Degree project

Challenges of Service Interchange in a cross cloud SOA Environment

Author: Heiko Großkopf
Supervisor: Prof. Dr. Welf Löwe
External Supervisor: Prof. Dr. Andreas Heberle, University of Applied Sciences Karlsruhe

Date: 2015-01-06
Course Code: 4DV01E, 60 credits
Level: Master
Department of Computer Science
Abstract

This Master’s Thesis examines and documents challenges related to the flexible interchange of web services within a cross-cloud Service Oriented Computing scenario (SOC).

Starting with a theoretical approach, hypotheses are defined and processed to create testing scenarios for a practical examination. Both examinations are used to identify possible challenges. Next, encountered challenges are described, discussed and classified. Lastly, solution approaches to identified challenges are presented. The solution approaches concern related topics, such as service standardization, semantic methods, heuristics, and security/trust mechanisms. Several approaches to different challenges are reviewed in this particular context, to present an overview for future research on the subject.

It is remarkable that there will be more service standardization in the future, but to achieve full automation it will be, on the long run, necessary to evolve and adopt more sophisticated solution approaches such as semantic methods or heuristics.

This work is embedded into the framework of a research cooperation between the Linnaeus University Växjö and the University of Applied Sciences Karlsruhe. Results however are also applicable to other research scenarios.

Keywords: Service Oriented Computing (SOC), SOA, Web services, Semantic web services, Service Interchange, Service Interoperability, Dynamic Binding, SOA Security, Trust
# Table of Contents

1 Introduction .................................................................................................................. 1  
   1.1 Background and Motivation .................................................................................. 1  
   1.2 Central Objectives of this Work .......................................................................... 2  
   1.3 Approach and Overview ...................................................................................... 2  
   1.4 Delimitation ......................................................................................................... 3  
   1.5 Outline .................................................................................................................. 3  

2 Fundamentals of relevant Technologies .................................................................... 4  
   2.1 Service Oriented Computing ................................................................................ 4  
   2.2 Service Oriented Architecture (SOA) ................................................................. 4  
   2.3 Cloud Computing .................................................................................................. 6  
   2.4 Web Services ........................................................................................................ 7  
      2.4.1 Basic Principles of SOAP Web Services ......................................................... 7  
      2.4.2 Web Service Standards Stack ...................................................................... 9  
      2.4.3 WS-BPEL Basics .......................................................................................... 16  
      2.4.4 The Service Component Architecture (in JDeveloper) ................................. 17  
   2.5 RESTful Web Services .......................................................................................... 18  
   2.6 Current State of Research and Research Context .............................................. 19  

3 Conceptual Approach ............................................................................................... 21  
   3.1 Definition of Hypotheses ..................................................................................... 21  
   3.2 Web Service Market Research Results .................................................................. 23  
   3.3 Definition of the Testing Scenarios ....................................................................... 24  
   3.4 Requirements ....................................................................................................... 26  
      3.4.1 Requirements for Services .......................................................................... 26  
      3.4.2 Requirements for ”semi-mockup” Services ................................................... 27  
      3.4.3 Requirements for the testing process ........................................................... 27  
      3.4.4 Further Requirements .................................................................................. 28  
   3.5 The Implementation Context ............................................................................... 28  
   3.6 The Implementation Approach ............................................................................. 29  
   3.7 An Example Scenario: Online Stock Brokerage Company ................................ 30  

4 The Practical Examination of Service Interchange ............................................... 31  
   4.1 Specification of the Testing Process ...................................................................... 31  
   4.2 Implementation and Identification of applicable Services .................................... 32  
      4.2.1 Self-defined “semi-mockup” Services ............................................................ 33  
      4.2.2 Xignite Global Quote (External Service) ....................................................... 41  
   4.3 Service Interchange and Testing .......................................................................... 47  
      4.3.1 Scenario 1: Same Interface, Same Service .................................................... 47  
      4.3.2 Scenario 2: Different Interface, Same Service ............................................. 48  
      4.3.3 Scenario 3: Different Interface, Related Service .......................................... 49  
      4.3.4 Scenario 4: Different Interface, Different Service ........................................ 58  
      4.3.5 Scenario 5: Service Interchange within a composed Application ................ 58  
   4.4 Generalization of Identified Challenges ............................................................... 65
5  Solution Approaches ........................................................................................................... 69
  5.1 Integration and Semantics ................................................................................................. 69
  5.1.1 Integration of heterogeneous Web Services ............................................................... 69
  5.1.2 Semantic Approaches of Service Integration ............................................................ 72
  5.1.3 Comprehensive Approaches to the Integration and Discovery of Web Services .......... 75
  5.2 SOA Security .................................................................................................................... 76
  5.2.1 Basic SOA Security ...................................................................................................... 77
  5.2.2 Authentication and Authorization .............................................................................. 77
  5.2.3 Confidentiality ............................................................................................................. 78
  5.2.4 Integrity and Non-Repudiation .................................................................................... 79
  5.2.5 Password Digest – One Time Passwords .................................................................. 79
  5.2.6 The Authentication System Kerberos ....................................................................... 80
  5.2.7 Security as a Service ................................................................................................... 80
  5.2.8 Security Policies ......................................................................................................... 82
  5.2.9 WS-SecurityPolicy ...................................................................................................... 82
  5.3 Trust related SOA Security ............................................................................................. 83
  5.3.1 Different Approaches of Trust Evaluation ................................................................. 83
  5.3.2 A Trust Model for Service Certification ................................................................... 84
  5.3.3 End-to-End Security Approach ............................................................................... 85
  5.4 REST and BPEL ............................................................................................................... 85
  5.5 Evaluation of Hypotheses ............................................................................................... 86
6  Conclusion ............................................................................................................................ 89
  6.1 Review ............................................................................................................................. 89
  6.2 Evaluation of Objectives ............................................................................................... 89
  6.3 Core Insights .................................................................................................................... 89
  6.4 Transfer of Insights to a Service Interchange Scenario ................................................ 90
  6.5 Possible Subjects for future Research ........................................................................... 90
  6.6 Future Outlook ............................................................................................................... 90
  6.7 Reflection on ethical and social Aspects of the examined Subject .............................. 91

References ............................................................................................................................. 93
Table of Figures ...................................................................................................................... 99
List of Tables .......................................................................................................................... 101
Statement of Authorship ......................................................................................................... 102
Appendix ................................................................................................................................. 103
1 Introduction

The following chapter introduces and outlines this academic paper. First the subject of this Master’s Thesis is described and its significance justified. This includes an explanation of the project’s academic background. Next the central objectives are presented in more detail as the subject is delimited. Finally, all chapters contained within this thesis are introduced briefly.

1.1 Background and Motivation

Based on the increasing success and use of technologies like Service Oriented Computing (SOC), it must be assumed that, in the near future, there will be an open, comprehensive, internet-based market of web and cloud services. Consequently, this also leads to a further developing symbiosis of cloud computing and Service Oriented Architecture (SOA). In such a market it is expected that web services will not substantially differ from each other functionally, but might instead differ according to their non-functional properties or Quality of Service. The latter might be affected through various circumstances, such as the client’s environmental context or the performance of the provider itself [1].

