPEL as the orchestration language and Web Services Choreography Description Language (WS-CDL) as the choreography language. Additionally, there exist several minor languages that may or may not rely on Web services; these include Business Process Modeling Language (BPML), XML Process Definition Language (XPDL), and Electronic Business using Extensible Markup Language (ebXML). The named ones will be described in the following subsections in more detail. Figure 19 depicts the historical development of common composition languages and specifications.
6.2 Business Process Execution Language (BPEL)

BPEL is an XML-based language that primarily describes the process flow of complex business processes, which require Web services to implement their functionality. The term “BPEL” is used in short without referring to a certain version of the standard. Before a detail description of the language, it is imperative to first explore the origins of this specification (see Figure 19). The earliest efforts in developing a business process language were carried out within the Web Services for Business Process Design (XLANG) language by Microsoft and Web Services Flow Language (WSFL) by IBM in May 2001. In August 2002, a combination of these two languages, which was then called Business Process Execution Language for Web Services (BPEL4WS) version 1.0, was jointly proposed by Microsoft, IBM, and BEA Systems. Later, with additional contributions from SAP and Siebel Systems its version 1.1 was released in May 2003 and submitted to the OASIS WSBPEL Technical Committee\(^1\), so that the specification could be developed into an official, open standard. To deal with all aspects of business processes, several extensions to BPEL4WS were pursued during years that followed: In March 2004, IBM and BEA Systems proposed a combination of BPEL with Java, named BPEL for Java technology (BPELJ), that allowed these two programming languages to be used together in order to build complete business process applications. Furthermore, IBM and SAP invented WS-BPEL Extension for People (BPEL4People) in July 2005 in order to include human user interactions within BPEL. In September of the same year, they also suggested WS-BPEL Extension for Sub-Processes (BPEL-SPE) to define sub-processes for reuse and modularization within BPEL. The OASIS WSBPEL Technical Committee first published a draft on version 2.0 in December 2005; the development of version 2.0 is currently in progress and will be finished till the end of 2006. Before, in September 2004, OASIS already decided to name the new specification Web Services Business Process Execution Language (WS-BPEL) 2.0 in conformity with other WS-* standards.

6.2 Business Process Execution Language (BPEL)

6.2.1 XLANG: A forerunner of BPEL

The initial public draft of XLANG was released in May 2001 by Microsoft. The initial development of XLANG was to support sequential, parallel, and conditional process flows for Microsoft’s BizTalk Server. Thatte (2001) defines XLANG as “a notation for the specification of message exchange behavior among participating web services” to execute business processes automatically. Hence, it orchestrates a series of available Web services to create multi-party business processes. Its focus is on the interaction between contractors and the exchanged messages. XLANG is an open protocol based on XML that is fundamentally derived from WSDL; it serves to establish a communication between the business process and the specific Web services.

As long as the public contract defined by the protocol is impervious to change, only external views of business protocols need be defined XXX, the implementation behind the public “face” of each service provider is completely irrelevant. As a result, these contracts do not support any business related semantics. They are merely a mapping between two port types – input and output – which interact. Beside this inter-service description feature, XLANG also provides a functionality for the intra-service level by extending WSDL with basic control process structures. Furthermore, XLANG also includes a robust exception-handling facility, with support for long-running transactions. For more information on the usage of XLANG specification with explanatory examples see Thatte (2001).

6.2.2 WSFL: A forerunner of BPEL

At the time XLANG was being developed by Microsoft, IBM was in turn working on their business process definition language, called WSFL. IBM’s results were released in May 2001. In contrast to XLANG, WSFL is a graph-oriented language. It, however, still corresponds to Microsoft’s counterpart to a great extent.

WSFL delineates two types of Web service compositions (see Leymann 2001): The first, referred to as Flow Model, specifies usage patterns of a collection of Web services. It thus describes the business process by defining the execution order of several Web services. Also, business rules for sequencing these steps may be integrated within this model. This may be viewed as an orchestration or description of private process flows. The second is described by WSFL as a Global Model. This type specifies the interaction pattern of a collection of partners. Hence, linking the endpoints of Web services, which correspond to the interface of two Web services. This type manages public process flows and could therefore be seen as a choreography language. For more details on its usage, see the specification (see Leymann 2001).

6.2.3 BPEL4WS in Detail

As discussed previously, XLANG and WSFL share several common features especially in their separation of public and private flows to model orchestration and choreography. These early workflow languages merged in July 2002 into BPEL4WS, the previous name this hybrid language. In BPEL4WS, processes may be created by using a combination of the graph-oriented style of WSFL and the algebraic programmatic respectively block-structured style of XLANG. After submitting the specification to OASIS for standardization in 2003, the technical committee is set to produce the next version, renamed WS-BPEL 2.0, as its first output. At the time of writing, the process was not yet complete, as a result, this thesis uses the last stable version of the specification for its reference (see Andrews et al. 2003).
BPEL4WS is built on top of several XML specifications, including WSDL 1.1, XML Schema 1.0, and XPath 1.0. WSDL messages and XML Schema type definitions provide the data model used by BPEL4WS processes. XPath provides support for data manipulation. All external resources and partners are represented as WSDL services. While the WSDL interface specifies the allowed operations, BPEL4WS provides a method to sequence them. The result of each BPEL4WS description is also a WSDL description allowing recursive composition. Beside these specifications, it is also recommended to rely on WS-Addressing, UDDI v2.0, and WS-Security.

BPEL4WS describes two usage patterns:

1. **Executable processes** define the exact details of business processes. This involves the description of specific activities intended to be performed and specific services intended to be used. The result may be executed using an orchestration engine.

2. **Abstract processes** or business protocols represent a public process description, on the contrary to private process descriptions using executable processes. An abstract process indicates the public message is exchanged between participating parties only. Hence, the description is not executable and does not pertain to internal details of the process flow.

Essentially, while the executable process pattern reflects the orchestration concept, abstract processes simulate the choreography of services.

```
<process name="ProcessName" ... >
    <partnerLinks> ... </partnerLinks>
    <variables> ... </variables>
    <correlationSets> ... </correlationSets>
    <faultHandlers> ... </faultHandlers>
    <compensationHandler> ... </compensationHandler>
    <eventHandlers> ... </eventHandlers>
    <!-- Activities defining the business process -->
    <sequence> ... </sequence>
</process>
```

Listing 9: Basic structure of a BPEL description for business processes.

The basic structure of a BPEL4WS description is given in Listing 9. A BPEL4WS process consists of several steps, which here are called activities. Both basic and structured activities are supported. A basic activity describes simple tasks (see Andrews et al. 2003):

- Invoke one-way or request-response operations with another Web service, using `<invoke>`.
- Activate a blocking wait for a matching message to arrive, using the `<receive>` construct.
- Send a message in reply to a received message, using `<reply>`.
- Manipulate and updating the value of data variables, using the `<assign>` pattern.
- Indicate faults and exceptions from inside the business process, using `<throw>`.
- Wait for a given time period or until a certain time has passed, using the `<wait>` construct.
- Insert a “no op” instruction into a business process, using `<empty>`.
6.2 Business Process Execution Language (BPEL)

- Terminate the entire executable process, using the `<terminate>` construct.

Structured tasks are used to combine the primitive tasks into more complex processes, also supporting conditional looping and dynamic branching. These are essentially the underlying programming logic for BPEL4WS:

- Define a collection of activities to be performed sequentially, using `<sequence>`.
- Select one branch of activity from a set of choices, using the `<switch>` construct.
- Define activities that are repeated until a criteria has been met, using `<while>`.
- Wait for a suitable message to arrive or for a time-out alarm to go off, using `<pick>`.
- Specify one or more activities to be performed concurrently, using the `<flow>` construct.
- Define a nested activity with its own variables, fault handlers, ..., using `<scope>`.
- Invoke compensation on an inner scope that has already completed, using `<compensate>`.

By combining all these constructs, complex business processes may be defined in an algorithmic manner.

Partner links and variables are inherent in the definition of each process. Partner links, described within the `<partnerLinks>` group, represent the communication exchange between collaborating partners, the process acting as one partner and another service acting as the other. It relies on `<portType>` definitions within the WSDL description of the underlying service. Another element commonly used is the `<variables>` element, which stores the state information related to the immediate workflow logic within the process. These are also used to manage data persistence across Web service requests. As a result, sent and received messages as well as other relevant data such as time-out values may be stored within the process.

BPEL4WS extends a definition for fault and compensation handlers. These are similar to ‘catch’ clauses in object-oriented programming languages and are triggered by the execution of a ‘throw’ task. Errors may be compensated and handled in this manner.

It is interesting to note that, no graphical notation is given within BPEL4WS. This is because of a decision taken by the OASIS technical committee ruling it irrelevant. Some vendors, however, invented their own notations and an informal mapping from BPMN to BPEL4WS is also available. More details on BPMN have been given in section 4.5.

6.2.4 Extensions to BPEL

After BPEL4WS was first introduced, vendors, not entirely satisfied with it, proposed several extensions. Most of these extensions were initiated by IBM. The three most important extensions are BPELJ, BPEL4People, and BPEL-SPE, which will be described shortly in the following paragraphs.

In March 2004, IBM and BEA Systems published BPELJ – a combination of the BPEL and the Java programming language (see Blow et al. 2004). BPELJ allows the inclusion of sections of Java code, called Java snippets, within the original business process definition. While BPEL is only geared towards ‘programming in the large’ supporting the logic of business processes at a higher level of abstraction, Java adds some ‘programming in the small’ capabilities to implement the actual business functions. Java snippets may be used for value calculations, document or message constructions and deconstructions, loop and branching conditions, logic of business
functions, as well as to perform side-effects without having to create a separate Web service. Although BPELJ introduces a few minor changes to the original BPEL specification, it primarily extends its functionality through the definition of Java partner links, additional variables, and <bipelj:snippet> activities. These activities consist of in-line Java code. See Listing 10 for a sample snippet. The Java snippet has complete access to all variables and partner links within its local scope. Due to this access, the snippet’s inherent efficiency is also enhanced. In conclusion, developers may create business processes using Web services and Java resources compositely.

```xml
...<variables>
  <variable name="factor1" type="xsd:float"/>
  <variable name="factor2" type="xsd:float"/>
  <variable name="product" type="xsd:float"/>
</variables>
...
<bipelj:snippet>
  <bipelj:code>
    product = factor1 * factor2;
  </bipelj:code>
</bipelj:snippet>
...
```

Listing 10: A Java snippet activity performing a simple value manipulation.

In 2005, a group of people from IBM and SAP published two additional extensions to the BPEL language – BPEL4People in July (see Kloppmann et al. 2005a) and BPEL-SPE in September (see Kloppmann et al. 2005b). BPEL was originally designed for automated business processes based on Web services, however, BPEL4People adds to it the additional support of ‘human user interactions’. These user interactions may range from simple to complex scenarios. For the integration of people as a dynamic resource in the system, an extensive study must be conducted on human roles viz. people activities. Another area of scrutiny is the identification of the people to interact with viz. people links.

People activities may broadly be classified into two distinct scenarios: simple and complex tasks. Simple scenarios may be characterized by tasks like receiving notifications or administration of processes that run for long or are subject to time-out. Permissions along a personnel hierarchy and the associated processes, are also part of this category. On the other hand, complex tasks involve independent decisions of two different entities or escalations of task-load due to pending duties on account of person failure. Task delegations and execution chains may also be considered a part of this category. In most cases, people activities must be linked to the organizational directory to decide on these tasks. Therefore, people links and people queries must be addressed. A specification defining the syntax and the semantics of BPEL4People has not been published till date.

BPEL-SPE allows the definition of a business process within the context of another business process. Kloppmann et al. (2005b) describe these sub-processes as a “fragment of BPEL code that may be reused within a process or across multiple processes” and hence supports modularization. A noteworthy feature of BPEL-SPE is that it distinguishes between standalone and inline sub-processes. The former is a regular BPEL process to be reused across multiple processes. It implements a singular receive-activity consuming the input message of the implemented operation. There is only a single call-return between the sub-process and its parent. Inline sub-processes are similar to standalone ones, however, they are locally defined within a BPEL process. It may thus be reused within that process. Because of the special relationship between the sub-process and its parent process, specific fault and composition handling is
6.3 Web Services Choreography Description Language (WS-CDL)

As discussed earlier, BPEL is mainly used to define the flow of execution, messages in the flow, as well as, other actions such as fault and compensation handlers. Overall, it allows the existing Web services to be orchestrated into composite Web services. Although BPEL’s abstract processes are intended to be used for choreographic purposes, they only represent an oversimplified approach. Languages that exclusively deal with choreography, such as WS-CDL, have an added advantage since they describe the relationship between services in a peer-to-peer scenario. As mentioned in section 6.1, the business logic of each participant within the choreography is omitted. Instead, the message exchange between collaborating partners is described in a declarative way.

In September 2003, a special working group for Web services choreography was formed within the W3C after Oracle’s submission of their original work on choreography. After a model overview (see Burdett a. Kavantzas 2004) and choreography requirements (see Austin et al. 2004) were elaborated, WS-CDL’s current version 1.0 was released in November 2005. Before this working group started its work, in June 2002, BEA Systems, Intalio, Sun Microsystems, and SAP had already released the first Web service choreography specification, which they called Web Service Choreography Interface (WSCI). To understand the underlying concepts of WS-CDL, we will look at WSCI in the following subsection.

6.3.1 WSCI’s Influence

The W3C describes WSCI as “an XML-based interface description language that describes the flow of messages exchanged by a Web Service participating in choreographed interactions with other services.” (see Arkin et al. 2002). It thus furnishes a global view and is a typical choreography language. It is layered on top of the existing Web service stack, primarily in conjunction with WSDL, although not necessarily bound to it. This layer describes the required behavior of a Web service corresponding to the message exchange it must support. While WSDL describes a static interface in the context of a well-defined message exchange (see Figure 20).

Overall, WSCI meets the following crucial requirements: It sets up message choreography which is the order in which messages are sent and received subject to certain rules. It also delineates transaction boundaries, compensation and exception handling. Furthermore, the Web service is endowed with the capability to manage multiple conversations. It defines properties, selectors, and connectors which describe the interconnection of participating Web services. Finally, WSCI allows dynamic participation which ensures the selection of the right target service.

Referring to Listing 11, this WSCI code structure may be added to the WSDL description in order to define a dynamic interface for the described Web service. The <action> element which is encapsulated within the <process> element and finally the <interface> tag, describes the basic request and response message referring to a specific WSDL operation. Structured activities may be presented by performing activities sequentially or in parallel. After the WSCI Note has been published in August 2002, WSCI was not pursued further. Instead, a Web Services Choreography Working Group was formed to develop another choreography description which we will turn to now.
6 Services Composition

Figure 20: Collaboration between service implementations interfaced by a WSCI description.

```xml
<interface name = "InterfaceName">
    <process name = "ProcessName" instantiation = "message">
        <action name = "ActionName" role = "Role" operation = "SpecPortType">
            ...
        </action>
    </process>
</interface>
```

Listing 11: WSCI structure that would be included into the WSDL description.

6.3.2 WS-CDL in Detail

Kavantzas et al. (2005) define WS-CDL as “an XML-based language that describes peer-to-peer collaborations of participants by defining, from a global viewpoint, their common and complementary observable behavior; where ordered message exchanges result in accomplishing a common business goal.” Each participant may then use the global definition viz. the contract to build and test solutions that conform to it. This language is intended to specify peer-to-peer protocols for B2B scenarios, where each party remains autonomous, integrations cannot be controlled by a single entity and hence the control for the overall flow has to be shared. Moreover, its description is regardless of the platform or programming model used. WS-CDL cannot be executed; it is merely a declarative language for defining interaction patterns to which multiple parties may agree. As mentioned before, WS-CDL complements BPEL and other extended Web service technologies, but is not explicitly bound to WSDL and may be used for a variety of other SOA services.

During its development, WS-CDL pursued the following wide range of goals (see Burdett a. Kavantzas 2004): Choreography definitions should be reusable by different participants and hence new choreographies should possibly be composed out of existing ones. It should also describe cooperations between multiple participants. Furthermore, relevant documentation and semantics should be included for all components. The choreography description should be
information oriented, thus state changes should be recorded. Additionally, these states should be aligned between all participants. Finally, exception handling and verification methods should be supported.

The pursuit of these goals led to the development of a unique language structure, that we will now discuss. WS-CDL itself is a layered language with one Package containing all other layers. All WS-CDL descriptions include a set of Roles representing related behaviors and services, Participants with related roles, and Relationships between these roles. Furthermore, it contains Channels used by the roles to interact with each other and Choreographies that use these channels to complete Interactions. Usually, the choreography structure includes a choreography definition with sub-choreography definitions, variable and token definitions, and the actual choreography. In other words, it simply describes how individual roles may collaborate to achieve a common goal. Through the addition of a Structured Composition layer, existing Interactions and Choreographies may be reused and arranged into sequences and parallel activities. As required for some processes, information pertaining to objects in the choreography may be stored within Variables and parts of the variable may be referred to through Tokens. In the next layer, Non-Observable Conditionals may be added to provide branching based on changes that occur within interactions. If knowledge between concerned roles must be shared, the state variables and their values must be made available. For that, such Observable Conditionals may be added to a choreography and finally some state management is required. Corresponding to the structure of WS-CDL, Listing 12 provides the basic XML structure for a simple process without defining non-observable and observable conditionals. For further explanations on the usage of WS-CDL elements, refer Kavantzas et al. (2005) and Burdett a. Kavantzas (2004).

6.4 Other Composition Standards

BPEL, WS-CDL, and their forerunners are not the only standards in the domain of Web services composition to be proposed in recent years. At the same time as OASIS’ development of BPEL and W3C’s work on WS-CDL, other organizations like the BPML.org, the Workflow Management Coalition (WfMC), the OMG, and the United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT) were also pursuing similar goals. In the following paragraphs, we will briefly overview the minor standards: BPML, Web Services Conversation Language (WSCL), XPDL, and ebXML.