A market like this would furthermore enable companies to individually implement business processes with interchangeable services that provide the best Quality of Service. They would be able to choose from a pool of different external service providers. Service consumers would also benefit from the advantages of cloud computing, whilst avoiding some commonly related risks such as, for example, a vendor lock-in [1].

The Quality of Service is usually specified in the Service Level Agreements (SLAs), which serve as a contract to assure a fixed level of service quality. SLAs alone however are not expected to be trustworthy indicators of non-functional quality characteristics, because these also depend on external circumstances. The physical distance between a client and a server, for example, has an impact on the response time of a service, and service-request peak times may affect the availability and throughput of a service during certain times of the day. Another argument to not only rely on SLAs is the fact that SLA figures are provided by the service providers themselves and may also serve as marketing material. To better evaluate web services, aside from relying on SLAs, independent service brokers employ innovative learning algorithms to identify the best-fit services available [1].

In spite of this positive outlook it is also expected that the expanded use of external services in SOC scenarios will inevitably lead to further challenges, risks, and consequently to an additional effort requirement.

To dynamically only implement services that provide a best-fit quality of non-functional properties, a regular service interchange has to be expected and needs to be facilitated. Services may not differ substantially on the functional level in the future, but it is likely that they might differ slightly due to a shortfall of service standardization. Additional challenges concern the technical integration and interoperability of services. Security issues are also very likely, as external service providers provide services as a black box, and internal security standards of the service provider might not be transparent to the consumer.

To provide an overview of the real amount of expected challenges during a service interchange, this paper covers a theoretical and practical examination of the matter. The objectives of this work are further described in the following sub-chapter.
1.2 Central Objectives of this Work

The scenario described above, is also motivated within a related PhD project that itself is part of a research cooperation between the Linnaeus University and the University of Applied Sciences Karlsruhe [1]. The latter project requires an investigation of challenges that might be present in this scenario and consequently need to be addressed. As a consequence this master’s thesis’ academic motivation is also to contribute to this particular project. This thesis most of all investigates and summarizes applicable content that is already known to research. It puts the content into a particular context, while remaining generic enough to allow external researchers to utilize the results within different scenarios. It is therefore also useful for other academic works.

The central objectives of this work are to identify and summarize problems and challenges that might occur during the interchange of web services in the above mentioned context. It is also a goal to identify and investigate solution approaches to solve these challenges. A comprehensive investigation of all solution approaches would though be beyond the scope of this thesis. Consequently selections of related projects are given that investigate each topic in a deeper manner. It is also intended to generate entry-points for following academic projects within the scope of the cooperation. All this has not been done yet in this particular context.

To achieve these objectives, various tasks had to be carried out to accomplish certain sub-objectives. These tasks are further described in the Approach and Overview sub-chapter.

1.3 Approach and Overview

This thesis uses the academic approach of a controlled experiment to examine the sub topic of service integration and its challenges. Furthermore a literature review is conducted to identify and investigate further challenges that, due to their complexity, cannot be part of a practical experiment within the scope of this work. The following paragraphs describe the academic approach in a more detailed manner.

At first, a literature review was conducted in order to find academic reports related to the topic of service interchange. The intention was furthermore to collect information about Service Oriented Computing, Service Oriented Architecture, and web services in particular. Web service interoperability and related concepts, such as web service standards, web service standard profiles, web service security requirements, as well as the differences between RESTful and SOAP web service paradigms were researched. This theoretical approach was the basis of the “Fundamentals of relevant Technologies” - chapter and also the basis for the definition of hypotheses that describe expectations based on literature, technology, and standards, especially in the domain of service integration.

A practical examination was carried out to further examine, illustrate, and verify the identified insights and hypotheses, especially the subject of service integration. The latter is crucial and prerequisite for a successful service interchange. To perform this practical examination in scientific manner, the methodology of a controlled experiment was defined accordingly. The definition included the delimitation of the testing subject, the specification of testing scenarios and requirements, and the conduction of a web service market investigation to further define a realistic and, more importantly, feasible testing process. Afterwards a suitable developing environment had to be selected, installed, and tested. The use of the relevant software suite had to be learned.

After the web service market investigation was conducted, suitable web services could be combined to design a realistic and representative initial business process. Some of these web services had to be self-defined to examine certain scenarios. The interchange was conducted per scenario in an iterative way. All results, as well as the gener-
ated problems and challenges, were documented simultaneously. Thereafter the identified problems and challenges were generalized and categorized. The problem categories were further examined and general solution statements were made.

During and after the practical examination additional research had to be performed to identify solution approaches and to investigate solutions to current technical problems.

1.4 Delimitation
This work does not contain any content that considers the algorithms and methods to measure service quality or “Service Level Achievements”. It does not cover the automatic or manual selection process of “best-fit” services in terms of the highest quality of non-functional properties. It furthermore does not concern any data mining or computer based learning methods. These subjects may be covered in related work situated within the same research environment [1].

The subject does also not cover the subjects of governance and identity management, as it is not directly related to the actual service interchange.

Furthermore this thesis does not claim to provide detailed and complete solutions to all identified problems and challenges. The primary goal of this thesis is to give and discuss possible solution approaches, which can be used as a foundation in the future. Future research can create solutions in more detail.

1.5 Outline
This introductory chapter introduced, justified, and delimited the subject of this thesis. Furthermore the basic approach of the examination was described.

The second chapter contains the introduction to the fundamentals of related technologies that are relevant for the subject of service interchange. The fundamentals are important to get a basic understanding of Service Oriented Computing, Service Oriented Architecture, Cloud Computing and especially web services, all within the scope of this subject. The fundamentals chapter is meant to provide the reader with sufficient knowledge to understand the activities conducted during the examination. It is not intended to replace a good textbook about SOC, SOA or web services.

Chapter 3 contains the preparation activities that are connected to the conduction of the practical examination. The chapter covers the specification of hypotheses and the planning of the further examination. This includes the definition of requirements for services, processes, and scenarios. The testing subject of the practical examination is furthermore delimited.

Chapter 4 covers the actual practical examination. This includes the creation and selection of services, the initial implementation of a testing application and process, and the actual testing of the predefined scenarios. After the testing, generalized challenges are categorized and discussed.

Chapter 5 includes the development of solution approaches to solve the identified challenges. Furthermore this chapter also includes solution approaches to challenges that were identified in the theoretical research and were derived from the predefined hypotheses of chapter 3.

Chapter 6 summarizes and concludes the findings and outcome of this thesis. Finally, a perspective of the future is discussed and evaluated.
2 Fundamentals of relevant Technologies

The last chapter introduced and motivated the general subject of this thesis, the following chapter introduces the reader to the fundamental technologies and standards used in this work. It gives the reader a brief overview about the field of Service Oriented Computing, including the basics of Service Oriented Architecture, Cloud Computing, and, in particular, web services. The latter play a crucial part in this thesis, especially in the practical examination. Both SOAP and RESTful web services are regarded, but the focus lies on SOAP-based web services. In addition an understanding of common web service standards and the extended web service Stack is very important for this investigation and will be introduced accordingly. This includes a brief introduction into XML and XML-based languages such as WSDL, SOAP, BPEL, XSLT or XPath. Finally the concept of the Service Component Architecture will be introduced.