Business Process Modeling Language (BPML) In 2000, Intalio initiated the BPMI.org, a non-profit organization that endeavors to promote the standardization of business processes. They coherently developed three languages for BPMSs: BPMN as a graphical notation (see chapter 4), BPML to model executable business processes, and Business Process Query Language (BPQL) as a management interface to the infrastructure. BPML was released in 2002 as an XML-based meta-language for the modeling of business processes, built on top of the Web services stack. It is defined as “a formal model for expressing abstract and executable processes that address all aspects of enterprise business processes, including activities of varying complexity, transactions and their compensation, data management, concurrency, exception handling and operational semantics.” (see Arkin 2002) In June 2005, the BPMI.org and the OMG announced the merger of their BPM activities and thus collaborating on the development of these languages. While BPML and BPQL were faded into the background, BPMN became a growing and highly significant force in the structuring of process description.
Web Services Conversation Language (WSCL)  

WSCL was developed by Hewlett-Packard in 2002, at the same time as WSFL and XLANG. It was submitted to the W3C at the end of that year for standardization and further development (see Banerji et al. 2002). WSFL, another language that describes business processes in an XML-based format, also relies heavily on the Web services stack. It defines the external, visible behavior of a Web service as an abstract interface from the perspective of one of the participants. It focuses on public processes, the definition of messages and the order in which these are exchanged, and may not implement any business logic or private processes. As a consequence, WSCL supports both the concepts of choreography and orchestration. To elaborate further, there are four main elements to a WSCL specification: document types are used to specify the schemas of documents being exchanged; interactions model the actions of the conversation between multiple participants; transitions specify the order of relationships between interactions, and; conversations contain all interactions and transitions plus some additional information for the defined conversation. Overall, it was primarily used by Hewlett-Packard itself. One of its drawbacks and reasons for failure was its sequential manner
of process execution which was inadequate for the needs of current applications.

**Electronic Business using Extensible Markup Language (ebXML)** EbXML is not a standard itself but refers to a family of standards, the purpose of which is to provide an XML-based and open infrastructure for the global use of electronic business information. It was published as an initiative by OASIS and UN/CEFACT in 1999. As of today, ebXML contains the following standards: ebXML Collaborative Partner Profile Agreement, ebXML Messaging Service Specification, ebXML Registry Information Model, ebXML Registry Services Specification, and ebXML Core Components Technical Specification. All these have been approved as OASIS and ISO 15000 standards. An important consideration is ebXML’s developments towards ebXML Business Process Specification Schema (ebBP) or Business Process Schema Specification (BPSS) which were published in April 2006 (see Dubray et al. 2006). It defines a standard language to configure systems for business collaboration execution between partners. It is thus a composition language. It is similar in scope to the previously explained specifications but differs in the fact, that it is not restricted to the Web services stack. EbBP specifies Business Transactions, the choreography for using these in Business Collaborations and the collaborations themselves.

**XML Process Description Language (XPDL)** Outside the Web services domain, there have been other initiatives to standardize the specification of executable business processes. The initiative of the WfMC is the most notable amongst these. Since 1993, the WfMC has been active to standardize both a workflow process definition language and the interfaces between various workflow components. In August 2002, the WfMC released XPDL to support the exchange of workflow specifications between different workflow products. The most current version 2.0 was published in October 2005. XPDL provides an XML file format that may be used to interchange process models between tools. It incorporates a unique concept regarding lines which are guided by points which lend the line a particular path. The nodes and lines have attributes which specify executable information such as roles, activity descriptions, timers, Web service calls, etc. (see WfMC 2006). It has provision for extensions for use with BPMN (see section 4.5).
Part III

SOA Implementations
7 Approaches to SOA Implementation

After analyzing the separate technologies and standards, mainly in the domain of Web services, we now turn to a discussion of approaches for a complete SOA implementation. The chapter begins with a description of Enterprise Service Bus (ESB) as one of the many abstract methods for SOA realization. Thereafter, we discuss two concrete SOA programming models. First, we look at Sun’s efforts to build SOA-compliant applications and the relevant aspects of the framework, Java Enterprise Edition (EE) 5. More specifically, Web service support and Java Business Integration (JBI) will be the primary areas of concern. Then we examine another SOA programming model, brought to life by a collaboration of industry leaders, known as Service Component Architecture (SCA). We then investigate its fundamentals and related technologies. This is followed by a brief mention of other approaches. Finally, we scrutinize the aforementioned programming models in greater detail.

7.1 The Enterprise Service Bus (ESB)

An ESB is a software architecture construct that provides distributed middleware infrastructure consistent with the principles of SOA. It thus, represents a way to realize an SOA. This concept was brought to life in 2002 by the concentrated efforts of IBM. Since then, the ESB approach towards integration has widely been adopted by other significant vendors in the middleware, integration, and Web services markets, and continues to grow steadily.

An ESB has four primary facets that enable service interaction capabilities. These are:

- messaging,
- intelligent routing,
- transformations, and
- Web services.

It also provides a centralized system control and integrates enterprise-level QoS as well. As the name indicates, an ESB uses a bus architecture. The usage of a bus architecture requires links from each object to the bus directly instead of implementing large numbers of point-to-point connections between several business objects. It is thus, easily extensible since additional connections can be added without any difficulty. A highly distributed, multi-protocol, standards-based message bus constitutes the underlying technology of an ESB. Figure 21 shows a high-level view of the ESB. ESBs apply to today’s standards, especially those established in the Web services domain, as described in the previous chapters.

An ESB embodies a unified concept of the three major styles of enterprise integration:

- SOAs which have applications that communicate through reusable services with well-defined, explicit interfaces,
7.1 The Enterprise Service Bus (ESB)

• message-driven architectures in which, applications send messages through the ESB to the receptor applications, and

• Event-Driven Architectures (EDA) which involve applications that generate and consume messages independent of each other.

ESBs have evolved from several conventional integration approaches: Application server, Enterprise Application Integration (EAI), and Message-Oriented Middleware (MOM). Application servers and EAI use a hub-and-spoke architecture, that requires every message to be sent across a central hub that distributes the information to the correct endpoint. Its benefits include centralized functions, such as management of routing logic and business rules, it however, does not scale well across departmental or business unit boundaries. Application servers interoperate through standard protocols. These servers also link functionalities in a tightly-coupled fashion and intertwine the integration and application logic together. Earlier, EAI was the most important approach for business integration along the complete value chain, which could then be distributed between different applications and platforms. Such EAI brokers provided increased value by separating the application logic from the integration and process routing logic. EAI still suffers from the hub-and-spoke architecture. In contrast, MOM uses a distributed integration architecture, in which applications may be connected in a loosely-coupled, asynchronous way. This approach, however, has two major disadvantages. It still requires low-level coding in an application and its integration logic is hard-wired and intertwined with the application logic, thus limiting the distributed characteristic of MOM. Finally, in an ESB, services may be configured instead of being coded. Process flow and service invocations may transparently span the entire distributed bus. An ESB provides a highly-distributed integration environment that extends well beyond the reach of hub-and-spoke architectures. It provides a clear separation of business logic and integration logic, for instance routing and data transformation (see Chappell 2004).

There are a lot of definitions for an ESB. For example, IMB describes an ESB as follows (see Keen et al. 2004): “We describe the Enterprise Service Bus as providing a set of infrastructure capabilities, implemented by middleware technology, that enable the integration of services in an
SOA”. Forrester Research gives a slightly different definition and defines an ESB as an “Infrastructure software that makes reusable business services widely available to users, applications, business processes, and other services” (see Vollmer a. Gilpin 2006). On the other hand, Dave Chappell defines the ESB concept as:

The ESB concept is a new approach to integration that can provide the underpinnings for a loosely coupled, highly distributed integration network that can scale beyond the limits of a hub-and-spoke EAI broker. An ESB is a standards-based integration platform that combines messaging, web services, data transformation, and intelligent routing to reliably connect and coordinate the interaction of significant numbers of diverse applications across extended enterprises with transactional integrity (see Chappell 2004).

However, Gartner Research provides a more comprehensive definition:

Enterprise service buses (ESBs) are a new kind of middleware that combines features from several previous types of middleware into one package. ESBs support Web services by implementing Simple Object Access Protocol (SOAP) and leveraging Web Services Description Language (WSDL) and Universal Description, Discovery and Integration (UDDI). Many ESBs also support other communication styles that involve guaranteed delivery and publish-and-subscribe; those that don’t soon will. All ESBs provide some value-added services beyond those found in basic communication middleware, such as message validation, transformation, content-based routing, security, service discovery for a service-oriented architecture (SOA), load balancing, and logging. Some services are built into the ESB core, while others run in “plug in” modules. ESBs have a distributed architecture wherein some services are executed near the application programs, rather than in a central hub. ESBs support Extensible Markup Language (XML) and often also support other message formats (see Schulte 2003). and logging. Some services are built into the ESB core, while others run in “plug in” modules. ESBs have a distributed architecture wherein some services are executed near the application programs, rather than in a central hub. ESBs support Extensible Markup Language (XML) and often also support other message formats (see Schulte 2003).

Clearly, the definition of an ESB is rather ambiguous. To set a common basis, Keen et al. (2004) provide a listing of minimal capabilities that an ESB must meet. Also, Chappell (2004) defines the most universally accepted characteristics of an ESB as follows:

- **Pervasiveness:** An ESB must be able to suit the needs of general-purpose integration projects across a variety of integration situations. It must be capable of building integration projects that may span an entire organization and its business partners.

- **Highly-distributed, event-driven SOA:** Loosely-coupled integration components must be deployed on the bus across widely distributed geographic deployment topologies, yet must be accessible as shared services from anywhere on the bus.

- **Selective deployment of integration components:** Adapters, distributed data transformation services, and content-based routing services must be selectively deployed when and where they are needed, and must be independently scaled.

- **Security and reliability:** All components that communicate through the bus may take advantage of reliable messaging, transactional integrity, and secure authenticated communications.
7.1 The Enterprise Service Bus (ESB)

- **Orchestration and process flow**: An ESB must allow data to flow across any application or service that is plugged into the bus, whether local or remote.

- **Autonomous yet federated managed environment**: An ESB must support local autonomy at a departmental and business unit level and must still be able to function in a larger-managed integration environment.

- **Incremental adoption**: Each individual project may merge into a much larger integration network, which may be remotely managed from anywhere on the bus.

- **XML support**: An ESB must take advantage of XML as its “native” data-type.

- **Real-time insight**: An ESB must provide the underpinnings to enable real-time insight into live business data. BAM must be built into the ESB fabric.

Pursuing these characteristics, an ESB may be constructed out of a set of components. According to (Vollmer a. Gilpin 2006), such components include the following:

- **Communication infrastructure**: The ESB may provide support for multiple communication abstractions to enable different service interaction protocols. Also, an ESB may offer the adaption to various connection types.

- **Request routing and version resolution**: This contains the major functionalities of the role played by the ESB in mediating and facilitating different routing methods. This may also include the handling of routing based on a specific version request.

- **Transformation and mapping**: This component may provide simple mapping of data formats for messages as well as aggregation or decomposition of service semantics so that each participant of the conversation is unaware of the details of the other one.

- **Service orchestration, aggregation, and process management**: An ESB may support the orchestration of services and hence, smaller services may be aggregated into larger services. This may also require transaction management.

- **Transaction management**: An ESB may provide an event framework that compensates transactions that “undo” previously performed transactions. This ensures that the result is consistent with the business rules. Since standards regarding SOA transactions are still under development, this is an evolving component of the ESB.

- **Security**: In most ESBs, existing security infrastructure is used for identification, authorization, access control, and others as required. However, an ESB may also enforce security policies which, in turn, require an additional infrastructure for policy definition and management.

- **Quality of Service**: To provide the QoS required by the requester, the ESB must ensure reliable networks and services.

- **Service registry and metadata management**: When the ESB maintains a service discovery, it may extend existing service metadata to enable services to be classified and to allow reuse. UDDI v3 support may be a minimum requirement.

- **Extensibility for message enrichment**: Extensions for messages may be provided through a declarative mechanism.
7 Approaches to SOA Implementation

- **Monitoring and management**: Each ESB may provide automated management as much as possible. However, it may also be necessary to include the human element to investigate certain problems in such systems. For this reason, an appropriate monitoring and management environment must be available.

- **Support for the service life cycle**: In addition to ESB, tools for the software development cycle may be provided by the vendor. This way, services may be tracked through their complete life cycle including development, reuse, integration, deployment, management, and optimization.

ESB has thus emerged as a central method of implementing SOA with its resilient functional capabilities as its most outspoken proponents. Current leading vendors are Cape Clear Software and BEA Systems Vollmer a. Gilpin (see 2006). Other vendors include Software AG, IBM, and IONA Technologies. However, since the concept of the ESB was first conceived by IBM, it has expanded into the realms of abstraction and no precise definition for it exists now. As a consequence, all that remains of ESB is a mere reflection of the initial concept as it underwent tremendous changes upon adoption by different parties. Herein lies a major disadvantage of this concept, since standardization is an important prerequisite for the long-term sustenance of an implementation framework. Nevertheless, the ESB technology is an evolving entry point for implementing SOA.

We will now turn to the importance of standardized SOA implementation methods, as we proceed to the following sections. This perhaps, will elucidate further the role of standardized implementations as a decisive turn in SOA’s evolution.

7.2 Java Enterprise Edition (EE) 5

In the previous chapter, we discussed the notable limitations in the use of an ESB in order to implement an SOA. Java EE 5 provides a more suitable approach to realize that purpose by adding a new dimension by the use of specifications.

7.2.1 Java EE 5 Web Services

This composite package of specifications offers full support for Web services, wherein, each Java component may be deployed in the form of a Web service. A Java EE Web service is defined by “JSR-109”. This specification describes the deployment of web services within Java EE 5 applications. It also defines the method of implementing Web service endpoints i.e. the business logic of the Web service. Such an implementation employs the use of Enterprise Java Bean (EJB) components. To be more precise, “JSR-109” describes how a WSDL port/endpoint may be bound to an EJB implementation. Also, the primary Application Programming Interface (API) used in Java EE 5 for Web service calls is implemented with Java API for XML Web Services (JAX-WS). This specification surpasses its predecessor Java API for XML-based Remote Procedure Calls (JAX-RPC) 1.1 with the additional support of several transport protocols for instance, SOAP and XML (see chapter 5). Finally, all these aforementioned concepts in combination with JBI, a standardized integration platform between business applications and Java, result in a fundamental technology that makes the realization of an SOA possible. We will discuss JBI in greater detail in the following section.
We now look at the process as a whole. First, on the server side, business logic is written as an EJB implementation which is annotated as a Web service. While packaging and deploying of this EJB, a WSDL file is generated according to the JAX-WS specification. This service is published such that it is accessible to clients. The implementation on the client’s side, independent of technology restrictions, may then access the Web service through an interface defined by JAX-WS.

### 7.2.2 Enterprise Java Beans (EJBs)

This section describes the realization of persistent data retention, the definition and implementation of operations to access this data, and the exposure of Web services to invoke these operations. The technology to accomplish these tasks is EJB. EJBs encompass standardized components residing in a server that runs Java 2 Platform, EE 1.3 or higher. The components facilitate development of multi-tier software systems by means of Java. EJBs are deployed in an EJB container that runs within the application server. Interaction between EJBs and their container as well as interaction between client and EJB/container combination is described in the specification (see DeMichiel a. Keith 2006) which is available in version 3.0 since May 2006.

Important concepts provided by EJBs are persistence, business processing, transaction processing, and distributed processing capabilities for enterprise applications. To supply such concepts, different types of EJBs exist addressing different use cases. All of them can be accessed either locally that means within the domain of one virtual machine or remotely across process boundaries as well as business boundaries. These different types are Entity Beans, Session Beans and Message Driven Beans. These different types of EJBs are discussed subsequently.

Persistent data within the system are modeled through *Entity Beans*. This data may model information structures, such as invoices or addresses. Entity Beans can be used to represent one data record of a database. The persistence is hereby obtained either by the developer of the bean itself which is called Bean Managed Persistence or can be provided by the EJB container which is referred to as Container Managed Persistence. In case of using the latter, deployment descriptors must define an abstract schema.

*Session Beans* are used to realize business logic and to process persistent data. Methods offered by these beans are executable by client applications. Implementations of these methods are specified in the Java implementation class, which is the bean class. There are two distinct types of Session Beans:

- **Stateful** Session Beans that are able to store information which may be used in subsequent method executions. Therefore, each stateful Session Bean has an unequivocal identifier assigned.

- **Stateless** Session Beans cannot be distinguished from each other, hence they have no identity and cannot store information. All information, required to process a task had to be passed via one method call. For the purpose of using Web service calls, only Stateless Sessions Beans can be used and exposed.

The last EJB type is the *Message Driven Bean*. Similar to the Session Bean, it provides asynchronous communication, but is responds to JMS messages rather than an Remote Method Invocation (RMI) events. This integration of JMS with EJB creates a new type of bean designed to handle asynchronous communication.
Each of the three EJB types must provide a Java implementation class, also known as bean class, as well as two Java interfaces, the Home Interface and the Remote Interface. The Java implementation class is used by the EJB container to create instances of beans and to provide the actual EJB implementation. The interfaces are used by the client code.

The EJB standard implementation defines a standard set of useful methods. These interfaces provide a common base for all Remote as well as Home Interfaces. To increase performance for local calls between EJBs within one virtual machine, as of EJB version 2.0, both these interfaces also exist as local interfaces, thus there are actually four interfaces (see DeMichiel a. Keith 2006). The EJB container generates classes for these interfaces acting as proxies for the client. If a client invokes methods on generated proxies, these proxies place method arguments into a message and send it to the EJB server. The corresponding method is invoked on an instance of the Java implementation class by the server.

Beside the actual beans, the EJB standard describes files known as Deployment Descriptors. These descriptors consist of XML files defining properties of an EJB. These properties include:

- name, class and interfaces of an EJB,
- restrictions and preconditions on methods the EJB offers,
- references to resources that have to be provided by the EJB container such as data sources,
- references to other EJBs or Web services,
- definitions for endpoints of Web services that are used by the EJB, and
- particular for Entity Beans with Container Managed Persistence, where persistence is handled by the EJB container, the name of an abstract schema as well as the definition of persistent fields and relations to each other are referenced in deployment descriptors.

Subsequently, we will proceed to a discussion of the JBI standard.

### 7.2.3 Java Business Integration (JBI)

JBI is a standardized platform that entails a service-oriented approach for the integration and structuring of enterprise functions. It creates a standard meta-container allowing several components to be plugged in to interoperate seamlessly. This framework provides a dynamic composition and deployment of loosely-coupled applications. This specification was pioneered by Sun Microsystems as “JSR-208”. Its first version was published in August 2006 (see Ten-Hove a. Walker 2005).

The mosaic composite of the JBI environment, depicted in Figure 22, is broken down into pieces for our consideration. These may be listed as (see Ten-Hove a. Walker 2005; Raj et al. 2006):

1. Service Engines,
2. Binding Components,
3. the Normalized Message Router, and
4. the JBI Runtime Environment or Meta-Container.
Of these pieces, two represent JBI components. These are Service Engines and Binding Components. Service Engines enable pluggable business logic and other services such as, processing, transformation, and routing services. Examples of Service Engines include the WS-BPEL Service Engine which orchestrates services and an EJB Wrapper that defines reusable Web services. While Service Engines deal with pluggable services, Binding Components enable pluggable external connectivity. They provide access to both internal and remote services which in turn become independent of transport and protocol. This decouples the service implementation from access mechanisms like SOAP, SMTP, or JMS (see section 5.2). The messages that form an integral part of this arrangement are normalized and de-normalized while being exchanged between clients.

The Normalized Message Router acts as a bus that directs normalized messages from source to destination components according to specified policies. It provides the basic message exchange infrastructure including reception, routing, redirecting of message data. The message exchange between JBI components and the Normalized Message Router are done through JBI normalized messages via the Delivery Channel. Such messages are XML documents that specify meta-data like context information, security tokens or data for other components. These messages also contain the message payload.

The final piece of the JBI environment that we look at is the JBI Runtime Environment or the JBI Meta-Container. This hosts the two JBI components, Service Engines and Binding Components, and additionally the Normalized Message Router. When messages are exchanged with external entities i.e. entities outside the Meta-Container, Service Engines communicate using the Normalized Message Router and the Binding Component. On the other hand, communication within a JBI environment is relatively simpler since all messages are already normalized and in standard WSDL 2.0 format.

JBI uses JBI Message Exchange Patterns, in which the interaction between a service provider and a service consumer is characterized by the WSDL messaging model. The following four patterns are supported by JBI:
7 Approaches to SOA Implementation

- **In-Only** for one-way message exchange,
- **Robust In-Only** for reliable one-way message exchange,
- **In-Out** for two-way message exchange, and
- **In Optional-Out** for two-way message exchange with optional provider responses.

To take a higher view of the above described JBI environment, we now turn to Service Units and Service Assemblies. A Service Unit describes which services are provided and consumed by a JBI component wherein each Service Engine and Binding Component represents a Service Unit. A set of Service Units may then be packaged into a Service Assembly which describes the service connection which inherently includes the mapping between the consumer and the provider endpoints. This Deployment Descriptions of both the Service Units and the Service Assembly are stored in jbi.xml files.

Since the concept of JBI is crucial to the understanding of SOA implementation through Java EE 5, we will now briefly discuss some significant points. Regardless of the implementation technology, multiple services may be wrapped as a Service Unit and may then be packages in a loosely-coupled manner into a Service Assembly. Messages exchanged between various Service Engines like BPEL Engines or external Web services, are normalized through the use of Binding Components. Message routing is handled by the Normalized Message Router. JBI proves its flexibility and resilience by allowing extensions for management, monitoring, and other enhancements. Overall, this framework models features similar to those in an ESB, which we have discussed in section 7.1. Despite being an open specification, JBI is not widely adopted in the business world. An open source project, pioneered by Sun Microsystems known as Project Open ESB\(^1\), fully implements JBI for the construction of a powerful ESB. It improves JBI with a single point for system administration and management and provides location transparency. Other implementations of the JBI specification include Apache ServiceMix\(^2\) and Mule\(^3\).

### 7.3 Service Component Architecture (SCA)

The creation of a service-oriented application today, presents certain challenges. These challenges arise from the vast number of diverse technologies, each with its own paradigm and programming model, which must all be mastered to assemble an integrated whole. Most of the existing component models were initially designed for object-oriented and not service-oriented purposes. While a distributed object-oriented application creates a tightly-coupled collection of object components that interact through technology-specific protocols, a distributed service-oriented application represents a loosely-coupled assembly of service components which are technology independent. These components are independent of programming language, framework, protocol, communication technology, and implementation. Moreover, the existing object-oriented models require much infrastructure-level coding which take the developer’s focus away from writing applications. SCA came into being to address these issues.

SCA is a set of specifications which describes a model for building applications and systems based on SOA. It extends and complements prior approaches which are then equipped to implement reusable services. Another important feature of SCA is that it is built on the firm

\(^1\)see [https://open-esb.dev.java.net/](https://open-esb.dev.java.net/), last checked 24/11/2006


\(^3\)see [http://mule.mulesource.org/wiki/display/MULE/Home](http://mule.mulesource.org/wiki/display/MULE/Home), last checked 24/11/2006
foundation of open standards. It is widely believed that SCA does not directly compete with other existing programming models since it can act as an extension to them. However, it can be said that SCA poses a competitive threat to Sun’s Java EE 5 (see section 7.2) in that Java EE 5 provides exceptionally good support for Web services but is still language dependent and thus skill intensive. For these reasons, SCA endangers the success of Java EE 5’s integration framework, JBI, in particular by providing a non-Java-specific, metadata-driven model to describe the composition of services.

To make SOA implementation more substantial, SCA works together with Service Data Object (SDO), a specification for data handling. It provides a common basis for the access and manipulation of various kinds of data from heterogeneous data sources, such as relational databases, XML data sources, and Web services.

The first draft of the SCA specification in version 0.9, along with SDO version 2.0, was released in November 2005. A consortium including BEA Systems, IBM, Iona Technologies, Oracle, SAP, Siebel Systems, Sybase, and Xcalia led its development at the time. Since then the initiative has steadily grown. In July 2006, nine new partners joined the collaboration. These are Cape Clear, Primeton Technologies, Progress Software, Red Hat, RogueWave Software, Software AG, Sun Microsystems, and TIBCO Software. At the same time, a community “Open Service Oriented Architecture” was formed. Currently available specifications in the form of version 0.95 contain several additions and improvements to the older version 0.9. The release of version 1.0 of the SCA set of specifications is scheduled for January/February 2007.

In the following section we look at SCA in further detail by discussing each element that comprises the set of specifications. We will also briefly examine SDO. The whole chapter concludes with a discussion on SCA and other component models in section 7.4.

### 7.3.1 SCA Fundamentals

SCA encourages the creation of business logic in the form of discrete components. Each component may offer its capabilities through a service-oriented interface, called a *Service*. The components utilize functions provided by other components, either local or remote, through service-oriented interfaces called *References*. SCA proposes the division of the process of constructing a service-oriented application in three distinct steps:

1. The *service construction* or implementation of all components including the definition of necessary Service and Reference interfaces. This implementation may be based on any programming language which includes object-oriented, procedural, and declarative languages as well as, XML-centric languages.

2. The *service assembly* to build complete business applications through the wiring of References and Services. The assembly is independent of the implementation language.

3. The *deployment* of the constructed implementation and assemblies.

Within the set of SCA specifications there exists a Client and Implementation specification for each supported implementation language. The assembly of several implementation components is described in the Assembly Model. The specification set further contains the definition of a declarative Binding model for multiple access methods, which may be used to invoke services. Furthermore, support for the specification of constraints, capabilities, and QoS expectations, a separate Policy Framework is inherently defined in the set. All of these specifications, either

---

4 see [http://www.osoa.org](http://www.osoa.org), last checked 24/11/2006
technology-neutral or technology-specific, and their relation to external specifications may be grouped into three layers as depicted in Figure 23.

![Figure 23: SCA Set of Specifications.](image)

### 7.3.1.1 SCA Assembly Model

The Assembly Model provides a unified, language-independent method, through the use of XML, to expose implementations as services. This specification covers a model for the assembly of services, both tightly and loosely-coupled, as well as a model for applying infrastructure capabilities to services and to service interactions. The following detailed description is based on this specification (see Beisiegel et al. 2006a).

The SCA Assembly Model consists of a series of structures that jointly define the configuration of an SCA system, in terms of service components and their connections to other components. There exist three structures that compose a complete service-oriented application:

- Component,
- Composite, and
- System.

A Component as depicted in Figure 24 consists of a configured instance of an Implementation that represents a business function. Due to this configuration, more than one Component may use the same Implementation. SCA has a provision for a wide array of implementation technologies, such as Java, BPEL, or C++. Other implementation types may be introduced by the developer through a compliant extensibility mechanism, inherently present in SCA. The business function is offered for use by other Components in the form of Services. Implementations may be functionally dependent on other Components, known as References. Components may also have Properties that may be set to influence the operation of the business function. By utilizing
these Services, References, and Properties, a Component acquires flexibility in its configuration that makes it possible for the developer to shape it appropriately.

Composites describe the content and linkage of an application. These group and link Components built from varied implementation technologies, allowing appropriate technologies being used for each business function. A Composite may be used as a complete implementation of one or several Components. Such Composite implementations may be used as Components in other Composites, which in turn make hierarchical structures possible. Composites model the basic unit of composition for a tightly-coupled construct, which is in turn deployable in an SCA System. Composites may contain the following items: Components, Services, References, Properties, and Wires. Wires connect components through their References and Services. All these items are discussed in greater detail in the next paragraph. Figure 25 depicts how these items may be assembled within a Composite. It is interesting to note that the Properties of the Composite are the summation of the Properties of all contained Components. For each Composite, an XML configuration file, containing its structural details, is created at the time of its inception, somewhat like a deployment descriptor.

An SCA Service is used to publish services provided by a composite, so that the implementation of a business function may be used by external clients. The set of operations provided by a Service is defined through an Interface. Currently, two Interface types are supported by SCA which are in turn provided by a Service and consumed by a Reference. These are the Java Interface and the WSDL Interface. Services use Bindings to describe the way in which they are published and through which access mechanisms the Service may be invoked by external clients. At present, SCA offers Bindings to Web services, other SCA Composites, stateless session EJB, and JMS. This list, however, is not exhaustive. SCA also provides an extensibility mechanism to create new Binding types. An SCA Service may define its provided characteristics through Policies (see section 7.3.1.3). There are two types of Services, Remotable Services and Local Services. While a Remotable Service is designed to be published and used by other Composites or Components in a loosely-coupled manner, a Local Service may only be used within the same Composite.

An SCA Reference represents Services that are external to the Composite. These may be accessed within the Composite like any other Component Service. It also uses Bindings to describe the access mechanisms to external Services, like Web services and JMS. Again, Binding types are extensible by the developer. Components may use the existing functionalities described in the Interface of the Reference. The Reference may define the required characteristics through Policies.

An SCA System, as shown in Figure 26, represents the complete run time assembly of any
number of Composites. It typically represents an area of business functions controlled by a single organization. A System, like a Composite, is a set of Services, References and Wires. Unlike Composites, however, a System does not have an XML configuration file describing its contents. Its configuration and structure may be inherently defined together by the discrete Composite configuration files deployed into the System. Another difference between Systems and Composites is that Systems assemble Services in a loosely-coupled manner. Also, a System is open to addition and removal of Composites through deployment and undeployment actions.

Table 7 summarizes all of the mentioned SCA structural elements which are presented according to the order of the letters of the alphabet. A sample application including these SCA elements is explained in the subsequent chapter. Code snippets out of configuration files and interface definitions are also present.

The SCA Assembly Model supports three basic Message Exchange Patterns:
### 7.3 Service Component Architecture (SCA)

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binding</td>
<td>Bindings are used by Services and References. References use Bindings to describe the access mechanism used to call the Service to which they are wired. Services use Bindings to describe the access mechanism(s) that clients should use to call the Service.</td>
</tr>
<tr>
<td>Component</td>
<td>SCA Components are configured instances of SCA Implementations, which provide and consume Services. SCA allows many different implementation technologies such as Java, BPEL, C++. SCA defines an extensibility mechanism that allows you to introduce new implementation types.</td>
</tr>
<tr>
<td>Composite</td>
<td>An SCA Composite is the basic unit of composition within an SCA System. An SCA Composite is an assembly of Components, Services, References, and the Wires that interconnect them. Composites can be used to contribute elements to an SCA System.</td>
</tr>
<tr>
<td>Implementation</td>
<td>An Implementation is a concept that is used to describe a piece of software technology such as a Java class, BPEL process, XSLT transform, or C++ class that is used to implement one or more Services in a service-oriented application.</td>
</tr>
<tr>
<td>Interface</td>
<td>Interfaces define one or more business functions. These business functions are provided by Services and are used by Components through References. Services are defined by the Interface they implement.</td>
</tr>
<tr>
<td>Local Service</td>
<td>Local Services are Services that are designed to be only used “locally” by other Implementations that are deployed concurrently in a tightly-coupled architecture within the same operating system process.</td>
</tr>
<tr>
<td>Property</td>
<td>Properties allow for the configuration of an Implementation with externally set data values. The data value is provided through a Component, possibly sourced from the Property of a containing Composite.</td>
</tr>
<tr>
<td>Reference</td>
<td>SCA References represent a dependency that an Implementation has on a Service that is supplied by some other Implementation, where the Service to be used is specified through configuration. In other words, a Reference is a Service that an Implementation may call during the execution of its business function.</td>
</tr>
<tr>
<td>Remotable Service</td>
<td>A Remotable Service is a Service that is designed to be published remotely in a loosely-coupled SOA architecture.</td>
</tr>
<tr>
<td>Service</td>
<td>SCA Services are used to declare the externally accessible Services of an Implementation. For a Composite, a Service is typically provided by a Service of a Component within the Composite, or by a Reference defined by the Composite.</td>
</tr>
<tr>
<td>System</td>
<td>An SCA System represents a set of Services providing an area of Business functionality that is controlled by a single organization.</td>
</tr>
<tr>
<td>Wire</td>
<td>SCA Wires connect service References to Services.</td>
</tr>
</tbody>
</table>

Table 7: Terms used in the SCA Assembly Model (see Beisiegel et al. 2006a).

- **Synchronous Request/Reply**, where the service consumer receives an immediate reply to the request, from the service provider.
- **Asynchronous One-Way**, where the service consumer only sends a message to the service provider.
Approaches to SOA Implementation

provider without expecting a reply.

• Asynchronous Callback, where the service consumer sends a request to the service provider who may issue a callback at a point later in time.

### 7.3.1.2 SCA Binding Model

In the previous section, we already looked at Bindings as part of a Composite within an assembly. As mentioned there, Bindings allow Services to be provided through specific access mechanisms and References to be satisfied via particular access methods. Each technology calls for its own Binding and thus, SCA currently specifies certain Bindings for its requirements. These are currently under development and are only available as drafts at this point of time. They are:

- Web service Bindings,
- Messaging and JMS Bindings,
- Enterprise Information System (EIS)\textsuperscript{5} and Java EE Connector Architecture (JCA)\textsuperscript{6} Bindings, and
- Stateless EJB Session Bean Bindings.

We now take a closer look at the Web service Binding as representative of all Binding types. The Web service Binding is also used in the sample application explained in the subsequent chapter.

The Web service Binding specification (see Holdsworth et al. 2006) defines the manner in which a Service may be made available as a Web service and how a Reference may invoke an existing Web service. As described in section 5.4.1, Web services are usually defined through WSDL. This Binding supports both versions of WSDL, version 1.1 and 2.0. It provides a way to integrate existing as well as synthetic WSDL ports/endpoints. In the former, all the feature and functionalities are available to the user. In the latter, the Web service technology is used without the need to define a complete WSDL document. Here, it is sufficient to clarify some necessary information in order to create a WSDL document. For both options, the information within the WSDL file is enough to invoke the Web service. The Web service Binding provides the fault for SOAP (see section 5.3.1, both versions 1.1 and 1.2, for use as the underlying transport mechanism.

This Binding introduces two new elements to be placed into the Service and Reference description which are defined within the configuration file of each Composite. The Web service Binding element binding.ws points to an existing WSDL port/endpoint. The SOAP Binding element soapbinding omits the WSDL port/endpoint altogether and specifies information needed to instead create a synthetic WSDL port/endpoint. An example for Web service Binding using SOAP is given in the following chapter.

\textsuperscript{5}An Enterprise Information System is any kind of computing system within an organization that provides a technology platform that integrates and coordinates their business processes.

\textsuperscript{6}The Java EE Connector Architecture is Java-based technology solution for connecting application servers and EISs as part EAI solutions.
7.3 Service Component Architecture (SCA)

7.3.1.3 SCA Policy Framework

The first draft of the Policy Framework, published in November 2006 (see Beisiegel et al. 2006b), deals with non-functional requirements which have an impact on SCA throughout the life cycle of Components and Composites. This framework provides support for the specification of constraints, capabilities and QoS expectations. It is based on WS-Policy and WS-PolicyAttachment.

The SCA Policy Framework model consists of Intents and Policy Sets. An Intent allows the developer to define abstract QoS capabilities independent of any particular technology in order to implement such a capability. In this manner, security, transactions, and reliable messaging characteristics may be incorporated into the assembly of Components and Composites. Intents may be used to express Interaction Policies or Implementation Policies (see Figure 27). Interaction Policies express capabilities and requirements for Services and References respectively. These govern the communication between the service consumer and the service provider. On the other hand, Implementation Policies express constraints that affect the contract between Component and its runtime environment, such as monitoring, logging, and access control.

The SCA Policy Framework defines a model that hides the complex set of assertions in existing policy specification. The abstract SCA policies are in turn, mapped to concrete policy artifacts defined by existing specifications like WS-Security.

In some cases, it is possible to define an Intent that is only completely satisfied when all members of a set of lower-level Intents are fulfilled. Such an Intent is referred to as a Profile Intent. It may be used in the same way in which any other Intent is used. SCA inherently provides a set of core Profile Intents representing regular QoS requirements like basic security and reliable messaging.

Another key concept of this model is the Policy Set. Policy Sets contain concrete policy and policy subject pairings and declare Intents that are realized through the Policy Sets collectively. A Policy Set is constructed out of Intent Maps. These define concrete policies and policy subjects that are tied to a specific Intent. These declare the defaults and fixed values for alternatives in a single domain. A Policy Set may also contain a direct link to a policy attachment. Furthermore, a Policy Set may refer to other Policy Sets, which provides a recursive inclusion capability for Intent Maps.

Figure 27: SCA Policy Framework: Policy Intent types.
Intents and Policy Sets may be attached to any global SCA element defined within an SCA Assembly. Policies may thus be specified for Bindings and then used in Policy Sets which are in turn utilized by Components, Services, and References.