This chapter however cannot be considered as a replacement for a fundamentals textbook about Service Oriented Computing or web services. Literature references are given for further information.

2.1 Service Oriented Computing

The subject of this thesis is situated in a Service Oriented Computing (SOC) environment. That means that the service oriented programming paradigm is followed. Service orientation hereby describes available computer-based resources as a service. According to Erl, service oriented programming enhances the agility during the implementation of new or changing business processes by using existing and reusable IT modules or services, which can also be outsourced [2]. This outsourcing leads to the division of software development and fosters cost reduction and specialization.

SOC itself is an umbrella term. It is not just another description for Service Oriented Architecture (SOA) [3]. SOC comprises various architectural models like design principles and patterns, as well as related concepts, technologies and development frameworks. Zhang describes SOC as a very important subject for today’s information-led society [4]. SOC brings the realms of business and IT together on a service level which both realms understand, because both business and IT use services. The field of SOC furthermore comprises the examination of methods on how to enhance, create, manage and operate business services with IT in the most effective and efficient manner. In conclusion SOC is not a synonym for SOA, but encompasses various parts of the service oriented programming and design paradigms.

Service Oriented Computing is a technology with rising popularity. SOC composes IT services as building blocks in order to support whole business processes, which enables the implementation of distributed systems while saving resources and funds. SOC is not a new technology in itself, but a collection and combination of existing technologies [5].

2.2 Service Oriented Architecture (SOA)

According to Newcomer and Lomow, “A service-oriented architecture is a style of design that guides all aspects of creating and using business services throughout their lifecycle - from conception to retirement. An SOA is also a way to define and provision an IT infrastructure to allow different applications to exchange data and participate in business processes, regardless of the operating systems or programming languages underlying those applications.” [6]

A Service Oriented Architecture (SOA) is a technologically neutral architecture pattern, which uses standards to facilitate general interoperability between different tech-
nologies and architectures in distributed systems. SOA supports business processes by providing an infrastructure for the orchestration and composition of standardized, encapsulated applications (services). It enables the agile creation of composite applications by using self-contained, interchangeable and theoretically reusable services [4].

SOA is not a product but a paradigm. A programming logic which generates a modular programming architecture out of a monolithic system by packaging functional coherent functionalities into an encapsulated service. SOA enables applications to be easily extended in functionality by adding further services. Services can also be interchanged if needed e.g. if they are not available anymore or possess disadvantageous non-functional properties. The functionality in itself is not affected. A reusability of services within a SOA is aspired to save e.g. development costs for new software implementations [4].

Services can be self-developed or provided by external service providers. In both cases, they are invoked and consumed over a network such as the company’s LAN, WAN or the Internet [4].

The SOA layered model
The SOA layered model consist of several layers and describes the relations between business logic, business services and IT logic/services. The layered model exhibits two approaches, a top down approach seen from the business consumer’s perspective and a bottom up view seen from the technical side [5]. The following illustration, Figure 2.1 shows the different layers, which are further explained in the following paragraph.

![Figure 2.1: SOA Layered Model](source: Own representation, based on Papazoglou, 2008 [5])

A company consists of several business domains (e.g. procurement, manufacturing, distribution, etc.). These domains possess several core business processes, which are described one layer below. The core processes are implemented in workflows that consist of business services. These business services however are complex composite web services, which can be decomposed further in order to identify the actual infrastructure services that orchestrate the business services. The infrastructure services cannot be decomposed any further. They are implemented within a developing environment that
facilitates the creation of SOA applications. The environment is further supported by operative systems like CRM, ERP, or Legacy systems [5].

Due to their service character, SOAP based web services are well suited to implement the SOA paradigm. Alternatives for SOAP web services are RESTful services or other technologies like CORBA (Common Object Request Broker Architecture). Web services are self-contained and therefore can be used independent of the context. They can be orchestrated and composed with languages like WS-BPEL (Web Service - Business Process Execution Language) to implement and support business processes and to generate a loosely coupled composite application [5].

The web service technology is introduced in more detail in chapter 2.4.

2.3 Cloud Computing

The term Cloud computing has become a popular and widely interpreted expression. The subsequent section briefly introduces the term in relation to the subject of this thesis. It does not intend to give a full introduction about the fundamentals of cloud computing technology. This section is especially important for describing the expression ‘cross-cloud SOA environment’, which is part of the title of this thesis. The term Cloud computing was defined in the NIST Definition of Cloud Computing as follows: “Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.” [7]

Cloud providers provide solutions for the whole IT stack. Their offerings range from the provision of virtualized hardware and infrastructure (Infrastructure as a Service ‘IaaS’), to the supply of whole development platforms (Platform as a Service ‘PaaS’), to the offerings of specialized applications - all usually provided over a web browser (Software as a Service ‘SaaS’). Figure 2.2 shows a possible illustration of the IT and services stack. Everything within the product stack can be provided as a service [7].

Cloud computing, or SaaS in particular, can be seen as a suitable technology to implement the service-oriented computing paradigm online and foster reusability. Services can be distributed over the Internet and consequently used by anyone authorized to do
so. The use of multiple cloud service providers’ external services, to implement a business process, is consequently called a cross-cloud environment [9].

Cloud computing comes with benefits but also presents challenges and risks. The use of external services can be equated with the outsourcing of parts of a company’s own IT. This can include parts of the hardware infrastructure, server platforms (software), or parts of the programming logic. It can also be assumed that a part of the company’s special expertise is outsourced as well. Advantages of this outsourcing are indisputable: the federation of customer requirements and investments leverages the quality and capabilities of the service provider’s solutions. The service provider’s experts may also be more specialized in their sector than internally employed IT professionals. However, cloud computing also has some drawbacks: it is highly dependent on the network connection and on the service provider. If there is no connection to the network, or a provider cannot maintain availability, tasks normally cannot be carried out. Furthermore a service provider could willingly and unwillingly provoke a vendor-lock-in, for example by not using common standards, to indirectly promote the sales of their own service portfolio. The outsourcing of programming logic can moreover be a security concern, since sensitive data might be processed by external service providers. Providers in general need to be trustworthy [10].

Cloud services can be realized in different technological ways. According to a web service market investigation conducted in connection to this thesis, SOAP based web services and RESTful services were most commonly available. Both technologies are described in the following subchapters.

2.4 Web Services

Web services are standardized, self-contained, and machine readable software modules, which encapsulate a certain programming logic and functionality. They facilitate data exchange between different systems, while being independent to internal programming languages and protocols. Web services possess a well-defined interface, which exposes its functionality to the outside world. The service implementation stays hidden and has no influence on the technical interoperability between services [5].