7.3.1.4 SCA Client and Implementation Model

After our discussion on the SCA Assembly Model, the SCA Binding Model, and the SCA Policy Framework, we now turn to the Client and Implementation specification. Composite descriptions defined in its configuration file, must be injected into the Implementation of each Component. Therefore, a language-specific specification is necessary to build service clients and service implementations. Due to the separation between the implementation and the configuration, protocol and middleware concerns are removed from the business logic. Furthermore, a late binding to a variety of transports is supported by SCA. Due to this dependency injection, service Implementations may be reused with different attributes. Currently there are Client and Implementation specification for the following languages available:

- Java,
- C++,
- BPEL,
- Hypertext Preprocessor (PHP),
- EJB, and
- Spring.

For further discussions now and also for the implementation described in the next chapter, we specifically discuss the Java Client and Implementation specification due to its comprehensive nature. This specification provides the following features (see Barack et al. 2006):

- Java mappings for all Assembly Model concepts such as “Component”, “Service”, “Reference”, and “Property”;
- Java mappings to implement asynchronous and conversational Services;
- Java mappings for specifying Component life cycle notifications; and
- Java mappings for specifying Service and Implementation scopes.

The specification defines implementation metadata using the annotation capability of J2SE 5. Now we will discuss annotations necessary for dependency injection into Component Implementations. In the case under consideration, for each Component, a Service Interface is built in Java and the Service Implementation is a Java class. The Implementation must now, evidently implement all the operations defined in the Interface. The annotation @Service is used for a Java class to specify that the Service is implemented by it. It is also possible to specify several Interfaces that must be implemented within one class. Whether a Service is remotable or local is defined in the Interface of the Service. In the case of a Remotable Service, the annotation @Remotable may be used. The SCA runtime manages the Component’s state determined by the

---

7 The Spring Framework is a popular platform used to construct Java applications. It aims to reduce the complexity of the programming environment. In particular, Spring provides a runtime container that provides dependency injection so that application components may avoid the need to program directly to middleware APIs.
7.3 Service Component Architecture (SCA)

scope specified by the @Scope annotation in the Implementation class. References and Properties of a Component defined in the XML configuration file are injected into the Implementation by means of @Reference and @Property annotations defined in front of the Service Reference and Property declarations.

With all these characteristics injected from the configuration file into the Implementation, SCA Services may be accessed by other Components. The SCA specification describes two ways to gain access to the Service. When the name of the required Components is known before runtime, Reference injection may be used. Therefore, constructor parameters are typed by the Service Interface and annotated with a @Reference annotation with additional attributes. On the other hand, if the name of the required Component is not known until runtime, the Service may be located using the Component’s Context functions.

Furthermore, the @Session and @Callback annotations are available to realize the Message Exchange Pattern of an Asynchronous Callback. See the referred specification for more details. In addition to this, the following chapter explains a sample application using the Java implementation type.

7.3.2 Service Data Objects (SDO) for Data Handling

In order to further empower this method of implementation, SCA is used in conjugation with SDO for effective data handling. This specification and API are developed by the same community that develops SCA. SDO is currently available in version 2.1 (see Adams et al. 2006). A high-level overview is provided in Beatty et al. (2003).

The motivation behind SDO is that through simplification of data programming, the developer may focus entirely on business logic instead of the underlying technologies. This is achieved through the unification of data programming across various data source types. It also enables applications, tools, and frameworks to easily query, view, bind, update, and introspect data. For example, SDO does not presume a particular query language or backend storage. Thus, Standard Query Language (SQL) or XPath or any other query language may be used for this purpose. The data storage may exist in the form of a relational database, object database or an XML data source.

The key concepts in the SDO architecture are as follows:

- Data Objects,
- Data Graphs, and
- Data Access Service (DAS).

Data Objects hold the actual data which includes any amount of named properties which may represent a simple data type or a reference to other Data Objects. Data Graphs are a set of data that provides the transportation units between components. Data Graphs may record all changes to data. Metadata regarding Data Objects and Data Graphs enables tools to better analyze the data.

The underlying disconnected data pattern foresees that a client retrieves a Data Graph from a Data Source, modifies the Data Graph and then may apply the Data Graph changes back to the Data Source. Access to the Data Source is provided through a mediator, the Data Access Service (DAS). It provides methods to load Data Graphs from storage and to save it back to that location (see Williams a. Daniel 2006). There exists a separate DAS to connect to relational...
databases, XML databases, and other data storages. The complete process is depicted in Figure 28.

![Figure 28: SDO’s disconnected graph pattern.](image)

SDO also provides an API that implements the SDO specification and makes its functionalities available to its developer. Currently, APIs are being developed for Java, C++, and Hypertext Preprocessor (PHP). The publication of SDO in version 3.0 is likely to unleash a comprehensive mediator for data access with additional APIs included.

### 7.3.3 Utilization of SCA/SDO

Even though the SCA and SDO specification are new and still evolving, many industry leaders are currently developing solutions that implement this language-neutral SOA programming model. Most of these vendors, as depicted in Figure 29 are now collaborating in an informal alliance, the “Open SOA Collaboration”, as mentioned previously. There also exist a few open source projects which pursue similar goals. We will now briefly discuss the most relevant of these.

![Figure 29: Partners of the OSOA Collaboration.](image)

IBM, as one of the leaders in SCA implementation, already provides mature and tested solutions for both design time and runtime integration. For instance, their “WebSphere Integration Developer V6.0” provides a common design time environment for all usage scenarios of the SCA technology. It is an Eclipse-based modeling tool which currently supports the SCA Assembly Model of version 0.9, released in November 2005. For a technical review of this tool, refer to [de Putte a. Gavin (2005)](http://www-306.ibm.com/software/integration/wid/) and also, for further clarity, a sample screenshot is displayed in Figure 30. Beside this design time SCA integration, three of IBM’s runtime solutions also support

---

7.3 Service Component Architecture (SCA)

SCA. IBM’s “WebSphere Process Server”\(^9\) and “WebSphere Application Server 6.1” with the “Feature Pack for SOA”\(^10\) provides a runtime environment to execute various SCA components. Within IBM’s “WebSphere Enterprise Service Bus V6”\(^11\), SCA represent a component model for the mediation module of its ESB. It supports several bindings for import and export, including JMS, Web services, EJB, and SCA. In conclusion, it is a notable achievement for SCA and SDO to be integrated into current solutions provided by IBM to such a vast extent, despite these being such nascent concepts.

![Figure 30: Screenshot of the IBM WebSphere Integration Developer.](image)

Another early implementation of these is in Oracle’s “Event-Driven Architecture Suite”\(^12\). This solution provides a flexible declarative environment to rapidly build and adapt event-driven applications. It is gradually evolving with the SCA specification into a service-oriented tool for the integration of a complex SOA. Other solutions worth mentioning are “Hydra”\(^13\), from Rogue Wave Software, the “SCA Framework for SOA”\(^14\) from Covansys, and Xcalia’s SDO implementation “XIC”\(^15\).

There also exist a few open source projects encouraging the utilization of SCA and SDO. As a design time project under development, the “Eclipse SOA Tools Platform”\(^16\) will provide frameworks and tools that enable the design, configuration, assembly, deployment, monitoring, and management of SOA software. More specifically, it includes an assembly model framework which may be used to connect services based on the SCA Assembly Model. Also, the creation of SCA-compliant services and their development will be supported by the Eclipse SOA Tools.

---


\(^12\)See [http://www.oracle.com/technologies/soa/events.html](http://www.oracle.com/technologies/soa/events.html), last checked 24/11/2006


Platform. Apart from this, a BPEL-to-Java translator and a BPMN modeler are currently being developed. Till now no stable versions of the same are known to be released.

An open source counterpart of notable popularity is “Tuscany”\textsuperscript{17} by Apache which is an SCA runtime application. It represents a roughly complete implementation of most SCA and SDO specifications, both in Java and C++. Refer to section 8.1 for details and a sample application elucidating the Tuscany framework.

It is believed that vendor sanction of SCA and SDO is inevitable upon complete development and as it evolves into a more refined and more powerful form.

### 7.3.4 Business and IT Value of SCA/SDO

The previously described SOA programming model defines business components as SCA Components and the data exchanged amongst them as SDOs. There exists a strict separation between the concerns of defining a component implementation and the usage of a service. This separation is enforced through the combination of SCA and SDO. This allows the developer of business solutions to focus on the actual business objectives instead of the technical details about the method of specific implementations. Clearly, the combined forces of SCA and SDO add immense value to both business organizations and IT in general.

The major outstanding value of this approach lies in its flexibility. SCA permits the combination of several business functions, wrapped as Components, irrespective of their implementation technology. Another reason that contributes to the overall flexibility of this pairing, is that previously defined Components and Services may be reused and inculcated into new processes, in a loosely-coupled manner. Hence, SCA lends the system the adaptability required for fast changing business processes today. It also implies that Components have a modular property which enables them to be modified and replaced without influencing the integrity of the whole system. Furthermore, the business software solutions may not necessarily arise from a single vendor anymore instead specialized Components may be purchased and assembled from different vendors.

According to Weaver (2005), another chief benefit of SOA, and more specifically of the SCA programming model, is a direct impact on programmer productivity since the development of complex business solutions is simplified. As a result of this, the business value also inevitably increases since the time taken for the application development is drastically decreased. It is important to note, that if a development was to begin from scratch, the time taken to minimize errors and develop complex concepts would be highly inefficient. Instead, IT solutions would work much better if previously used and adequately tested services were directly inculcated into the current application. This is what makes SCA such a powerful concept. SCA and SDO together provide a higher realm of abstraction which tends to simplify processes by decreasing the complexity of the used technologies and hence, again leads to more productivity. This allows developers to be more selective and specialize in areas for implementation, alleviating the technical burden upon them. Consequently, such enhancements bring with them enormous business benefits like greater quality of the product and lesser time-to-market. On the whole, all these add up to an increased business value of IT, achieving exceptional business solutions.

\textsuperscript{17}see \url{http://incubator.apache.org/tuscany/}, last checked 24/11/2006
In the current day scenario, the industry is divided on their preference for two approaches to realize SOA. These are JBI and SCA. The implementation used in support of this thesis, as mentioned before, uses SCA. We will elucidate the reasons that led to the selection of SCA as a superior choice to JBI, in the author’s opinion.

According to Jean-Jacques Dubray of SAP Labs LLC (see Dubray), there are three classical approaches of back-end development that finally spawned the beginnings of JBI and SCA. The first method, Invocation, utilizes a common request/response approach using Java EE, but fails to support long running integration scenarios. The second method, Mediation, is based on the mediated message exchange pattern. The Java Community Process published an API to address the need for standardized mediation infrastructure. This was called JBI. It is a set of API’s focusing on building an open, extensible, and modular ESB. The main problems surrounding JBI was its centrally coordinated character. Some companies were unwilling to pay for the substantially extra cost of “mediating” message interchanges. Also, there is no normalization possible between two JBI instances. This leads to restrictions on bindings and results in JBI’s limited scope to small and local problems.

The third approach, Activation, consists of producing components which may be accessed via different middleware. Activation maximizes the autonomy of the components themselves and their ability to be composed with other components; i.e. business logic may be implemented by a component that does not rely on middleware facilities. This is the concept which SCA embodies. A highly coveted aspect of SCA is that it is compatible with all runtime environments. Thus, SCA has more to offer in terms of scope and flexibility. Now, stepping aside from the technical aspects of these approaches, we will briefly look at the industry response to each of these.

The composition standards currently under development are expected to revolutionize SOA as we know it by increasing the speed to a significant extent. Unfortunately, due to dissension amongst vendors, the expected gains in the field have been delayed. Also, there is no clear winner that has emerged as a leader in the area of SOA implementation.

Matters become worse by the emergence of two primary camps, each vehemently lobbying for technologies. Since SOA exists as a concept alone, more than one method may be used to achieve its ideals. The pursuit of this entirely relies upon the final underlying technology of the implementation. Of the two camps, the Java proponents advise JBI, whereas the other camp extends its support for SCA in combination with SDO. JBI is a standard for composing service containers into composite applications. SCA/SDO adopt a broader approach to the composition problem and are able to work with multiple language on multiple platforms. It is interesting to note that while JBI is an established standard, SCA has not achieved that status yet.

The JBI support drive initially, was quite understandably backed by Sun Microsystems and their ally TIBCO. Smaller supporters of this technology are IONA, PolarLake and Software AG. SCA, on the other hand, enjoys the support of IBM and BEA Systems. Minor support in this case features IONA, Oracle, SAP, Siebel Systems and Sybase. In June 2006, several other organizations joined the previously formed OSOA Collaboration, including Sun Microsystems. This was a big setback to JBI, since support for it began to sway in favor of SCA especially from the industry giants in Java.

In the meantime, Microsoft refrained from entering either camp. Instead, Microsoft developed Windows Communication Foundation (WCF) which is their foundation for building service-oriented applications in 2003, similar to SCA but does not describe any assemblies. It only
focuses on service and client development and while it supports multiple languages, it runs on .NET alone. It goes without saying that while SCA represents an open standard, WCF does not. It is due for release in 2007.

According to Forrester (see Gilpin et al. 2006), if in addition to IBM and BEA, the other major vendors that have endorsed SCA and SDO, come out with products supporting it, then the de facto standard for the development of SOA applications will be established. The prospects for JBI, however, are dim. It might probably only survive as a Java implementation to be plugged into SCA.

However, many industry experts firmly believe that JBI and SCA are not a threat to each other. Instead it is far more likely that they will co-exist as a part of larger implementation. SCA is the upward looking of the two, with great development potential due to its abstract and flexible nature. Also, SDO is unique in that it is the only data persistence API for SOA data services. This also drives SCA towards widespread adoption.

The Forrester report concludes that the fate of JBI and SCA/SDO is not likely to be known until the first quarter of 2007. It says that eventually, vendor support will determine the shaping of these models (see Gilpin et al. 2006).
8 Sample Implementation Based on SCA

Until now, we discussed numerous abstract models, various standards, specifications and programming models, all pertaining to SOA. Now, we turn to the presentation of a sample application that highlights these concepts and thus making advantages and disadvantages evident. For this implementation, SCA was chosen as the underlying programming model. An SCA runtime, provided by the Apache open source project Tuscany, serves as the programming framework for this implementation. Beside this, many other technologies are being utilized, which will be discussed at a relevant time.

In the first section of this chapter, we take a closer look at the Apache Tuscany project. Subsequently, a sample business process that leads to this implementation is presented. This includes a discussion on the scope and coverage of the implementation and corresponding SCA concepts. We then examine in detail, the created SCA Composites and their assembly into SCA Systems. Finally, the chapter concludes with a discussion of the results achieved.

8.1 Tuscany as SCA Runtime

The Tuscany project is an Apache Open Source project which is mainly driven by IBM and BEA Systems. It is currently under incubation within the Apache incubator. The project began in December 2005, with the release of its first code in late January 2006.

Tuscany delivers a runtime framework compatible with SCA and SDO which is independent of any specific product and technology. There are three distinct frameworks being developed in parallel at present:

1. SCA for the composition of service networks,
2. SDO as the uniform interface for data handling, and
3. DAS as the simple SDO interface to relational databases.

Tuscany’s first milestone release in C++ was published in June 2006, followed by its Java counterpart in August 2006. At this point, SCA version 0.9 and SDO version 2.01 were supported by Tuscany. The evolution of the specifications and the growing demands of the community compelled Tuscany to adapt to a more advanced version of SCA, version 0.95. This was the second milestone release in C++ at the beginning of November 2006. The Java counterpart, however, has not yet been released in its complete, fully functional form.

The current Tuscany runtime supports a large variety of bindings and implementation types. The implementation types include Java, C++, JavaScript, and PHP. The bindings include Web service, RMI, JSON Remote Procedure Call (JSON-RPC), Ruby, Spring, and Celtix. To expand its scope, Tuscany supports several extensibility mechanisms to create more implementation types, protocol bindings, and policies. Due to the rapid development trend in Tuscany, it is
likely that new technologies will be inculcated in upcoming releases. The basic network of Tuscany runs with a Tomcat application server or a standalone J2SE environment.

The following sample implementation is based on a forerunner of the second Java milestone implementation which will soon be released in its completed form. The sample implementation utilizes a selection of the offered bindings and implementation types.

8.2 Sample Business Process Description

Structures of business processes work in several hierarchical layers which each represent a different level of concepts. In this section, we are primarily concerned with the higher level description of the business process used. We do, however, require a brief insight into the implementation in order to proceed to a higher level of abstraction. This sample implementation uses several components which interoperate in various methods to service functional business processes. This is achieved by using a combination of services and user interfaces.

The sample application is based on a hypothetical business process that may be described as follows: Let us assume that an organization would like to plan its projects through a service-oriented application. And also, let us say that this mainly involves the handling of a database within the organization that holds all information of its employees and projects. The database serves as the information repository on the organization as well as a universal reference of critical organization records. For any new projects that this company may engage, a careful selection of employees suited to the particular task. This is followed by a selection of the right employees, their affiliation to the corresponding projects and informing them of the status. Such a system works as a collaboration of several reusable components and remote services; for example, to be able to compare the salaries of employees involved in the same project, a currency converter Web service may be used. Also, in order to acquire specific contact details of each employee, for example a detail such as a phone number, a remote Web service. Not only is the phone number get listed in this query, messages may also be sent through the same Web service.

The user interface performs all the above action. There are three interfaces used in this particular study. The first interface is capable of performing the entire set of tasks. The other two interfaces, however, so not possess a similar capability and instead extend limited functionalities for specific tasks.

Now we descend to a level of greater technicality as we enlist the specific SCA concepts that are dealt with in this implementation:

1. The creation of Component Implementations that provide both Remotable and Local Services. Remotable Services may be published to remote clients through various protocol bindings. On the other hand, Local Services merely represent internal application logic. In these Component Implementations, there exists the definition for configurable Service References for the completion of the code link to external configuration files. Java was chosen as the primary implementation technology.

2. The creation of Components that use and configure References of Component Implementations.

3. The creation and configuration of Services that are published as Remotable Services. This purpose is accomplished through Web service Bindings and RMI Bindings. Such Services
8.3 Development of SCA Composites

may be accessed by WSDL or Java Interfaces for Web service Bindings and RMI Bindings, respectively.

4. The creation and configuration of References that may consume both Remotable and Local Services via a Web service Binding or RMI Binding. These are channeled through WSDL or Java Interfaces.

5. The configuration of Assemblies of Implementations, Components, Services, and References into several SCA Composites, each are defined through configuration files.

6. The creation of user interfaces to the configured Composites in this particular implementation is manifested as Web applications and a Java Standalone Swing application.

7. The deployment of different Composites into one SCA System which lays the underlying communication platform to create various flexible business processes.

8.3 Development of SCA Composites

For flexibility and reusability issues the business process previously described is divided into several Composites. These may be assembled in a flexible manner and accessed through different clients. As explained in section 7.3.1.1, an SCA Composite is defined as (see Beisiegel et al. 2006a):

An SCA Composite is the basic unit of composition within an SCA System. An SCA Composite is an assembly of Components, Services, References, and the Wires that interconnect them. Composites can be used to contribute elements to an SCA System.

In other words, an SCA Composite embodies smaller resources, links them together and comprises an integral part of the SCA System. For the business process described above, we utilize five distinct SCA Composites. We now further examine each of these five:

1. The Phonebook Composite, which offers a Remotable Service to acquire contact details from a database and send messages to a selected location.

2. The EDM Project Composite, which also offers a Remotable Service. In addition to several other functions, it provides functionality for selective logic, access, and addition of projects.

3. The Phonebook Webclient Composite, which is able to consume Remotable Phonebook Composite Services and offers an interface as a Web application.

4. The EDM Project Webclient Composite, which is able to consume the EDM Project Composite Services and offers an interface as a Web application.

5. The EDM Project Swing Client, which consumes both the Phonebook and EDM Project Composite Services. It provides a J2SE Standalone Swing Application as its interface.
8.3.1 The Phonebook Composite

As stated earlier, the Phonebook Composite exposes a Remotable Service namely, the Phonebook Service. This service accesses contact information in a legacy system and sends messages to selected contacts. Figure xxx depicts the structure of this Composite. The following artifacts are contained in it:

- The **Phonebook Data Service Component** that provides phonebook data information.
- The **Send SMS Service Reference** that sends messages through a Web service.
- The **Phonebook Service Component** that aggregates the former.
- The **Phonebook Service** that exposes the Composite functionalities as a Web service.
- The **Assembly** that configures and wires all the elements of the Composite.

![Figure 31: SCA Composite for the Phonebook.](image)

### 8.3.1.1 Phonebook Data Service Component

This Component accesses the phonebook database and provides phonebook data information to the Phonebook Service Component. The database follows a simple model containing the entity “Person” along with its attributes, as seen in the **ER Diagram** in Figure 32. This relational database accessed through the DAS mediator. A separate DAS configuration file was created for that purpose (see Listing 24).

![Figure 32: Entity-Relationship Diagram of the Phonebook Data Model.](image)
As mentioned before, for each service, an implementation and a Component must exist. Implementations provide Services and have References to services they require. The `PhonebookDataServiceImpl` implementation class offers a service providing a `PhonebookDataService` interface to clients within the Phonebook Composite. The `PhonebookDataService` allows its clients to retrieve phone numbers for a given name. Listing 13 shows the `PhonebookDataService` Java interface.

```java
package org.hendrik.phonebook.services.phonebookdata;

import java.rmi.RemoteException;
import org.hendrik.phonebook.*;

/** *
 * Phonebook Data Service Interface.
 */
public interface PhonebookDataService {
    public PhonebookEntries getPhonebookEntry(SearchParameters searchParameters) throws RemoteException;
}
```

Listing 13: The Phonebook Data Service Java interface.

The `PhonebookDataServiceImpl` Java implementation, as shown in Listing 14, implements the former `PhonebookDataService` interface. It uses the `@Service` annotation to declare the Service and its interface provided by the implementation. To create the corresponding SCA Component, this implementation must necessarily be configured. In this case, this Component does not have any references and properties; it only defines its implementation class. Each SCA Component is represented by a Component element in the configuration file, refer to Listing 15.

### 8.3.1.2 Send SMS Service Reference

This Component accesses a Web service from [http://www.webservicex.net](http://www.webservicex.net) that sends messages to the given referenced addresses. This remote Web service offers the description of its functionality as a WSDL file. Within the configuration file, a Reference is defined through an interface and a binding, as shown in Listing 16.

Due to the limited support of http proxies in the current Tuscany SCA runtime, the native SCA support for References to remote Web services could not be implemented. Instead, this Reference was replaced by a separate Component that queries this Web service independently using the XFire Framework\(^1\).

### 8.3.1.3 Phonebook Service Component

This Component provides the remotable Phonebook Service. It aggregates phonebook data information and the service that sends messages. A `PhonebookServiceImpl` class has to be created. It offers a service providing a `PhonebookService` interface to clients in the Phonebook Composite. For the Phonebook Service Component, like for the Phonebook Data Service Component, an implementation and a Component configuration must be provided. This Component implementation references two other services, one providing a `PhonebookDataService` interface and the other a `SendSMSWorldService` interface. To reflect this within the implementation,

---

package org.hendrik.phonebook.services.phonebookdata;

import java.rmi.RemoteException;
import org.apache.tuscany.sdo.util.SDOUtil;
import org.hendrik.phonebook.*;
...
/**
 * Phonebook Data Service Implementation.
 */
@Service(PhonebookDataService.class)
public class PhonebookDataServiceDASImpl implements PhonebookDataService {
    /* *
     * Registers all types out of the Project Factory.
     */
    static {
        SDOUtil.registerStaticTypes(PhonebookFactory.class);
    }

    private static final String dbPath = "jdbc:derby://localhost/C:/c5077992/SCA-Impl/Projects/databases/phonebook/phonebookDB;";
    private static final String dbDriver = "org.apache.derby.jdbc.ClientDriver";

    public PhonebookEntries getPhonebookEntry(SearchParameters searchParameters)
        throws RemoteException {
        ...
    }
}

Listing 14: The Phonebook Data Service Java implementation.

<!-- Phonebook Data Service Component -->
<Component name="PhonebookDataServiceComponent">
  <implementation java class="org.hendrik.phonebook.services.phonebookdata.PhonebookDataServiceDASImpl"/>
</Component>

Listing 15: Phonebook Data Service Component Configuration.

<!-- Send SMS Service Reference -->
<Reference name="SendSMSWorldServiceReference">
  <interface.wsdl interface="http://www.webserviceX.NET#wsdl.interface(SendSMSWorldSoap)"
                wsdl:wsdlLocation="http://www.webserviceX.NET
                wsdl/SendSMSWorldService.wsdl"/>
  <binding.ws endpoint="http://www.webserviceX.NET#wsdl.endpoint(SendSMSWorld/SendSMSWorldSoap)"
             location=wsdl/SendSMSWorldService.wsdl"/>
</Reference>

Listing 16: Send SMS Service Reference Configuration.

additional @Reference annotations (see Listing 18) are used for injection from the configuration file. Since this service is exposed as a Remotable Service, a PhonebookService interface has to be created that must be annotated with @Remotable (see Listing 17). Again, the Phonebook Service implementation needs to be configured as a Component with its implementation class and all its references as part of the configuration file, as seen in Listing 19.
8.3 Development of SCA Composites

```java
package org.hendrik.phonebook;

import org.osoa.sca.annotations.Remotable;
import org.apache.tuscany.api.annotation.DataType;
import java.rmi.RemoteException;
import org.hendrik.phonebook.*;

/**
 * PhonebookService Java interface.
 */
@Remotable
@DataType(name="commonj.sdo.DataObject")
public interface PhonebookService {
    public String sendSMS(PhonebookEntry param0, String param2) throws RemoteException;
    public PhonebookEntries getPhonebookEntry(SearchParameters param4) throws RemoteException;
}
```

Listing 17: The Phonebook Service Java interface.

```java
package org.hendrik.phonebook.services.phonebook;

import org.osoa.sca.annotations.Reference;
import org.osoa.sca.annotations.Service;
import org.hendrik.phonebook.*;

/**
 * Phonebook Service Implementation.
 */
@Service(PhonebookService.class)
public class PhonebookServiceImpl implements PhonebookService {
    private PhonebookDataService phonebookDataService;
    private SendSMSWorldSoap sendSMSWorldService;

    @Reference
    public void setPhonebookDataService(PhonebookDataService phonebookDataService) {
        this.phonebookDataService = phonebookDataService;
    }

    @Reference
    public void setSendSMSWorldService(SendSMSWorldSoap sendSMSWorldService) {
        this.sendSMSWorldService = sendSMSWorldService;
    }

    public PhonebookServiceImpl() {
    }

    public PhonebookEntries getPhonebookEntry(SearchParameters searchParameters) throws RemoteException {
        ...
    }

    public String sendSMS(String senderEmail, PhonebookEntry pbEntry, String content) throws RemoteException {
        ...
    }
}
```

Listing 18: The Phonebook Service Java implementation.
8.3.1.4 Phonebook Service

This artifact publishes the Phonebook Service Component with all its functionalities over a Web service binding for access by other Composites and remote Web service clients. The interface thus provided exists as a WSDL file, as given in Listing 26. This WSDL file may then be used by the client to generate the corresponding Java interfaces and implementations using the SDO API to finally access this service. Within the configuration file (see Listing 20), this Service is attributed with an interface and a binding tag specifying its access mechanism. The complete configuration file is given in Listing 28:

Listing 20: Phonebook Service Configuration.

8.3.2 The EDM Project Composite

As mentioned before, the EDM Project Composite exposes a Remotable Service namely, the Project Service. This service accesses project, employee, and other related information from a data source and provides logic to retrieve and add projects. Furthermore, a Web service for currency conversion is used to map the employees’ salaries to a common currency. This Composite also includes a connection to the Phonebook Composite described above, to obtain contact information of the organizations’ employees and to inform them of project affiliations through messages. Figure 33 depicts the structure of this Composite. The following artifacts are contained in it:

- The **EDM Data Service Component** that provides project and employee data information.
- The **Currency Converter Service Reference** that delivers up-to-date conversion rates through a Web service.
- The **Phonebook Service Reference** that provides contact information and sends messages.
8.3 Development of SCA Composites

- The *Project Service Component* that aggregates the three aforementioned artifacts.
- The *Project RMI Service* that exposes the Composite functionalities as a RMI service.
- The *Project Web Service* that exposes the Composite functionalities as a Web service.
- The *Assembly* that configures and wires all the elements of the Composite.

![SCA Composite for the EDM Project](image)

**Figure 33:** SCA Composite for the EDM Project.

### 8.3.2.1 EDM Data Service Component

This Component accesses a Derby database that provides the service requester with information about projects and employees of the organization. Each employee is associated with a department and may be part of one or several usergroups. Furthermore, each employee has certain skills which in turn are necessary to be part of a certain project. Specific queries allow the selection of employees that match the required skills of a given project. This datamodel, provided by the SAP Education Data Model (EDM) outlined as an ER diagram is shown in Figure 34.

The EDM Data Service Component provides its data to the Project Service Component. To access this relational database, DAS is used as a mediator between the data source and the client. The DAS configuration file is provided in Listing 25.

### 8.3.2.2 Currency Converter and Phonebook Service Reference

These Components access Web services through a WSDL interface. While the Currency Converter Service Reference passes its requests on to a Web service that is universally accessible on the Internet, the Phonebook Service Reference requests data from a Remote Service that is exposed in the local system. Essentially, the Phonebook Service Reference consumes the Service provided by the Phonebook Service Composite as described in section 8.3.1.4. Both Web services provide a WSDL file that describes the functionalities that may be accessed by these References. To communicate with the Currency Converter Web service and due to the missing support of http proxies, this Reference is implemented as a separate Component using the XFire Framework.
8.3.2.3 Project Service Component

This Component provides a remotable Project Service. It aggregates the two References and uses the Service provided by the EDM Data Service Component to offer advanced functionality. For this purpose, a `EdmProjectServiceImpl` Java implementation must be given. Its interface `EdmProjectService` is offered to clients within the EDM Project Composite. The Project Service Component is configured in a way similar to the previously described Phonebook Service Component (see Listing 29).

8.3.2.4 Project Web and RMI Service

This Composite offers two remote Services to clients outside this Composite. First, the Project Service Component functionalities are published over a Web service binding and are offered through a WSDL file as its description (see Listing 27). This WSDL may again be used by other clients to generate corresponding Java interfaces and implementations to query this Service. This Service is represented in the configuration file as indicated in Listing 29.

The Project Service Component may also be accessed through a second Service, the Project RMI Service. This Service offers an RMI binding and a Java interface. Hence, other remote RMI clients are able to consume this Service through its RMI binding. Within the SCA configuration file, this Service is represented as outlined in Listing 21. The complete configuration file of the whole Composite is given in Listing 29.
8.3 Development of SCA Composites

8.3.3 The Phonebook Webclient Composite

As mentioned before, the Phonebook Webclient Composite represents a client that consumes the Phonebook Service and provides a Web application to the user. Through this interface, the user may access all the functionalities described in the WSDL file of the Phonebook Service. As also seen in Figure 35, this simple Composite contains the following artifacts:

- The Phonebook Service Reference that provides contact information and sends messages.
- The Phonebook Service Component that accesses the Phonebook Service Reference.
- The User Interface as a Web application that provides access to the Phonebook Service Component.
- The Assembly that configures and wires all the elements of the Composite.

Figure 35: SCA Composite for the Phonebook Webclient.

The Phonebook Service Reference and the Phonebook Service Component follow the same concepts as the Composites described previously. For that reason, this will not be repeated here. The configuration file describing the Assembly of all of the elements within the Composite is given in Listing 30.

The Web application utilizes a set of different technologies, which will be briefly outlined subsequently. This user interface is essentially implemented utilizing the basic technologies of HyperText Markup Language (HTML) and JavaScript. For enhancements, this interface uses the Dojo Toolkit\(^2\) which offers several open source JavaScript widgets for easier and faster Web development. It also provides the fundamentals for Asynchronous JavaScript and XML (AJAX) as another technique for creating Web applications. It is AJAX’s intent to make Web

---

8 Sample Implementation Based on SCA

pages more responsive through the exchange of small data chunks between the server and the client behind the scene. This results in the integration of partial Web page updates instead of reloading the whole page each time a request is sent and a response is received. Thus, the interactivity, usability, and the speed may be increased significantly. To obtain data from the Phonebook Service Component, a mediator has to be used. This mediator is represented as a Java Servlet. It exchanges information with the HTML page through JSON-RPC-formatted data and communicates directly with the Phonebook Service Component.

8.3.4 The EDM Project Webclient Composite

As mentioned before, the EDM Project Webclient Composite represents a client that consumes the Project Service and that provides a Web application to the user. In this application, the Project Service is accessed through a WSDL interface (see Listing 27). In this, the user only has the right to list all the projects and their details instead of all the functionalities that are offered by that Web service. Because of the close resemblance, we do no discuss the detailed descriptions of this Composite again. As seen in Figure 36, this simple Composite contains the following artifacts, which are configured as shown in Listing 31:

- The Project Service Reference that provides contact information and sends messages.
- The Project Service Component that accesses the Project Service Reference through a Web service binding.
- The User Interface as a Web application that provides access to the Project Service Component.
- The Assembly that configures and wires all the elements of the Composite.

![Figure 36: SCA Composite for the EDM Project Webclient.](image)

8.3.5 The EDM Swing Client Composite

The EDM Project Swing Client Composite represents a much more powerful interface to the Project Service as compared to the Webclient Composite. It makes full use of the functionalities provided by the Project Service, that are exposed in its WSDL file (see Listing 27). This Composite, as shown in Figure 36, and configured according to Listing 31 contains the following artifacts:
8.4 Deployment of Composites into an SCA System

After examining each of the five implemented Composites in general, their structure, and their technological underpinnings, we now turn to their deployment. The deployment of these Composites into an SCA System makes their functionalities available for use. These Composites are configured such that they form a reliable interdependency which utilizes the services provided by each. Hence, the deployment of these into the same SCA System forms the basis for complete flexibility in communication amongst them. Depending upon the interconnectivities, several combinations may be possible. Each of these leads to the formation of a separate application and function. Figure 38 gives an overview of the entire system. The various interconnections, described ahead in detail, which are relevant to the sample implementation, are listed as:

1. The connection between the Phonebook Webclient Composite and the Phonebook Composite.
2. The connection between the EDM Project Webclient Composite and the EDM Project Composite.
3. The connection between the EDM Project Swing Client Composite, the EDM Project Composite, and the Phonebook Composite.

8.4.1 Phonebook Service Web Application

The interconnection between the Phonebook Webclient Composite and the Phonebook Composite provides the user interface access to the Phonebook Service. More specifically, the Phonebook Service may be queried for phone numbers by entering the corresponding fields for first and last

![Diagram of SCA Composite for EDM Project Standalone Swing client.](image_url)
name. A table where different data fields can be selected and sorted will display the search results of the query. Upon selection of one or multiple entries, the Phonebook Service is consumed again. After some preprocessing, the Service uses the Send SMS Web Service to actually send the message to the corresponding phone number as an SMS. A status message after spanning all these services is returned to the user interface. The status messages are displayed in a console. This console is also used for any error messages that might appear due to invalid inputs, incorrectly formatted data, or the inability to connect to the remote services. Like all the data exchanged between these services, the request and response thus invoked are SOAP messages. The request from the Phonebook Service to the Send SMS Web Service and the response in the other direction are given as an example in Listing 22 and 23 respectively. The screenshot of this Web application, as shown in Figure 39, outlines a process of searching phonebook entries with the given name, sending messages, and catching several input errors within the process.

8.4.2 Project Service Web Application

The second separate application is based on a composition of the EDM Project Webclient Composite and the EDM Project Composite. This application also utilizes a web interface based on HTML and JavaScript. The EDM Project Composite provides the simple feature of accessing the EDM Project Service by means of a Web service binding, and returns all the projects from its database in response. These results are again displayed in the form of a table that may be sorted by the user. Upon the completion of the process, the console displays error and success messages. A screenshot of this application while active, is shown in Figure 40.

8.4.3 EDM Project Standalone Swing Application

The third scenario represents the most thorough example. It relies on three Composites deployed in the SCA System. The EDM Project Swing Client represents the user interface which is linked
8.4 Deployment of Composites into an SCA System

Figure 39: Web Interface of SCA Phonebook System Use Case.

Figure 40: Web Interface of SCA EDM Project Use Case.
8 Sample Implementation Based on SCA

Listing 22: SOAP Request from the Phonebook Composite to the remote Web service.

to the EDM Project Composite for comprehensive functionalities for the handling of existing and new projects. This service, in turn, consumes the Phonebook Service provided by the Phonebook Composite. As mentioned before, the EDM Project Service offers two kinds of bindings. The Web service binding was utilized by the previous scenario whereas this particular scenario uses the RMI binding. An RMI Server is set up by the EDM Project Service, which the EDM Project Swing Client can then establish a connection with. This combination of the three mentioned Composites, allows the EDM Project Composite to simultaneously add a new project and automatically retrieve Phone numbers of the selected project members. Then it informs them about the project via a short message. The front-end of these functionalities, such as display and addition of projects, are depicted in the screenshot in Figure 41.