The interface of a web service is described by an interface description language file (WSDL-file). This file is based on XML and depicts, among other things, the operations a service provides by defining in- and output messages and their parameters. The communication between web services is handled via a network such as the Internet and they are commonly message based. The messages are transferred via the SOAP Protocol, which is based on the common meta-language XML [5].

Web services can be orchestrated with help of the Business Process Execution Language (BPEL). BPEL allows the integration of web services via Partner Links, which represent a service in a component architecture, into a process flow. Besides this it allows further actions such as the invocation of web services, or the copying of their parameter payloads to each other [5].

Web Service based composite applications are generally deployed on standard middleware platforms [5].

2.4.1 Basic Principles of SOAP Web Services

Web services feature several characteristics and aims. Firstly they should be loosely coupled and stateless in order to be independent of the context they are used in. Secondly they should be programmable and reusable to save costs. They should also be able to flexibly realize and support business processes as building blocks. Hence they are a measure to avoid the need of unnecessary new software developments [5].
Web services differ according to their granularity. They can contain everything from a few lines of code, to a more complex module in BPEL, to a whole composite application which itself is exposed as a web service. However the granularity of a web service should be well balanced and it should be functionality coherent. It ought not to be too fine-grained to avoid too much communication overhead and not be too coarse-grained to avoid a tight coupling and consequently a high dependency [5].

A web service can be seen as a resource, which provides access to a functionality of a different application, knowing and publishing its in- and output values to enable clients to consume the service. Thereby web services are based on “Open Internet Based Standards” to maintain interoperability. Accordingly they are an enabler technology for interoperability between intra- and inter-enterprise applications. Moreover a web service can be either, client, server or peer. It can consume and provide services to fulfill its defined functionality. A web service application can be used by human users, invoked by BPEL processes, apps, other applications or other web services in any location, within or outside of the enterprise [5].

Web services can also be described by their non-functional properties such as performance and security measures, availability, geographical coverage or contact data. These non-functional properties are defined in so called Service Level Agreements (SLAs) which serve like a contract between a service provider an a service consumer. Services can also be brokered and even auctioned [5]. The service invocation process for SOAP web services can be described as visualized in the following figure 2.3.

A service provider registers its service at a service registry. The service registry is an instance for service consumers to lookup a service by sending a registry request query. The registry returns the convenient WSDL file of the service to the service consumer, enabling it to directly invoke the services operations by using SOAP messages. The role of the service registry can also be carried out by a service repository, a directory service or a service broker. A service provider can simultaneously be a service consumer and vice versa, the relation between the actors can be client/server as well as peer to peer [5].

**Simple and complex Web Services**

Web services possess the ability to invoke other services to fulfill a complex task or transaction. They can be categorized as simple or complex services and can be called synchronously or asynchronously. A synchronous service invocation works analogously
to a remote procedure call (RPC); the service requester blocks and waits for the answer of the service responder. An asynchronous service invocation implements a document based interaction between requester and responder, the requester can maintain its work and does not have to wait for the answer of the service responder. Depending on the case, a message-oriented middleware or a message queue might be in service [5].

Simple services are easy, synchronously called, informal services which implement easy request-response operations. All simple services are stateless. Due to their synchronous interaction patterns, which implement a RPC-style messaging, simple web services are rather tightly coupled. The message path is normally predefined and the binding style is static. The objective of a simple web service is reuse. There are different kinds of simple web services. Pure Content Services provide a programmable access to simple information using parameters upon request. Examples for such services are “Stock Quote” or “Weather” services. Identifiers for an invocation could be a stock identification number for a stock quote service, or a zip-code for a weather service. Simples Trading Services can be used to aggregate information seamlessly out of various resources, which can also be distributed across separate systems to support a requester in making decisions. Information Syndication Services are mostly commercial services which plug in to a value added website to integrate the offerings of this particular website. An example for such service is a web service which integrates the search of used cars according to preferences into a process or workflow [5].

Complex services are composite services. A complex service can be composed of a whole set of elements such as multiple BPEL processes that further orchestrate other services. An example for such a service could be an inventory service that checks the availability of an article in a warehouse and reserves the article upon adding it to the shopping basket. In addition, there are complex services which include interactions with human users. These kind of services expose their functionality in the presentation layer (browser) to the user and require an interaction. A whole workflow of user-interactive services implements a “multi-step web service application behavior” [5].

Complex services may possess complex operations that furthermore implement subprocesses and therefore take longer than simple services to respond. To not block the entire application, complex web services are mostly invoked asynchronously. Typically, complex services are more loosely coupled than simple processes; they implement a document-based and routed message style, underlying platforms can be diverse. The binding protocol uses dynamic late binding. The target of complex web services and business processes is to achieve a broad applicability [5].

2.4.2 Web Service Standards Stack
Web services use open standards such as SOAP and WSDL. Both were mentioned before, as SOAP is the major protocol for implementing the messaging in web service based SOAs. WSDL is the description language to define web service interfaces [12]. Open standards are developed and maintained by standardization organizations. The most important three organizations are the W3C [13] (World Wider Web Consortium), OASIS [14] (Organization for the Advancement of Structured Information Standards), and the IETF [15] (Internet Engineering Task Force). All three are further introduced in the following paragraphs:

The W3C’s mission statement is “To lead the Web to its full potential” [16]. Its goal is to develop open standards and guidelines in order to enable a web for all humans without constraints, a web on different devices, a web of linked data and services, and a web of trust. The W3C works on standards with a fulltime workforce and the public, as an international consortium. Its most prominent employee is Tim Berners-Lee, who is known as the inventor of HTML and the World Wide Web. The W3C specifies im-
portant standards like *HTML*, *OWL*, *XML* and further *XML* based standards, such as *SOAP*, *WSDL* etc. [13].

OASIS present themselves as a non-profit consortium with over 5000 members coming from over 600 organizations in over 65 counties. The scope of OASIS encompasses, according to their own presentation, the Internet of Things, Cloud Computing, Content Technologies, Emergency Management and other areas. The goal is to promote and spread open standards, to bring together developers, and to achieve a consensus in the industry. The consortium is led by a board of directors which is democratically elected every 2 years. OASIS provides the SOA reference architecture [17], and is responsible for important web service standards such as WS-BPEL, UDDI and various other web service (WS-*) standards. The WS-I (Web Services Interoperability Organization) is now a part of OASIS [18]. The WS-I will be introduced later [14].

The IETFs concern is the development and evolution of the Internet and its architecture. The IETF is a community which is open to anyone interested. It mostly consists of “network designers, researchers and operators” who work together in different areas such as routing, transport, or security - and are further organized in working groups. The work is usually communicated via mailing lists, meetings are usually held every 4 month. IETF is responsible for important fundamental protocols such as TCP, IP, UDP and HTTP [15].

Web service and related standards can be summarized and illustrated as an extended web service stack. The figure below shows such an illustration, however it neither contains all possible web service standards available, nor is such a standardized an illustration available. A more detailed illustration of web service standards can be viewed in a poster by the InnoQ GmbH [19]. Even that poster, however, does not show all web service standards, since there are more than 100 web service standards available for many cases.

The illustration below (Figure 2.4) shows a selection of important standards for the realization of a SOA. The standards are classified into classes to inform about their purpose. Classes are further grouped into layers that build on one another: The Transport layer contains protocols that are used for the transport of messages, such as HTTP or SMTP. The protocols may further use the TCP or UDP transport protocols. Messages and interface definitions, as well as further extensions, are most commonly all based on XML. Therefore the next layer contains languages that are closely related to XML, like XPath or XSLT. The classes Messaging, Description, Discovery and Orchestration contain standards that are, at least in this illustration, also based on XML. Finally, the top layer contains additional standards that extend the protocols in the layers below. They implement important goals like web service security, reliable messaging etc.

---

1 This is not the OSI model transport layer.
A selection of web service standards will be introduced further, as understanding them is necessary for understanding the content of the following chapters. The following paragraphs briefly introduce languages and standards that are relevant for this thesis. To work with web services, it is advisable to learn and understand XML and XML-based languages. However, a comprehensive tutorial of standards cannot be part of this thesis. The reader is hereby referred to the relevant literature and specifications.

XML

As mentioned above, almost all web service standards are based on the Extensible Markup Language (XML). Therefore a basic understanding of XML is crucial for working with web services. XML is an SGML-based language for creating documents of XML encoded data in a human-legible and machine-readable way [20]. XML documents shall be easily applicable in order to support a diverse variety of network-based applications, such as heterogeneous web services. XML annotates data in so-called Markups that allow the data to be logically structured. The markups embrace data in the form of start and end nodes, known as <tags>, and also provide further elements to, for example, define document declarations or comment data. XML documents can be read and parsed by applications using a software module, the XML processor [21].

XML is the common in- and output format of SOAP messages. The following figure 2.5 shows a self-defined sample XML file.

```
<?xml version="1.0"?>
<Person>
  <Name>
    <FirstName>Adam</FirstName>
    <LastName>Smith</LastName>
  </Name>
  <Address>
    <StreetName>High Street</StreetName>
    [...]
  </Address>
</Person>
```

**XML-Schema** is used to specify and describe a class of XML documents. The XML-Schema Definition (XSD) language [22] is used to define XSD-files that can be
imported into any XML document. XSD describes the meaning of markups or tags described in the previous section about XML. The language provides a way to define the schema of an XML document, including its elements and data types. Elements and types can be complex (composed of further elements) or simple (only atomic). XSD furthermore allows the definition of constraints that delimit or extend the content of an XML document. These restrictions can be used to validate the content and structure of XML documents. However, the XML-Schema-based validation is, according to the W3C, not sufficient for all scenarios.

XML-Schema is very important for working with web services, as it is the foundation to specify in- and output parameters in the type definitions for the messages defined in the WSDL documents. WSDL and SOAP will be introduced later. The following figure 2.6 shows a simple XSD file. The content of this file describes a part of the XML file above.

![Figure 2.6: Example XML-Schema File](image)

There are further languages that are closely related to XML and commonly employed when working with XML documents. These languages can address, query, link or transform XML documents by using their own syntax. The following table 2.1 shows a selection of some of those XML languages. XML itself and XML-Schema are also included.

<table>
<thead>
<tr>
<th>XML „Language“</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML</td>
<td>Meta language to describe data</td>
</tr>
<tr>
<td>XML-Schema</td>
<td>Definition of XML structures</td>
</tr>
<tr>
<td>XSLT</td>
<td>Transformation of XML-Documents</td>
</tr>
<tr>
<td>XPath</td>
<td>Addressing within an XML tree</td>
</tr>
<tr>
<td>XPointer, XLink, XInclude</td>
<td>Linking of XML-Resources</td>
</tr>
<tr>
<td>XQuery</td>
<td>Selection of data within a XML-data set</td>
</tr>
<tr>
<td>XUpdate</td>
<td>Data manipulation within a XML-data set</td>
</tr>
<tr>
<td>XML Signature, XML-Encryption</td>
<td>Signature and Encryption</td>
</tr>
</tbody>
</table>

Table 2.1: XML Languages

Important XML-based languages that are used in the examination are XPath and XSLT. Furthermore XML Signature and XML-Encryption are important standards for web service security.
**XPath** is always used within the context of a host language, mainly to address and select parts within an XML document tree. It can further be used to conduct simple data manipulations. XPath uses an abstract syntax that is not comparable to the XML syntax. It addresses the nodes in the tree structure of an XML document. These can be element-, attribute- or text-nodes. XPath is important in the context of this work, as it is used in various higher level languages like in the Business Process Execution Language (BPEL) or in XSLT. The standard’s specification can be accessed in the recommendations on the W3C’s website [23].

Figure 2.7 shows an example of XPath in the context of WS-BPEL. The code is an excerpt of the copy procedure within an ‘Assign’ activity. The XPath is marked bold.

```xml
<from variable=”inputVariable” part=”payload”
    query=”/ns1:simpleStockInformationRequest/ns1:symbol”/>
<to […]/>
```

**XSLT** allows the transformation and mapping of XML data. It is used within the transformation element in Oracle’s JDeveloper within BPEL. The language XSLT is used to transform XML documents by transforming XSL stylesheets. The latter can be applied to XML documents in order to alter the structure of XML trees. XSL moreover allows the extension of XML with style information, to, for example, show XML formatted data in a web browser [24].

An XSLT transformation essentially transforms one XML document into another XML document. As the payload of SOAP messages is usually an XML document, XSLT can be used to map different data in- and outputs and alter that document.

XSLT and XSL comprise an XML vocabulary to perform transformations. During a transformation, a set of transformation rules is applied on an XML source tree. The outcome is a result tree that can be totally different from the source tree. The XML structure trees are addressed using XPath. An XSLT uses patterns and templates to match and transform the data. XSLT is processed by an XSLT processor. The basic application of XSLT is illustrated in Figure 2.8.

**WSDL**

The Web Service Description Language (WSDL) is an XML format that provides grammar to describe a web service interface. A web service is defined as a collection of network endpoints that communicate via messages. A web service interface consists of an abstract part, which contains the message, and type definitions as well as the definitions of port types. The abstract part of a WSDL document is reusable. A concrete part specifies the actual binding of port types to network protocols, such as the SOAP proto-
The parts of a WSDL document are described in more detail in the following paragraph [26].

The interface contains a <type> tag, which acts as a container to define elements and data types. The elements and data types specify the in- and output parameters that can be used in the messages. Elements and types can be simple or complex, furthermore schemas can be restricted or extended. The definition of types is commonly achieved by XSD. The XSD code can be included into the type tag or imported using an external XSD file.

The <message> part is the next building block of a WSDL document. It comprises the definition of in- and output messages using the previously defined schema elements in the type definition. Alternatively, messages can also use simple type definitions. The messages are then used to specify operations within the Port Type definition.

The Port Type definition abstractly specifies the operations a service endpoint provides. The <portType> tag includes one to multiple operation tags that further include input and output messages. The output messages are referenced by the messages definitions in the <message> part.