8.5 Discussion and Evaluation

We have seen how application logic may be further organized into separate application components. These components may be developed independently and then made available for composition. This may be realized in SCA, by means of Composites which constitute Services and References. These may rely on different binding technologies, due to which, flexible composition is possible. In the sample implementation, the EDM Project Composite provides a Service that is accessible both through a Web service and an RMI binding. Two separate clients make use of these bindings to access the EDM Project Service. This implies that the composition is independent of the implementation technology used for the Composite. Hence, one Composite
8.5 Discussion and Evaluation

Listing 23: SOAP Response from remote Web service to the Phonebook Composite.

Figure 41: J2SE Standalone Swing Interface of SCA EDM Project Use Case.

may easily be replaced by another Composite increasing the overall flexibility and agility. This unique feature is offered by the SCA programming model, thus addressing a chief factor in SOA realization.

The Phonebook Service, offered by the Phonebook Composite, may be used within two separate compositions. This modularity lends it the advantage of being reusable. In the first case, the
Phonebook Service is used as a directory to search for individuals and their phone numbers to which messages may be sent. In the second case, the service of sending messages is invoked automatically when a new project is added by the user in the EDM Project Swing Client. This demonstrates several crucial concepts in SOA, such as reusability and loose coupling, which are furnished by SCA.

This implementation only represents a small system of Composites. The vision that the author had in mind, however, extends over a larger scale. The implementation demonstrates the assembly of several composite to achieve a relatively simple task. In global business operations, the same concept can be extended to include a wide range of Composites each fulfilling separate functions in combination with each other. These Composites may either be developed specifically for the desired application or purchased from a Composite developer. Another advantage conferred would be that these Composites would only have to be configured and assembled together into the desired business application. This modularity would revolutionize the entire system, by separating the technical concerns of the Composite developer from those of the system assembler. The rewards for this are enormous. The entire system is simplified significantly, the time to market is greatly reduced, and the developer productivity is increased.

The implementation has revealed several first hand advantages of SCA to the author. SCA appears to be a beginning in the quest for a complete programming model in the realization of SOA. It is independent of any implementation technology and middleware. It supports numerous binding types for communication over various protocols. SCA, in general, proposes an extensibility mechanism for interface, binding, and implementation types. This feature, however, has not yet been included in Tuscany. SCA, and especially its Tuscany implementation, is based on existing, established standards. This takes into account the current knowledge base of the developer, easing the technical skill required for its use. The model also allows a separation of non-functional concerns from the core logic. This is significant because a vast amount of time and resources are spent in accurately developing the business logic; such a model thus largely simplifies these issues. Although SCA suggests the use of QoS, Tuscany does not support these yet. The fact that SCA utilizes a model based approach makes it an organized and somewhat simplified source. Overall, SCA is an organized, efficient and highly flexibly method of SOA realization. In this implementation, it is the author’s opinion that while Tuscany embodies several of SCA’s ideals, it still does not completely conform to all of its features.

In the course of this implementation using Tuscany, several problems have been encountered by the author. Due to Tuscany’s incubation status, the framework contains several “bugs”, missing documentation, and dependency issues. Since, all parts of Tuscany are currently under development, changes made to one may drastically affects the working of the entire framework. To further add to a substantial list of problems is that Tuscany relies on several evolving technologies which again in turn affect the Tuscany framework. It implements basic DAS that has an ambiguous mapping which is extremely restrictive. To summarize, Tuscany has a lot to prove and achieve in the quest for a complete realization of what SCA truly represents.

Even SCA, however, bears some minor disadvantages. The definition of Services, Composites and their behavior is vague and due to the several meanings that may arise from it, it poses difficulties. Also, languages cannot be mixed within on Composite, for instance Java code will be unable to call C++ code directly. Another problem lies in the failure to reuse of tightly coupled artifacts within a composite. Also, there exists no concept of events in SCA.

In conclusion, SCA is an imaginative, adaptable and resilient model that simplifies the approach of realizing SOA. Although, there are some disadvantages to it, most of these are negligible in front of the vast possibilities it presents. Even industry leaders acknowledge its immense
potential as the future of SOA implementation. Tuscany is a framework to inculcate SCA itself, and despite its many failings, it must be noted that it lies at a cutting edge research frontier, which justifies some of these flaws. In time however, these technologies will evolve, improve and establish themselves which will lead to an enormous business integration transformation.
9 Summary

In this thesis we have visited all aspects of Service-Oriented Architecture. We have discussed the various concepts upon which it has evolved as well as the concepts that were created as the realization of SOA took place. We may now conclude that SOA integrates two distinct fields. These are IT and Business. Their interdependency represents the essence of e-commerce. There have been several direct repercussions of this interdependency, for example, business is no longer restricted by distance. Today, IT exists as an integral part of every business. It empowers mankind with speed, vast resources and great connectivity. The challenge now, is to further advance our current stage of development.

The author selected SOA as a subject for this thesis because it promises to be the future of the IT-Business interrelation. It is predicted that global players will need more automation, more integration, and connectivity to surpass the current rate of advance. SOA is the answer to this advancement. Now we take one final look and summarize all the information presented in this thesis.

The in depth analysis into SOA given in this thesis is divided into three main parts viz. “Fundamentals”, “Key Technologies Realizing SOA” and “SOA Implementations”. In the first part we discussed the key concepts involved in SOA and the specific business needs that led to the necessity for such a technology. In essence, after the onset of Internet, when businesses began to grow to conform to the IT revolution, there was a disparity between the abstract business ideas and the highly technical IT concepts. For supply chains and globally connected businesses, the need for a convenient combination of the business and IT skills as well as a fair amount of automation of the entire process was required and thus emerged the concept of SOA. It grew as an abstract idea with a multi-layered structure. These layers separated abstract functions from low level ones, thus enabling different roles to be fulfilled within one structure. This layered structure is referred to as the SOA Stack. For SOA to become a universally accepted architecture, it was necessary to define a common model for the benefit of reference to all. This model that defines SOA comprehensively is known as the SOA Reference Model.

The second part describes the key technologies realizing SOA. For any two people to effectively communicate, there must exist a common language of idea exchange between them. Such a common language did not exist between the business analyst and the software engineer when the Business-IT interdependency was first forged. The need for common notations thus arose. Notations exist in many different types and forms, often addressing different purposes. The particular notation that benefited the use of SOA was BPMN. It became the common ground between the abstract strategies of business and the technical coding of IT. It was thus most suitable for SOA.

Web services are amongst the most common and popular approach to SOA realization. These are wrapped functionalities that are universally defined and accessible. With Web services, there emerged the need to standardize their nature in order to make wide spectrum communication
amongst them possible. With the development of numerous standards and different organiza-
tions building them further in their own ways, some standards had to be established as universal. These are WSDL that describes the Web service, UDDI that registers it and SOAP, that deals with message exchange between Web services. There is no dispute or controversy regarding these standards and their establishment and usage is considered final. There are several new standards currently under development. It is yet to be seen which newer standards will emerge as mainstays. This problem of conflicting standards arises because different organizations define standards fulfilling similar purposes, which belies the very reason for having a standard. Another notable concept is service orchestration, which is a term used to describe the composition of Web services. BPEL is evolving as the frontrunner for orchestration standardization. It describes the process flow of complex business processes. While orchestration refers to an imperatively described and executable process, choreography is a declarative description to track message sequences between multiple collaborating sources. WS-CDL is the most promising specification addressing choreography.

There has been significant progress in all these aspects of Web services. The only inadequately addressed issue is QoS. Security is amongst the greatest concerns in the use of Web services. Unless security threats are laid to rest, businesses will be unwilling to commit highly sensitive information in the form of Web services. The provision of security in Web services is crucial to their longevity and success as a technology. Another key aspect that needs great improvement is the management of Web services. While there are several standards to cater to this, none are flawless.

The final part of this thesis deals with the various approaches to SOA implementation. Ultimately, an abstract concept like SOA must gradually sink to a lower level and develop a more concrete shape. This concreteness is represented in terms of a programming model. First we discuss the highly abstract concept of ESB. ESB is a theoretical, non-standard concept to realize a SOA. Various organizations have adopted ESB and transformed it to their own interpretation. As a result, the true essence of the original concept has been lost. It was then realized that ESB had lost its appeal with the loss of its standardized nature.

Thus more concrete models replaced it. These were Java EE 5 and SCA. Java EE provides a Java-dependent, complex framework to implement SOA solutions. It is however, extremely skill intensive. Its complexity makes it exclusive and expensive to use. JBI is a standardized platform that entails a service-oriented approach for the integration and structuring of enterprise functions. It primarily provides a basis for the construction of an ESB. JBI has the support of one of the largest Java industry leaders, Sun Microsystems, although IBM and BEA Systems do not concur. In the end of 2005, they, on the other hand, have conferred their support to a parallel approach to the implementation of SOA known as SCA. SCA is a set of specifications that particularly aims at building applications and systems based on SOA. It is platform and programming language independent. It heavily relies on preexisting standards and therein lies one of its greatest advantages. It divides the development of an application into three parts viz. service constructions, based on different implementation technologies, service assembly to build complete applications through wiring of services and deployment of the constructed services and the assembly descriptions. It works in combination with SDO, which lends the system the capacity for flexible data handling. The handling and access of data sources then becomes universal. SCA also provides features like reusability of defined components. One of the greatest advantages of SCA is that it separates the concerns of the programmer from the assembler making it possible to concentrate on the business logic. This also increases programmer productivity and simplifies development. In SCA, the composition of loosely-coupled components
is possible. Overall, SCA is, in the author’s opinion, the most exceptional method for SOA implementation.

For this reason, the implementation supporting the thesis utilizes SCA. Through this implementation, the author has attempted to demonstrate some outstanding features in SCA. The implementation uses wrapped components that are connected such that reusability is possible. Also, the interacting components are independent of language as well as binding technologies.

In conclusion, SOA will become an important prerequisite for business integration over the next few years. The conflicting standards evolving around SOA are likely to settle. Their establishment will determine a new step in the evolution of this technology. For now, the author maintains that SOA will provide the underlying foundation for any future evolution in business integration.
Appendix
A DAS Configuration Files

Listing 24: DAS configuration for Phonebook database.

```
<?xml version="1.0" encoding="UTF-8"?>
<Config
    xsi:NamespaceSchemaLocation="http:///org.apache.tuscany.das.rdb/config.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    dataObjectModel="http://www.hendrik.org/phonebook">
    <Table tableName="PB_PERSON" typeName="PhonebookEntry">
        <Column columnName="ID" propertyName="id" primaryKey="true"
generated="true"/>
        <Column columnName="SALUTATION" propertyName="salutation"/>
        <Column columnName="FIRST_NAME" propertyName="firstName"/>
        <Column columnName="LAST_NAME" propertyName="lastName"/>
        <Column columnName="COUNTRY_CODE" propertyName="countryCode"/>
        <Column columnName="PHONE_NUMBER" propertyName="phoneNumber"/>
    </Table>
</Config>
```

Listing 24: DAS configuration for Phonebook database.

```
<?xml version="1.0" encoding="UTF-8"?>
<Config
    xsi:noNamespaceSchemaLocation="http:///org.apache.tuscany.das.rdb/config.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    dataObjectModel="http://www.hendrik.org/edm/project">
    <Table tableName="RI_EDM_EMPLOYEE" typeName="Employee">
        <Column columnName="EMPLOYEE_ID" propertyName="id" primaryKey="true"
generated="true"/>
        <Column columnName="DEPARTMENT_ID" propertyName="departmentId"/>
        <Column columnName="SALUTATION" propertyName="salutation"/>
        <Column columnName="FIRST_NAME" propertyName="firstName"/>
        <Column columnName="LAST_NAME" propertyName="lastName"/>
        <Column columnName="EMAIL" propertyName="email"/>
        <Column columnName="SALARY" propertyName="salary"/>
        <Column columnName="CURRENCY" propertyName="currency"/>
    </Table>
    <Table tableName="RI_EDM_PROJECT" typeName="Project">
        <Column columnName="PROJECT_ID" propertyName="id" primaryKey="true"
generated="true"/>
        <Column columnName="LEAD_ID" propertyName="leaderId"/>
        <Column columnName="TITLE" propertyName="title"/>
        <Column columnName="DESCRIPTION" propertyName="description"/>
        <Column columnName="START_DATE" propertyName="startDate"_converterClassName="org.hendrik.edm.project.services.EdmDataServiceDASImpl$DateConverter"/>
        <Column columnName="END_DATE" propertyName="endDate"_converterClassName="org.hendrik.edm.project.services.EdmDataServiceDASImpl$DateConverter"/>
        <Column columnName="STATUS" propertyName="status"/>
    </Table>
    <Table tableName="RI_EDM_SKILL" typeName="Skill"/>
```
Listing 25: DAS configuration for EDM database.
B WSDL Description Files

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/
xmlns:tns="http://www.hendrik.org/phonebook"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.hendrik.org/phonebook"
xmlns="http://schemas.xmlsoap.org/wsdl/"
name="PhonebookService">

<wsdl:types>
  <xsd:schema targetNamespace="http://www.hendrik.org/phonebook"
xmns:tns="http://www.hendrik.org/phonebook"
xmns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:complexType name="SearchParameters">
      <xsd:sequence>
        <xsd:element name="firstName" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
        <xsd:element name="lastName" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="PhonebookEntries">
      <xsd:sequence>
        <xsd:element name="phonebookEntry" type="tns:PhonebookEntry"
minOccurs="0" maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="PhonebookEntry">
      <xsd:sequence>
        <xsd:element name="firstName" type="xsd:string" />
        <xsd:element name="lastName" type="xsd:string" />
        <xsd:element name="countryCode" type="xsd:string" />
        <xsd:element name="phoneNumber" type="xsd:string" />
      </xsd:sequence>
    </xsd:complexType>

    <xsd:element name="getPhonebookEntry">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="searchParameters" type="tns:SearchParameters" />
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

    <xsd:element name="getPhonebookEntryResponse">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="PhonebookEntries" type="tns:PhonebookEntries" />
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
</wsdl:types>
</wsdl:definitions>
```
<xsd:schema>
  <xsd:element name="sendSMS">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="senderEmail" type="xsd:string" />
        <xsd:element name="phonebookEntry" type="tns:PhonebookEntry" />
        <xsd:element name="content" type="xsd:string" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="sendSMSResponse">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="sendSMSResponse" type="xsd:string" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

<wsdl:message name="getPhonebookEntryRequest">
  <wsdl:part element="tns:getPhonebookEntryRequest" name="getPhonebookEntryRequest" />
</wsdl:message>

<wsdl:message name="getPhonebookEntryResponse">
  <wsdl:part element="tns:getPhonebookEntryResponseRequest" name="getPhonebookEntryResponseRequest" />
</wsdl:message>

<wsdl:message name="sendSMSRequest">
  <wsdl:part name="sendSMSRequest" element="tns:sendSMS" />
</wsdl:message>

<wsdl:message name="sendSMSResponse">
  <wsdl:part name="sendSMSResponse" element="tns:sendSMSResponse" />
</wsdl:message>

<wsdl:portType name="PhonebookService">
  <wsdl:operation name="getPhonebookEntry">
    <wsdl:input message="tns:getPhonebookEntryRequest" />
    <wsdl:output message="tns:getPhonebookEntryResponse" />
  </wsdl:operation>
  <wsdl:operation name="sendSMS">
    <wsdl:input message="tns:sendSMSRequest" />
    <wsdl:output message="tns:sendSMSResponse" />
  </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="PhonebookServiceSOAP" type="tns:PhonebookService">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="getPhonebookEntry">
    <soap:operation soapAction="http://www.hendrik.org/phonebook/getPhonebookEntry" />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
Listing 26: WSDL file describing the Phonebook Web service.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wSDL:definitions xmlns:soap="http://schemas.xmlsoap.org/wsd1/soap/"
                  xmlns:tns="http://www.hendrik.org/edm/project"
                  xmlns:wSDL="http://schemas.xmlsoap.org/wsd1/
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  targetNamespace="http://www.hendrik.org/edm/project"
                  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <wSDL:definitions>
    <xsd:schema targetNamespace="http://www.hendrik.org/edm/project">
      <xsd:complexType name="Projects">
        <xsd:sequence>
          <xsd:element name="projects" type="tns:Project"
                        minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>
    <xsd:complexType name="Project">
      <xsd:sequence>
        <xsd:element name="id" type="xsd:int"/>
        <xsd:element name="title" type="xsd:string"/>
        <xsd:element name="description" type="xsd:string"/>
        <xsd:element name="startDate" type="xsd:date"/>
        <xsd:element name="endDate" type="xsd:date"/>
        <xsd:element name="leaderId" type="xsd:int"/>
        <xsd:element name="memberIds" type="xsd:int"
                    minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="requiredSkillIds" type="xsd:int"
                    minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="Employees">
      <xsd:sequence>
        <xsd:element name="employees" type="tns:Employee"
                    minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </wSDL:definitions>
</wSDL:definitions>
```
<xsd:element name="Employee">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="id" type="xsd:int"/>
      <xsd:element name="salutation" type="xsd:string"/>
      <xsd:element name="firstName" type="xsd:string"/>
      <xsd:element name="lastName" type="xsd:string"/>
      <xsd:element name="email" type="xsd:string"/>
      <xsd:element name="currency" type="xsd:string"/>
      <xsd:element name="salary" type="xsd:decimal"/>
      <xsd:element name="departmentId" type="xsd:string"/>
      <xsd:element name="skillIds" type="xsd:int" maxOccurs="unbounded" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="EmployeeRequirements">
  <xsd:sequence>
    <xsd:element name="skills" type="tns:Skill" maxOccurs="unbounded" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Skills">
  <xsd:sequence>
    <xsd:element name="skills" type="tns:Skill" maxOccurs="unbounded" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Skill">
  <xsd:sequence>
    <xsd:element name="id" type="xsd:int"/>
    <xsd:element name="name" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Departments">
  <xsd:sequence>
    <xsd:element name="departments" type="tns:Department" maxOccurs="unbounded" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Department">
  <xsd:sequence>
    <xsd:element name="id" type="xsd:string"/>
    <xsd:element name="name" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:element name="getMatchingEmployees"/>
type="tns:EmployeeRequirements">
</xsd:element>