To bind the abstract interface to an actual endpoint and protocol, the concrete parts of the WSDL document have to be defined. The <binding> tag binds an abstract Port Type to a network protocol, such as the SOAP protocol. The communication style which the protocol performs in order to communicate with other services can be selected. There is a document or a procedure style.

Finally the <service> tag binds the WSDL file to a collection of related endpoints that are deployed on a back-end server.

WSDL is described in more detail in the WSDL specification on the W3C website [26]. Furthermore practical examples of WSDL will be created and used in the practical examination in chapter 4.

**SOAP**

The SOAP protocol is a commonly used protocol for communication between web services. It provides a framework for the transmission of structured data such as XML. SOAP usually relies on the HTTP protocol and TCP, but since it is protocol neutral it can also rely on other protocols such as SMTP or FTP using UDP etc. The root element of a SOAP message is its envelope element, which further comprises a header and a body element. The header element of SOAP is optional and can be extended individually. Extension can, for example, serve purposes like the addressing of messages or the provision of information security. The body element comprises the actual message payload that has to be transmitted. Furthermore SOAP is capable of sending SOAP Faults in order to provide information about errors during the processing of a SOAP message. The envelope and body elements are mandatory parts of a SOAP message, the header and fault elements are optional. An overview about the recommendations of the W3C regarding SOAP can be accessed at the W3C’s website [27]. An example SOAP message is illustrated in the following figure 2.9.
SOAP and WSDL based web services on their own do not, however, comprise important functionalities, such as the direct addressing of SOAP messages, the insurance of a reliable message delivery or the maintenance of information security. The top layer of the extended web service stack shown above contained several important extensions that are able to implement these requirements. The standards are referred to as WS-* standards. WS-Addressing [29] allows the addressing of SOAP messages in the SOAP header, WS-Reliable Messaging [30] facilitates a reliable message delivery and WS-Coordination [31] is a standard that further comprises WS-Atomic-TX [32] and WS-BusinessActivity [33] to perform consistent transactions. The standards are however not significant for this examination and will therefore not be further described. WS-Security standards will be introduced in chapter 5.

As mentioned above, the SOAP protocol contains an extendable header element that can be complemented by additional functionalities and standards. Developers are free to develop extensions and consequently more than 100 partly vendor-specific specifications of web service standards are available. This enormous diversity of available standards leads inevitably to interoperability problems, since different standards are commonly expected to require that a counterparty “understands” the particular standard in order to process it [34].

The Web Services Interoperability Organization (WS-I) was founded in order to deal with interoperability issues of web services. The organization created profiles to limit the scope of web service standards to a manageable number and created a set of best practices developed by the organization [18]. According to the WS-I, the best practices are still advanced in order to maintain web service interoperability across distinct platforms, operating systems and programming languages. The profiles are guidelines for Web service developers to create web services that conform to WS-I. They can be downloaded on the WS-I website. In addition, sample applications and testing tool are provided. The latter can be used to test the conformance of a SOAP messages to the WS-I guidelines.

Finalized profiles, according to the WS-I website, are the Basic Profile 1.0 and 1.1, the Attachments Profile 1.0, the Simple SOAP Binding Profile, the Basic Security Profile 1.0 and the Reliable Secure Profile. The most important profiles are the Basic Profile and the Basic security profile; further profiles complement these profiles. In addition Errata files are available that provide actualizations and corrections to existing profiles.

As the WS-I is a large organization that examined the interoperability of web services according to web service standards and their application for several years, the challenges during service interchange that concern WS-* standards cannot be a part of
this examination. The reader is hereby referred to the profiles and to further materials of the WS-I [18].

In some scenarios even the WS-I cannot totally guarantee web service interoperability. The WS-Policy [35] standard can be employed as an additional way to facilitate web services interoperability. The WS-* standard provides a framework to define policy alternatives and assertions that state requirements, capabilities and further behavioral characteristics of a service. A policy can be used to describe a web service’s employed standards, and also to allow the definition of requirements to a service. Consumer and service provider can then negotiate a valid policy for the communication. Policy assertions can also comprise requirements to non-functional aspects such as characteristics of the Quality of Service.

2.4.3 WS-BPEL Basics

According to Alves et al, “WS-BPEL defines a model and a grammar for describing the behavior of a business process based on interactions between the process and its partners. The interaction with each partner occurs through web service interfaces, and the structure of the relationship at the interface level is encapsulated in what is called a partnerLink.” [36]

BPEL allows the modeling and execution of business processes that integrate web services using a standard process integration model. The BPEL standard is based on XML and supports WSDL. It employs further technologies like XML-Schema, XPath, and XSLT, of which the latter two are also used for data manipulations [36].

A BPEL process can be modeled as a synchronous or asynchronous BPEL process. A synchronous BPEL process uses a ‘Receive’ activity to receive input data and returns the output data by using a ‘Reply’ activity. The caller waits until the reply is received. If the process is asynchronous, the calling entity is explicitly called back using a ‘callbackClient’ activity\(^2\). This is necessary as the calling entity continues its activities and does not wait for the invoked instance [36].

In order to realize a state, variables have to be defined within the scope of the process, to save parameter values. The variables can be defined using message types from the WSDL file or XML-Schema data types. The definition of variables is shown below in Figure 2.10.

```xml
<variables>
  <variable name="inputVariable" messageType="ns1:nameOfMessageType"/>
  <variable name="outputVariable" messageType="ns1:nameOfMessageType"/>
  <variable name="simpleString" type="xsd:string"/>
</variables>
```

Figure 2.10: Variables in BPEL

BPEL uses WSDL in connection with Partner Links to integrate web service endpoints. The Partner Links are invoked using the ‘Invoke’ activity. The following Figure 2.11 shows an example Invoke activity in BPEL.

\(^2\) In Oracle JDeveloper
The Invoke activity contains the specified in- and output variables for the invoked service, the name of the invoked Partner Link as well as the Port Type and operation names that are defined in the service’s WSDL file.

Usually an Assign activity is used to copy data from one service endpoint to another. The activity consists of a copy element that specifies a source and a target variable to copy values. The from and to elements reference the variable name, which in this case is a message type variable, the part name of the message type, and an XPath address to query the actual parameter. Figure 2.12 shows an Assign activity.

An alternative to the Assign activity that is used, for example in Oracle’s JDeveloper, is the Transformation element. It is not part of the official BPEL specification but allows the integration of XSL files into a BPEL process to perform XSLT transformations on the data. BPEL also allows the implementation of structure activities, such as the implementation of loops, switch cases, if conditions or scopes. Scopes delimit a process sequence and allow the specification of local variables; they can further be used to implement an exception-handling. BPEL code is processed by a BPEL engine. The specification of WS-BPEL can be accessed at OASIS [36].

### 2.4.4 The Service Component Architecture (in JDeveloper)

The Service Component Architecture (SCA) is a collection of specifications that form a model to implement SOA applications. SCA is built on open standards to enable business functionality to be implemented as a composed set of services. The resulting composite applications can be exposed as services by using Service bindings. These service bindings expose an interface to external users, to allow access to the composite application. Furthermore SCA can also consume external services that are referenced in the Reference bindings. A policy framework allows the definition of constraints, capabilities and Quality of Service expectations [37].

SCA allows the creation and composition of service components and further elements. The elements can be connected using Wires. Examples of service elements in JDeveloper include BPEL processes to orchestrate internal and external references, Human Workflow elements to allow a human interaction within a process, Business Rule elements to define rule-based decision logic, Mediator elements to route and transform messages and Adapters to include external technologies and protocols, such as databases, files, http messages etc.

The SCA was used as the central environment to create SOA composite applications for this thesis. Oracle JDeveloper in particular was used as a part of the Oracle SOA
Suite 11g to create SOA composite applications in practice. The use of the software is further documented in the practical examination, which is part of chapter 4.

The following illustration (Figure 2.13) shows Oracle’s Fusion Order Demo application that can be downloaded at the Oracle website\(^3\).

![Figure 2.13: Fusion Order Demo application SCA in Oracle JDeveloper; Source: [38]](image)

The figure shows the graphical design view of the SCA that is part of the composite.xml artifact. The left swim lane represent the service bindings, which in this case contain the exposed interface of the central OrderProcessor BPEL process. The interface allows external access to the composite application. The opposite swim lane to the right shows external references, which in this graphical excerpt are references to the supplier-, to a notification-, to a shipping- and to a batch processor- service. On the canvas in the center the main OrderProcessor BPEL process is wired to all further elements, such as the BPEL process InternalWarehouse. The canvas furthermore includes Mediator elements that allow the routing and transformation of messages. A Human Workflow element, which in this case enables the human interaction with the OrderProcessing BPEL process, is also present. The documentation of JDeveloper can be accessed at the Oracle website [39].

### 2.5 RESTful Web Services

REST (Representational State Transfer) does not specify a web service standard like SOAP or WSDL, but is a set of design guidelines. There is no such thing as a REST architecture. RESTful web services conform more or less to the specified criteria of REST itself. Their conformity can be measured in the REST maturity model [40].

Essentially a RESTful web service needs to implement four concepts and four properties. A service has to define Resources that reference Representations, using URIs.

The Representations have to be further linked to each other. Services and Resources have to be addressable, stateless, connected and need to employ a uniform interface [41].

In contrast to SOAP and WSDL-based web services, RESTful services do not provide a single endpoint, but a number of Resources that can be addressed and accessed using URIs. URIs should only use logical addresses and be descriptive and structured⁴. RESTful services commonly use the HTTP protocol and its request/response messages. A “standard”⁵ RESTful service can use all available methods of the HTTP protocol. The most important HTTP methods for REST are HTTP GET, HTTP PUT, HTTP POST, and HTTP DELETE. Further methods are, amongst others, HTTP HEADER, and HTTP OPTIONS - both provide metadata about a resource. The use of HTTP fulfills the requirement to employ a uniform interface. The HTTP protocol can be secured using HTTPS (HTTP + SSL). The GET method is considered secure as it does not alter a resource. The methods GET, PUT and DELETE are idempotent, which means that they can be sent more than once without affecting the result [41].

A resource references a subset of data, which can be sent within a message as a Representation. The Representation is a copy of the data and can be compared to the payload of a SOAP message. It can be provided in different formats, such as XML, JSON, and CSV etc. The most commonly used format of RESTful services is JSON (JavaScript Object Notation), according to the web service market research conducted within the scope of this thesis [41].

Representations can be requested from a server using HTTP GET in connection with the referencing URI of the Resource. A request message must contain all required information for the service to deliver the requested data, since in REST a service must operate stateless. A Representation can also be sent to a server, for example to create a new Resource or update an existing one. HTTP PUT in connection with a new URI or HTTP POST on an existing URI can be employed for this purpose. HTTP PUT also allows the modification of existing resources. Finally HTTP DELETE allows the deletion of a resource. Representations should be linked to each other to fulfill the requirement for connectedness [41].

RESTful web services are mainly introduced in this chapter to delimit them from SOAP and WSDL based web services. RESTful services are not BPEL-compliant without further adjustments [42] and their integration into the practical examination would be beyond the scope of this paper. Some integration approaches will however be briefly discussed in chapter 5. The services found during the web service market research where more suitable for web applications than for complex business processes.

2.6 Current State of Research and Research Context

The main goal of this academic paper is to create an overview of the challenges related to the interchange of web services in a cross-cloud SOA environment. Furthermore, solution approaches to identified challenges will be given. Service interchange relates to many subjects within the domain of Service Oriented Computing, as well as related domains such as Information Security, Enterprise Application Integration, Semantic web etc. Consequently, the work relies much related literature. As a research context, Kirchner et al. 2011 is named [1].

---

⁴ E.g. http://example.com/wiki/Person/Adam to reference a person called Adam as a resource.
⁵ There are extensions to http that provide additional methods
In order to identify challenges and issues related to the subject, and furthermore give possible approaches for solutions, the subjects of SOA and web services, in particular, required comprehensive examination. The subject of SOA and web services is complex but also mature, and much useful content can be accessed in fundamental textbooks about SOA, Cloud Computing and web services. Notable fundamental literature used in this thesis was written by Thomas Erl [2], Michael Papazoglou [5], Nicolai M. Josuttis [12], Liang-Jie Zhang et al. [4], Eric Newcomer et al. [6], Jared Carstensen et al. [10], Laurent Menard [43], and Leonard Richardson et al. [41]. This list of authors is not exhaustive; there is far more literature available about the fundamentals.

All these reference texts, however, do not contain hands-on approaches on how to practically implement a cross-cloud SOA. Furthermore, none of these books provided sufficient information about interoperability challenges in connection with web service interchange; their web service interoperability sections were also not sufficient. Consequently, a practical examination of integration challenges had to be conducted. Because such an examination requires appropriate knowledge, further practically-oriented books and the oracle SOA Suite documentation [39] had to be viewed. Identified books were, among others, “Oracle SOA BPEL process manager 11gR1 – A hands-on tutorial” [44] and “Oracle SOA suite 11g R1 developer's guide - Develop service-oriented architecture solutions with the Oracle SOA Suite” [45]. Of those the Developers Guide proved the most useful, since it provided a straight-forward tutorial on how to work with JDeveloper in Oracle SOA Suite 11g.

During the practical examination it became clear that the subjects of information security, and trust, especially in a cross-cloud SOA environment, are particularly important. To get a basic understanding of the subject of IT-Security it is advisable to read fundamental textbooks about that subject. Because SOA security requires a set of additional measures to maintain security, apart from ordinary IT security, it was important to read additional books about SOA security, such as that written by Kanneganti and Chodavarapu [34]. Further information about SOA security could be accessed in academic journals, such as Geuer-Pollmann [46]. A good introduction of the subject of Trust, including various example approaches, could be accessed in Bouguettaya’s Book Advanced web services [47]. The book also provides references to several related texts, which approach the subject of Trust Management.