<xsd:element name="getMatchingEmployeesResponse" type="tns:Employees">
</xsd:element>

<xsd:element name="getAllProjects" type="xsd:int">
</xsd:element>

<xsd:element name="getAllProjectsResponse" type="tns:Projects">
</xsd:element>

<xsd:element name="deleteProject" type="xsd:int">
</xsd:element>

<xsd:element name="deleteProjectResponse" type="xsd:string">
</xsd:element>

<xsd:element name="addProject" type="tns:Project">
</xsd:element>

<xsd:element name="addProjectResponse" type="xsd:string">
</xsd:element>

<xsd:element name="getAllSkills" type="xsd:int">
</xsd:element>

<xsd:element name="getAllSkillsResponse" type="tns:Skills">
</xsd:element>

<xsd:element name="getAllDepartments" type="xsd:int">
</xsd:element>

<xsd:element name="getAllDepartmentsResponse" type="tns:Departments">
</xsd:element>

<xsd:element name="getMaterial" type="xsd:string">
</xsd:element>

<xsd:element name="getMaterialResponse" type="xsd:string">
</xsd:element>

<xsd:element name="getEmployee" type="xsd:int">
</xsd:element>

<xsd:element name="getEmployeeResponse" type="tns:Employee">
</xsd:element>

<xsd:element name="getSkill" type="xsd:int">
</xsd:element>

<xsd:element name="getSkillResponse" type="tns:Skill">
</xsd:element>

<xsd:element name="getDepartment" type="xsd:string">
</xsd:element>

<xsd:element name="getDepartmentResponse" type="tns:Department">
</xsd:element>

</xsd:schema>
</xsd1:types>
<xsd:message name="getAllProjectsResponse">

<wsdl:message name="getMaterialResponse">
  <wsdl:part name="getMaterialResponse" element="tns:getMaterialResponse"/>
</wsdl:message>

<wsdl:message name="getEmployeeRequest">
  <wsdl:part name="getEmployee" element="tns:getEmployee"/>
</wsdl:message>

<wsdl:message name="getEmployeeResponse">
  <wsdl:part name="getEmployeeResponse" element="tns:getEmployeeResponse"/>
</wsdl:message>

<wsdl:message name="getSkillRequest">
  <wsdl:part name="getSkill" element="tns:getSkill"/>
</wsdl:message>

<wsdl:message name="getSkillResponse">
  <wsdl:part name="getSkillResponse" element="tns:getSkillResponse"/>
</wsdl:message>

<wsdl:message name="getDepartmentRequest">
  <wsdl:part name="getDepartment" element="tns:getDepartment"/>
</wsdl:message>

<wsdl:message name="getDepartmentResponse">
  <wsdl:part name="getDepartmentResponse" element="tns:getDepartmentResponse"/>
</wsdl:message>

<wsdl:portType name="EdmProjectService">
  <wsdl:operation name="getAllProjects">
    <wsdl:input message="tns:getAllProjectsRequest"/>
    <wsdl:output message="tns:getAllProjectsResponse"/>
  </wsdl:operation>
  <wsdl:operation name="getMatchingEmployees">
    <wsdl:input message="tns:getMatchingEmployeesRequest"/>
    <wsdl:output message="tns:getMatchingEmployeesResponse"/>
  </wsdl:operation>
  <wsdl:operation name="deleteProject">
    <wsdl:input message="tns:deleteProjectRequest"/>
    <wsdl:output message="tns:deleteProjectResponse"/>
  </wsdl:operation>
  <wsdl:operation name="addProject">
    <wsdl:input message="tns:addProjectRequest"/>
    <wsdl:output message="tns:addProjectResponse"/>
  </wsdl:operation>
  <wsdl:operation name="getAllSkills">
    <wsdl:input message="tns:getAllSkillsRequest"/>
    <wsdl:output message="tns:getAllSkillsResponse"/>
  </wsdl:operation>
  <wsdl:operation name="getAllDepartments">
    <wsdl:input message="tns:getAllDepartmentsRequest"/>
    <wsdl:output message="tns:getAllDepartmentsResponse"/>
  </wsdl:operation>
  <wsdl:operation name="getMaterial">
    <wsdl:input message="tns:getMaterialRequest"/>
    <wsdl:output message="tns:getMaterialResponse"/>
  </wsdl:operation>
  <wsdl:operation name="getEmployee">
    <wsdl:input message="tns:getEmployeeRequest"/>
  </wsdl:operation>
</wsdl:portType>
Listing 27: WSDL file describing the EDM Project Web service.
Listing 28: Phonebook Composite configuration file.
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
    xmlns:wsdl="http://www.w3.org/2006/01/wsdl-instance"
    xmlns:dbsdo="http://incubator.apache.org/tuscany/xmlns/
databinding/sdo/1.0-incubator-M2"
    xmlns:rmi="http://incubator.apache.org/tuscany/xmlns/
binding/rmi/1.0-incubator-M2"
name="org.hendrik.edm.project">
<!-- Import Phonebook Factory -->
<dbsdo:import eds factory="org.hendrik.phonebook.PhonebookFactory"/>

<!-- Import Currency Converter WSDL -->
<dbsdo:import eds location="wsdl/CurrencyConvertorService.wsdl"/>

<!-- Project RMI Service -->
<service name="EdmProjectService">
    <interface.java interface="org.hendrik.edm.project.EdmProjectService"/>
    <rmi:binding rmi host="localhost" port="1099"serviceName="EdmProjectRmiService"/>
    <reference name="edmProjectService">EdmProjectServiceComponent</reference>
</service>

<!-- Project Web Service -->
<service name="EdmProjectWebService">
    <interface.wsdl interface="http://www.hendrik.org/edm/project
        #wsdl.interface(EdmProjectService)"
wsdl:wsdlLocation="http://www.hendrik.org/edm/project
wsdl/EdmProjectService.wsdl"/>
    <binding.ws endpoint="http://www.hendrik.org/edm/project#wsdl.
        endpoint(EdmProjectService/EdmProjectServiceSOAP)"
conformanceURIs="http://ws-i.org/profiles/basic/1.1"
location="wsdl/EdmProjectService.wsdl"/>
    <reference name="edmProjectService">EdmProjectServiceComponent</reference>
</service>

<!-- Project Service Component -->
<component name="EdmProjectServiceComponent">
    <implementation.java class="org.hendrik.edm.project.services.project.
        EdmProjectServiceImpl"/>
    <reference name="edmDataService">EdmDataServiceComponent</reference>
    <reference name="currencyConvertorService">CurrencyConvertorServiceReference</reference>
    <reference name="phonebookService">PhonebookServiceReference</reference>
</component>

<!-- EDM Data Service Component -->
<component name="EdmDataServiceComponent">
    <implementation.java class="org.hendrik.edm.project.services.edmdata.
        EdmDataServiceDASImpl"/>
</component>

<!-- Phonebook Service Reference -->
<reference name="PhonebookServiceReference">
    <interface.wsdl interface="http://www.hendrik.org/phonebook
        #wsdl.interface(PhonebookService)"
wsdl:wsdlLocation="http://www.hendrik.org/phonebook
wsdl/PhonebookService.wsdl"/>
    <binding.ws endpoint="http://www.hendrik.org/phonebook#wsdl.
        endpoint(PhonebookService/PhonebookServiceSOAP)"
location="wsdl/PhonebookService.wsdl"/>
</reference>

<!-- Currency Convertor Service Reference -->
<reference name="CurrencyConvertorServiceReference">
    <interface.wsdl interface="http://www.webserviceX.NET#wsdl.
        interface(CurrencyConvertorSoap)"
wsdl:wsdlLocation="http://www.webserviceX.NET wsdl/CurrencyConvertorService.wsdl" />
</binding.ws>
</reference>
</composite>

Listing 29: EDM Project Composite configuration file.

```xml
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
   xmlns:wsdli="http://www.w3.org/2006/01/wsdl-instance"
   name="hendrik-phonebook-webclient">
   <!-- Import Phonebook Factory -->
      factory="org.hendrik.phonebook.PhonebookFactory"/>
   <!-- Phonebook Service Implementation at Client Side -->
   <component name="PhonebookServiceComponent">
      <implementation.java class="org.hendrik.phonebook.webclient.services.phonebook.PhonebookServiceImpl"/>
      <reference name="phonebookService">PhonebookServiceReference</reference>
   </component>
   <!-- Phonebook Service Reference to Web Service at Server -->
   <reference name="PhonebookServiceReference">
      <interface.wsdl interface="http://www.hendrik.org/phonebook #wsdl.interface(PhonebookService)
         wsdl:wsdlLocation="http://www.hendrik.org/phonebook wsdl/PhonebookService.wsdl" />
      <binding.ws endpoint="http://www.hendrik.org/phonebook #wsdl.endpoint(PhonebookService/PhonebookServiceSOAP)"
         location="wsdl/PhonebookService.wsdl" />
   </reference>
</composite>
```

Listing 30: Phonebook Webclient Composite configuration file.

```xml
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
   xmlns:wsdli="http://www.w3.org/2006/01/wsdl-instance"
   name="hendrik-edm-project-webclient">
   <!-- Import Project Factory -->
      factory="org.hendrik.edm.project.ProjectFactory"/>
   <!-- EDM Project Service Implementation at Client Side -->
   <component name="EdmProjectServiceComponent">
      <implementation.java class="org.hendrik.edm.project.webclient.services.project.EdmProjectServiceImpl"/>
      <reference name="edmProjectService">EdmProjectServiceReference</reference>
   </component>
   <!-- EDM Project Service Reference to Web Service at Server -->
   <reference name="EdmProjectServiceReference">
</composite>
```
Listing 31: EDM Project Webclient Composite configuration file.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
            xmlns:rmi="http://incubator.apache.org/tuscany/xmlns/binding/rmi/1.0-incubator-M2"
            name="hendrik-edm-project-client">
  <!-- Import Project Factory -->
                   factory="org.hendrik.edm.project.ProjectFactory"/>

  <!-- Project Service Component -->
  <component name="EdmProjectServiceComponent">
    <implementation.java class="org.hendrik.edm.project.client.services.project.EdmProjectServiceImpl"/>
    <reference name="extService">EdmProjectRmiService</reference>
  </component>

  <!-- Project RMI Service -->
  <reference name="EdmProjectRmiService">
    <interface.java interface="org.hendrik.edm.project.EdmProjectService"/>
    <rmi:binding.rmi host="localhost" port="1099" serviceName="EdmProjectRmiService"/>
  </reference>
</composite>
```

Listing 32: EDM Project Swing Client Composite configuration file.
D Contents of DVD-ROM

The given DVD contains the following data:

1. Thesis in the PDF format with different levels of compression.

2. Most References used while compiling this document.

3. The Implementation:
   a) Source Code of all five Composites.
   b) Binaries of all five Composites.
   c) Required Derby Databases.
   d) Javadoc to all five Composites.
   e) Maven Site for all five Composites.
   f) A description on how to run these Composites.

4. Basic Tools:
   a) Java Development Kit 5.0.
   b) Maven including its repository.
   c) Tomcat Server.
   d) Derby Database Server.
Glossary, List of Acronyms

The important terms used in this thesis are defined here. Frequently used acronyms and abbreviations are also explained within the following glossary. Terms are arranged in alphabetical order – acronyms are sorted according to the acronym itself.

Application-To-Application (A2A) The communication of one software application with another. The applications may be located at different domains of ownership.

Asynchronous JavaScript and XML (AJAX) A web development technique for the creation of interactive Web applications that allow partial page updates due to the information being exchanged behind the scenes.

Application Programming Interface (API) The interface that a computer system, library or application provides in order to allow requests for services to be made of it by other computer programs, and/or to allow data to be exchanged between them.

Architecture of Integrated Information Systems (ARIS) A method to analyze processes and adopt a holistic view of process design, management, workflow, and application processes.

Business-To-Business (B2B) Typically stands for relations between at least two companies.

Business Activity Monitoring (BAM) A software that aids in monitoring of business processes.

Business Process Diagram (BPD) A flowchart-like diagram to describe business processes produced on the basis of the BPMN specification.

Business Process Execution Language (BPEL) An orchestration language that describes the process flow of complex business processes. It was first published by the OASIS in August 2002.


WS-BPEL Extension for People (BPEL4People) A WS-BPEL extension that makes it possible to include human user interactions within BPEL. It was published in July 2005.

Business Process Execution Language for Web Services (BPEL4WS) The name of the first version of the BPEL standard. See “BPEL” for more information.

BPEL for Java technology (BPELJ) A WS-BPEL extension that allows the use of the Java programming language within BPEL process descriptions. It was published in March 2004.

Business Process Management (BPM) A management concept that addresses how organizations may identify, model, develop, deploy, manage, and analyze their business processes.

Business Process Modeling Language (BPML) An orchestration language developed by the BPMI.org.

Business Process Modeling Notation (BPMN) A process modeling notation developed by the BPMI with wide industry support. It is based on Petri nets and built to close the gap between the business analyst and the software developer. It was finally adopted as a standard by the OMG in February 2006.

Business Process Management System (BPMS) A generic software system that may serve all activities of the business process life cycle of BPM.

Business Process Query Language (BPQL) A management interface to a business process management infrastructure that includes a process execution facility (process server) and a process deployment facility (process repository).

Business Process Reengineering (BPR) An approach attempting to optimize existing business processes with respect to efficiency.

Business Process Schema Specification (BPSS) See ebBP.

Complex Web Service A Web service, which is composed of several other Web services. Some additional process logic is provided. A complex Web service may be treated as a simple Web service after composition.

Composite Application An Application that is put together through composition, usually of Web services.

Composite Service See “Complex Web Service”.

Common Object Request Broker Architecture (CORBA) A standard for software components, created and controlled by the Object Management Group (OMG). It defines APIs, communication protocol, and object/service information models to enable heterogeneous applications written in various languages to run on various platforms and interoperate. First developments were started in October 1991; it is currently available in version 3.0.2 published in December 2002.

Data Access Service (DAS) The mediator between the Data Source and the Client for data access as part of the SDO specification.

Document Type Definition (DTD) A specification written in SGML containing information about the format of a particular document.

Enterprise Application Integration (EAI) An early concept for business integration along the whole value chain, which may be distributed between different applications and platforms; however, it suffers from the usage of the hub-and-spoke architecture.

ebXML Business Process Specification Schema (ebBP) A process description language as part of the ebXML family. It is currently under standardization with the OASIS.
Electronic Business using Extensible Markup Language (ebXML) A family of standards, the purpose of which is to provide an XML-based and open infrastructure for the global use of electronic business information.

Event-Driven Architecture (EDA) A software architecture that defines how systems may be engineered and designed to sense and respond to events.

Enterprise Edition (EE) A Java programming platform, as part of the Java Platform, for developing and running distributed multi-tier architecture Java applications, based largely on modular software components running on an application server.

Enterprise Information System (EIS) Any kind of computing system within an organization that provides a technology platform that integrates and coordinates their business processes.

Enterprise Java Bean (EJB) A Java programming technology that accomplishes persistent data retention, the definition and implementation of operations to access this data, and the exposure of Web services to invoke these operations.

Event-Driven Process Chain (EPC) A process modeling notation developed at the University of Saarbruecken in 1992 by Prof. Scheer. Today, it is used within the ARIS framework.

Entity Relationship (ER) A model or diagram to describe a piece of the real world through connecting a set of objects and relationships. This model is mainly used in database design.

Enterprise Resource Planning (ERP) An integrated business management system that includes all aspects and computerization methods that are needed to effectively plan and manage a business.

Enterprise Service Bus (ESB) A software architecture construct that provides a distributed middleware infrastructure consistent with the principles of SOA. It is a central integration layer allowing the communication of different services.

HyperText Markup Language (HTML) A markup language for the creation of Web pages.

Hypertext Transfer Protocol (HTTP) A protocol for the Internet to transfer hypertext requests and information between servers and browsers.

Secure Hypertext Transfer Protocol (HTTPS) A protocol for the Internet that provides safe data transmission by encrypting and decrypting information sent over the Internet.

Information Technology (IT) A broad subject concerned with the use of technology in managing and processing information.

JavaScript A scripting language based on the concept of prototype-based programming. The language is best known for its use in websites.

Java API for XML-based Remote Procedure Calls (JAX-RPC) A Java technology that allows for the development of SOAP-based interoperable and portable Web services.

Java API for XML Web Services (JAX-WS) A Java programming language API for the creation of Web services.
Java Business Integration (JBI) A specification representing an approach to implementing an SOA. It is mainly be used to develop an ESB.

Java EE Connector Architecture (JCA) A Java-based technology solution for connecting application servers and EISs as part EAI solutions.

Java Messaging Service (JMS) A messaging standard by Sun Microsystems that allows application components based on the Java 2 Platform to create, send, receive, and read messages.

JSON Remote Procedure Call (JSON-RPC) A remote procedure call protocol encoded in JSON.

Kerberos A computer network authentication protocol that allows individuals to communicate over an insecure network to prove their identity to one another in a secure manner. Kerberos prevents eavesdropping or replay attacks, and ensures the integrity of the data. Its designers at the Massachusetts Institute of Technology (MIT) aimed primarily at a symmetric key cryptography that requires a trusted third party.

Message-Oriented Middleware (MOM) A category of inter-application communication software that generally relies on asynchronous message-passing as opposed to a request/response metaphor.

Namespaces in XML A W3C specification allowing to combine various XML vocabularies within one XML document with the goal to prevent overlapping elements.

Network Accessible Service Specification Language (NASSL) IBM’s first service description language, superseded by WSDL in 2000.


Object Management Group (OMG) An open membership, not-for-profit consortium that produces and maintains computer industry specifications for interoperable enterprise applications. For more information, see http://www.omg.org/.

Hypertext Preprocessor (PHP) A programming language designed for producing dynamic Web pages.

Quality of Service (QoS) The measure of service quality provided to the user.

Remote Method Invocation (RMI) A Java application programming interface for RPCs which allows Java objects stored in the network to be run remotely.

Remote Procedure Call (RPC) Call of a function module that runs in a different system (destination) from the calling program. Also Remote Function Call (RFC).

Service Component Architecture (SCA) A specification suited for the delivery of applications that conform with the principles of SOA.
Service Description Language (SDL) Microsoft’s first service description language, superseded by WSDL in 2000.

Service Data Object (SDO) A specification and an API for flexible data handling independent of the query language and database used.

Simple Mail Transfer Protocol (SMTP) A protocol to exchange emails over the Internet, established in 1982.

Service-Oriented Architecture (SOA) A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations.

SOAP A simple XML-based message format to transport information between Web services, introduced by the W3C. During its first developments, it was short for “Simple Object Access Protocol” but also interpreted as “Service Oriented Architecture Protocol”, but since SOAP version 1.2, SOAP is no longer an acronym and thus used as an independent name.

Spring A popular platform used to construct Java applications. It aims at reducing the complexity of the programming environment. In particular, Spring provides a runtime container that provides dependency injection so that application components can avoid the need to program directly to middleware APIs.

Standard Query Language (SQL) Commonly used language used to create, modify, retrieve, and manipulate data from relational databases.

Secure Sockets Layer (SSL) A cryptographic protocols which provides secure communications on the Internet.

Total Cost of Ownership (TCO) The total amount of money required to own and maintain a system.

Universal Description, Discovery, and Integration (UDDI) An open and XML-based directory service mostly used for finding Web services. When interrogated by SOAP messages, it provides access to WSDL documents. It is sponsored by OASIS.

Unified Modeling Language (UML) An object modeling and specification language used in software engineering. It contains a set diagrams for different purposes. Currently, it is available in its version 2.0.

United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT) One of the United Nations bodies with the mission to improve the ability of business, trade and administrative organizations, from developed, developing and transitional economies, to exchange products and relevant services effectively – and so contribute to the growth of global commerce. See http://www.unece.org/cefact/ for more details.

Unicode An industry standard designed to allow text and symbols from all of the writing systems of the world to be consistently represented and manipulated by computers.
Uniform Resource Identifier (URI) A more general form than URL for addressing Web resources. It consists of two parts - a protocol such as HTTP(S) or FTP and a path. The path contains both the host name (and possibly the port number) as well as the server path.

World Wide Web Consortium (W3C) An international consortium where member organizations, a full-time staff, and the public work together to develop Web standards. See http://www.w3.org for more details.

Windows Communication Foundation (WCF) A communication subsystem developed by Microsoft to enable applications, in one machine or across multiple machines connected by a network, to communicate. It merely represents a method to realize SOA.

Workflow Management Coalition (WFMC) A non-profit, international organization of workflow vendors, users, analysts and university/research groups, founded in August 1993. Their mission is to promote and develop the use of workflow through the establishment of standards for software terminology, interoperability and connectivity between workflow products. For more details see http://www.wfmc.org.

Web Services Addressing (WS-Addressing) A widely-used extension for the SOAP message header that provides transport-neutral mechanisms to address Web services and messages.

Web Services Atomic Transaction (WS-AtomicTransaction) A specification that provides the definition of the atomic transaction coordination type that is to be used with a WS-Coordination specification. It is used for short-running activities with ACID properties.

Web Services Authorization Language (WS-Authorization) An upcoming specification to describe how access policies for a Web service are specified and eventually managed.

Web Services Business Process Execution Language (WS-BPEL) A language to specify the orchestration of Web services.

Web Services Business Activity (WS-BusinessActivity) A specification providing the definition of the business activity coordination type that is to be used with a WS-Coordination specification. It is used for long-running business transactions.

Web Services Choreography Description Language (WS-CDL) A language with which the choreography of Web services may be specified.

Web Services Coordination (WS-Coordination) A QoS specification describing an extensible framework for providing protocols that coordinate the actions of distributed applications.

Web Services Federation Language (WS-Federation) A W3C specification describing how to establish and manage trust relationships in a heterogeneous federated environment.

Web Services Interoperability (WS-I) A group dedicated to the interoperability concerns of disparate platforms. They do not produce technology standards. Instead, this organization provides profile documents that establish a tried and tested collection of standards.

Web Services Manageability (WS-Manageability) A specification that introduces the general concepts of a manageability model in terms of manageability topics and the aspects used to define them.

Web Services Metadata Exchange (WS-MetadataExchange) A distributed discovery specification for Web services. The standard is applied to directly inquire the endpoints which returns its own metadata.
Web Services Policy Framework (WS-Policy) A specification out of the Web services security family that makes security policies of Web services available to service consumers.

Web Services Policy Attachment (WS-PolicyAttachment) A part of the WS-Policy specification family. It defines how to flexibly attach such policies to existing Web services.

Web Services Privacy (WS-Privacy) An upcoming specification to describe syntax and semantics for binding organizational privacy policies to Web services and instances of data in messages.

Web Services Reliable Messaging Protocol (WS-ReliableMessaging) A protocol that allows messages to be delivered reliably between distributed system applications. It was published in February 2005.

Web Services Secure Conversation Language (WS-SecureConversation) A W3C specification providing secure conversations between services by helping establish and share security tokens amongst them.

Web Services Security (WS-Security) A W3C specification providing a set of mechanisms to secure SOAP message exchange.

Web Services Trust Language (WS-Trust) A W3C specification describing an extensible model for setting up and verifying trust relationships for secure message exchange between Web services.

Web Services Architecture (WSA) An interoperability architecture that identifies and describes global elements of the global Web services network that are required in order to ensure interoperability between Web services.

Web Service Choreography Interface (WSCI) An XML-based interface description language that describes the flow of messages exchanged by a Web Service participating in choreographed interactions with other services.

Web Services Conversation Language (WSCL) A service composition language developed by Hewlett-Packard. It defines the external, visible behavior of a Web service as an abstract interface from the perspective of one of the participants.

Web Services Description Language (WSDL) A major specification within the Web services stack. It describes the public interface to a Web service, functional characteristics, and other crucial information.

Web Services Distributed Management (WSDM) An OASIS standard that seeks to unify management infrastructures by providing a vendor, platform, network, and protocol neutral framework for enabling management technologies to access and receive notifications of management-enabled resources.

Web Services Flow Language (WSFL) An XML-based format describing the composition of Web services specifying usage and interaction patterns. It was developed by the WfMC and was superseded by BPEL4WS in August 2002.

Web Services Management Framework (WSMF) A logical architecture for managing computing resources, including Web services themselves, through Web services.

World Wide Web (WWW) A computer network consisting of a collection of Internet sites that offer text, graphics, sound, and animation resources through the hypertext transfer protocol.
**X.509** A standard for public key infrastructure mainly specifying standard formats for public key certificates and a certification path validation algorithm. It is under the supervision of the International Telecommunication Union Telecommunication Standardization Sector (ITU-T).

**Web Services for Business Process Design (XLANG)** A business process language based on XML and developed in 2001 by Microsoft for their BizTalk server. It adds to the Web service stack. In August 2002, it was superseded by BPEL4WS.

**Extensible Markup Language (XML)** A markup language, recommended by the W3C in 1996 and released as a standard in 1998. It is used to define special markup languages by describing information where the data itself can be contained in the same file.

**XML Encryption** A W3C specification allowing the encryption of specific parts in an XML document.

**XML Schema** A W3C specification to describe the structure of an XML document.

**XML Signature** A W3C specification making it possible to securely verify the origin of a message through digital signatures.

**XML Path Language (XPath)** A short syntax for addressing portions of an XML document.

**XML Process Definition Language (XPDL)** A format that provides an XML file format that may be used to interchange process models between tools.

**XML Schema Definition (XSD)** An XML Schema instance, that specifies how to formally describe the elements in an XML document.

**Extensible Stylesheet Language (XSL)** A family of languages which allows the description of how files may be encoded in the XML standard and how these may be formatted or transformed.

**XSL Formatting Objects (XSL-FO)** An XML language out of the XSD family specifying the visual formatting of an XML document.

**XSL Transformations (XSLT)** An XML language out of the XSD family for transforming XML documents.

**Yet Another Workflow Language (YAWL)** A process modeling notation, which is based on Petri nets and comprehensively supports all workflow pattern defined by Prof. van der Aalst.
van der Aalst a. ter Hofstede 2005


van der Aalst et al. 2003


Adams et al. 2006

ADAMS, Matthew; ANDREI, Cezar; BARACK, Ron; BLOHM, Henning; BOUTARD, Christophe; BRODSKY, Stephen; BUDINSKY, Frank; BUENNIG, Stefan; CAREY, Michael; DOUghan, Blaise; GROVE, Andy; HALASEH, Omar; HARRis, Larry; MERSEwSKy, Ulf von; MOE, Shawn; NALLY, Martin; PREotiuc-PIETRO, Radu; ROWLEY, Mike; SAMSON, Eric; TAYLor, James; THIEFAINE, Arnaud: Service Data Objects For Java Specification / IBM Corporation, Oracle, BEA Systems, SAP AG, Rogue Wave Software, Xcalia. Version: November 2006. http://www.osoa.org/download/attachments/36/Java-SDO-Spec-v2.1.0FINAL.pdf?version=1, Last checked: 15/09/2006. 2006 (2.1.0). – Specification

Anderson et al. 2005a

ANDERSON, Steve; BOHREN, Jeff; BOBEZ, Toufic; CHANLIAU, Marc; DELLA-LIBERA, Giovanni; DIXON, Brendan; GARG, Praerit; GUDGIN, Martin; HADA, Satoshi; HALLAM-BAKER, Phillip; HONDO, Maryann; KALER, Chris; LOCKHART, Hal; MARTHHERUS, Robin; MARUYAMA, Hiroshi; NADALIN, Anthony; NAGARATNAM, Nataraj; NASH, Andrew; PHILPOTT, Rob; PLATT, Darren; PRAFULLCHANDRA, Hemma; SAHU, Maneesh; SHEWCHUK, John; SIMON, Dan; SRINIVAS, Davanum; WAINGOLD, Elliot; WAITE, David; WALTER, Doug; ZOLFONOON, Riaz: Web Services Trust Language (WS-Trust) / IBM, Microsoft, Sun, Verisign, BEA. Version: February 2005. ftp://www6.software.ibm.com/software/developer/library/ws-trust.pdf, Last checked: 08/22/2006. 2005. – Specification

Anderson et al. 2005b

ANDERSON, Steve; BOHREN, Jeff; BOBEZ, Toufic; CHANLIAU, Marc; DELLA-LIBERA, Giovanni; DIXON, Brendan; GARG, Praerit; GUDGIN, Martin; HADA, Satoshi; HALLAM-BAKER, Phillip; HONDO, Maryann; KALER, Chris; LOCKHART, Hal; MARTHHERUS, Robin; MARUYAMA, Hiroshi; NADALIN, Anthony; NAGARATNAM, Nataraj; NASH, Andrew; PHILPOTT, Rob; PLATT, Darren; PRAFULLCHANDRA, Hemma; SAHU, Maneesh; SHEWCHUK, John; SIMON, Dan; SRINIVAS, Davanum; WAINGOLD, Elliot; WAITE, David; WALTER, Doug; ZOLFONOON, Riaz: Web Services Secure Conversation Language (WS-SecureConversation) / IBM, Microsoft, Sun, BEA, Verisign, OpenNetwork, Layer 7, Computer Associates, Oblix, Reactivity, RSA Security, Ping Identity, Actional.
Andrews et al. 2003

Arkin 2002

Arkin et al. 2002

Austin et al. 2004

Bajaj et al. 2006
Bajaj, Siddharth; Box, Don; Chappell, Dave; Curbera, Francisco; Daniels, Glen; Hallam-Baker, Phillip; Hondo, Maryann; Kaler, Chris; Langworthy, Dave; Nadalin, Anthony; Nagaratnam, Nataraj; Prafullchandra, Hemma; Riegen, Claus von; Roth, Daniel; Schlimmer, Jeffrey; Sharp, Chris; Shewchuk, John; Vedamuthu, Asir; Yalcinalp, Umit; Orchard, David: Web Services Policy 1.2 - Framework (WS-Policy) / W3C, VeriSign, Microsoft, Sonic Software, IBM, SAP, BEA Systems. Version: April 2006. http://www.w3.org/Submission/2006/SUBM-WS-Policy-20060425/, Last checked: 07/12/2006. 2006 (1.2). – Specification

Bajaj et al. 2003

Ballinger et al. 2004
Ballinger, Keith; Box, Don; Davanum, Srinivas; Ferguson, Don; Graham, Steve; Liu, Canyang K.; Leymann, Frank; Lovering, Brad; Nadalin, Anthony; Nottingham, Mark; Orchard, David; Riegen, Claus von; Sedukhin, Igor; Shewchuk, John;

Banerji et al. 2002
Banerji, Arindam; Bertolini, Claudio; Beringer, Dorothea; Chopella, Venkatesh; Govindarajan, Kannan; Karp, Alan; Kuno, Harumi; Lemon, Mike; Pogossiants, Gregory; Sharma, Shamik; Williams, Scott: Web Services Conversation Language (WSCL) 1.0 / Hewlett-Packard. Version: March 2002. http://www.w3.org/TR/wsc110/, Last checked: 06/23/2006. 2002 (1.0). – Specification

Barack et al. 2006

Beatty et al. 2003

Beisiegel et al. 2006a

Beisiegel et al. 2006b

Bider 2000

Bieberstein et al. 2005
Bieberstein, Norbert; Bose, Sanjay; Fiammane, Marc; Jones, Keith; Shah,

Blow et al. 2004

Booth et al. 2004

Booth a. Liu 2006

BPML.org a. OMG 2006

Bray et al. 2004

Burdett a. Kavantzas 2004

Cabrera et al. 2005a

Cabrera et al. 2005b
Cabrera, Luis F.; Feingold, George Copeland M.; Freund, Robert W.; Johnson, Tom Freund J.; Joyce, Sean; Kaler, Chris; Klein, Johannes; Langworthy, David; Little, Mark; Nadalin, Anthony; Newcomer, Eric; Orchard, David; Robinson, Ian; Shewchuk, John; Storey, Tony: Web Services Atomic Transaction (WS-AtomicTransaction) / IBM, BEA Systems, Microsoft, Arjuna, Hitachi, IONA.
Bibliography


Cabrera et al. 2005c

Chappell 2004

Chinnici et al. 2006

Christensen et al. 2001

Clement et al. 2004

Cowan a. Tobin 2004

Davis et al. 2006

DeMichiel a. Keith 2006

Dubray
Dubray et al. 2006

Endrei et al. 2004

Erl 2005

Gilpin et al. 2006
GILPIN, Mike; HEFFNER, Randy; RYMER, John R.: Emerging SOA Standards: SCA/SDO And JBI. 2006. – Forschungsbericht

Gudgin et al. 2003a

Gudgin et al. 2003b

Gudgin et al. 2006

Hammer a. Champy 1993

Havey 2005

Holdsworth et al. 2006
Bibliography

Hurwitz 2005

Kavantzas et al. 2005

Keen et al. 2004

Keller et al. 1992

Kloppmann et al. 2005a

Kloppmann et al. 2005b

Lawrence et al. 2006

Leymann 2001

MacKenzie et al. 2006
MACKENZIE, Matthew C.; LASKEY, Ken; MCCABE, Francis; BROWN, Peter; METZ, Rebekah: Reference Model for Service Oriented Architecture / OASIS. Version: August
Bibliography


Mitra 2003

Newcomer a. Lomow 2004

OMG 2005

Peltz 2003

de Putte a. Gavin 2005

Raj et al. 2006

Schulte 2003

Ten-Hove a. Walker 2005

Thatte 2001

Thompson et al. 2005
THOMPSON, Jess; NATIS, Yefim V.; PEZZINI, Massimo ; MALINVERNO, Paolo: Predicts 2006: The Strategic Impact of SOA Broadens / Gartner Inc. and/or its Affiliates. 2005 (ID Number: G00131143). – Forschungsbericht

Vollmer a. Gilpin 2006
VOLLMER, Ken; GILPIN, Mike: The Forrester Wave: Enterprise Service Bus, Q2 2006. 2006. – Forschungsbericht
Weaver 2005

WfMC 2006

Wilkes 2002

Williams a. Daniel 2006
Index

AJAX, 103, 104
Apache Tuscany, 93

BAM, 7
BPEL, 58
BPEL-SPE, 61
BPEL4People, 61
BPEL4WS, 59
BPELJ, 61
BPM, 7
BPML, 65
BPMN, 32
BPMS, 7
BPSS, 67
Business Process, 6

Choreography, 56, 63
Client-Server, 14
Composition, 50, 56

DAS, 87
Dojo, 103, 104
DTD, 38

ebBP, 67
ebXML, 67
EIS, 84
EJB, 75
EPC, 29
ER Model, 26

Flowchart, 26
Function Tree, 26

HTML, 103, 104
HTTP, 39
HTTPS, 39

JavaScript, 103, 104
JAX-RPC, 74
JAX-WS, 74
JBI, 76
JCA, 84
JEE 5, 74

JMS, 39
JSON-RPC, 103, 104
JSR-109, 74

Mainframe, 14
Monolithic, 14
Mule, 76

N-Tier, 14
Namespaces in XML, 38

OASIS, 15, 17
Open ESB, 76
Orchestration, 56, 58
Organizational Chart, 26

Petri Net, 27
Process Integration, 9
Process Notation, 26

Quality of Service, 50

SCA, 78
SCA Assembly, 80
SCA Binding, 84
SCA Component, 80
SCA Composite, 81
SCA Implementation, 86
SCA Policy, 85
SCA Reference, 81
SCA Service, 81
SCA System, 81
SDO, 87
ServiceMix, 76
SMTP, 39
SOA, 11, 14, 17
SOAP, 40

Three-Tier, 14

UDDI, 47
UML Activity Diagram, 28
Unicode, 38

W3C, 15
Index

WS-Addressing, 42
WS-AtomicTransaction, 53
WS-Authorization, 51
WS-BusinessActivity, 53
WS-CDL, 63, 64
WS-Coordination, 53
WS-Federation, 51
WS-I, 15
WS-Manageability, 54
WS-MetadataExchange, 48
WS-Policy, 45, 51
WS-PolicyAttachment, 45
WS-Privacy, 51
WS-ReliableMessaging, 39, 53
WS-SecureConversation, 51
WS-Security, 51
WS-Trust, 51
WSCI, 63
WSCL, 66
WSDL, 43
WSDM, 54
WSFL, 59
WSMF, 54

XLANG, 59
XML, 38
XML InfoSet, 38
XML Schema, 38
XPath, 38
XPDL, 67
XSL, 38
XSL-FO, 38
XSLT, 38

YAWL, 31
Confirmation

I, Hendrik Mueller, hereby confirm that I completed this thesis independently and that I only used the references and sources mentioned in this thesis.

Bangalore, India November 30th, 2006