The subject of service interchange is, especially in advanced projects, related to the subject of Semantic web and semantic web services to facilitate the dynamic discovery and integration of web services. There are books available to develop an understanding of the general subject of the semantic web, such as [48], as well as more specific books about semantic web services [49]. To understand how such methods could be applied in a practical context, many academic journals about the subject are also available.

There are also some research projects which follow the sophisticated goal of integrating further methods to facilitate the automatic and dynamic discovery, integration, composition, and execution of web services, in order to automatically create complex processes that support business requirements. In this field further academic journals, such as García et al. [50], or Paganelli et al. [51], are also available.
3 Conceptual Approach

Within the framework of the last chapter, the fundamentals of relevant technologies were introduced. Furthermore a literature review was performed in order to identify the relevant technologies. The goal of this thesis is to identify challenges of service interchange in a cross-cloud SOA environment. The identification process shall encompass a practical examination, therefore a conceptual approach to this practical examination had to be specified.

To generate an approach, expected challenges were identified in connection with the literature review. The expectations were generalized and formulated in hypotheses. These hypotheses represent the core guidelines of this examination. The research, however, made clear that not all of those hypotheses formulated could be examined in practice. This was because an examination would either lie beyond the scope of this thesis, or because those aspects had already been examined with considerable effort. Consequently the practical examination is limited to the examination of integration challenges. Further challenges and approaches to solutions are discussed in chapter 5.

To determine the availability of functionally comparable and interchangeable services, a web service market investigation was carried out. The market research was a major aid in decision making for choosing a business domain that provided a feasible testing context. The research was conducted to gain a general overview of the service market. This overview served as the basis for the definition of further requirements for services and a testing process. Following this it also became the basis for further research to identify applicable web services.

At the same time, a testing approach was specified. The first step was the definition of testing scenarios to enable the integration of the predefined hypotheses into the planning of the practical examination. The second step required the definition of further requirements concerning the testing environment. This technical testing environment was declared in order for others to replicate the experiments.

3.1 Definition of Hypotheses

The following hypotheses (shown in Table 3.1) display the insights derived from the theoretical research about services and interchangeability. The hypotheses link the preliminary theoretical papers with the practical experiments, and reflect the challenges and issues expected during the implementation of the practical examination and during service interchange in the real world. The hypotheses are also part of the foundation and guideline for the definition of the actual testing scenarios, which are defined later in this chapter. The hypotheses will be evaluated in the end of the fifth chapter.

<table>
<thead>
<tr>
<th>Hypothesis 1 (Equal Interface)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hypothesis 1.1 (Technical View):</strong> Different implementations of the same service interface are always technically interchangeable if they apply the same standards.</td>
</tr>
<tr>
<td><strong>Motivation:</strong> Web service interfaces can be implemented by different service providers. The implementation is a black box for the service consumers. Meanwhile the service interface remains untouched and therefore can be integrated and exchanged technically, if the same additional standards are adopted.</td>
</tr>
<tr>
<td><strong>Hypothesis 1.2 (Service Functionality):</strong> Different implementations of the same service interface cannot always be interchanged functionally. Services which implement the same interface can implement a different functionality and behavior. Services are then not interchangeable.</td>
</tr>
</tbody>
</table>
Motivation: If a service implements an interface, it cannot be granted that it implements the same functionality like another service implementation of the same interface. In addition the behavior of the service might differ. The interface description file only describes the technical links and conditions of a service but no functional characteristics.

**Hypothesis 1.3 (Data Model):** Services which implement the same interface and provide the same functionality can differ according to their data back-end. This includes the encoding of data, calling conditions and output data. Integration challenges will occur.

Motivation: A service might access a different database back-end. This may lead to the possibility that the back-end requires input data in a different format, or encoding, in order to be invoked. It furthermore might also generate output data that is formatted or encoded differently. Adjustments will have to be made.

**Hypothesis 2 (Different Interface Technically):** Functionally equivalent services that employ different service interfaces need to be technically adjusted in order to be interchangeable. Examples of these differences include disparate parameters, data types or solution approaches.

Motivation: Services that employ different service interfaces can be functionally equivalent. Disparate interfaces can use varying approaches to implement a functionality, for example by using different complex message types to deliver messages. Further technical differences include the data types of parameters. Services that employ different interfaces also include all the issues that come with a disparate functionality and with different data models.

**Hypotheses 3 (Composite Applications):** Composed Services need to provide a steady interface and steady data to the outside world.

Motivation: In order to avoid having the service consumers make adjustments, every time a differing service is exchanged within a composed service, the interface and data output of a composed service needs to be steady.

**Hypothesis 4 (WS-* Standards):** Web services have to fulfill the same standards in order to be interoperable.

Motivation: Standards are meant to enable interoperability by definition. Web services use the open Internet standards in order to fulfill interoperability.

**Hypothesis 5 (Semantics):** The pure functional content and logic of a web service cannot be identified automatically by a machine without semantic annotations. Services have to be searched individually by hand to guarantee interoperability.

Motivation: A web service interface cannot give sufficient information about the functional logic of a web service. Therefore semantic annotations will have to be included to enable a machine readable discovery of functionally interoperable web services. WSDL files describe all technical linking points and variables, but this is not enough to determine the functional content and comparability of two web services.

**Hypothesis 6: (Security and Trust)**

**Hypothesis 6.1:** The service interchange within a cross-cloud SOA environment leads to additional security challenges.

Motivation: The implementation of many distributed, externally provided services into a local BPEL process leads to further security issues concerning the goals of information security. Furthermore external providers need to be trusted with potentially
Hypothesis 6.2: It cannot be guaranteed that a service provider implements the same security measures and standards within its compositions.

Motivation: An external service provider provides a service endpoint to a consumer. The consumer can only view external characteristics or policies of a service, but not the internal elements of the “black box”. Consequently it is not evident which external elements the service provider uses himself and which security standards are applied internally.

Hypothesis 7 (SOAP and REST): SOAP based web services and RESTful web services are not combinable without special measures in a BPEL process.

Motivation: REST is not based on actions as SOAP/WSDL-based web services are, but is based instead on representations. REST can consequently not be orchestrated in a BPEL process without further adjustments, since it does not provide invokable operations.

Table 3.1: Definition of Hypotheses

3.2 Web Service Market Research Results

An overall goal of this investigation is to give solution approaches to facilitate the service interchange in a business scenario, where business processes can be implemented individually with the best-fit services available. To achieve this task SOAP based web services will be used as the main building blocks, as they use operation, are activity based and closely related to the implementation of complex business processes. In the future a comprehensive Internet/Cloud based web service market is expected, where many service providers offer functionally equal services [1]. To portray today’s situation, a web service market investigation was conducted. This research was also required to create a foundation for the design of a feasible testing scenario and to identify applicable services for the actual service interchange.

The research started with the use of meta-search engines to identify web service repositories and registries that contained SOAP web services. It was observed that many major web-based companies had shut down their open service registries years ago [52]. UDDI registries especially did not exist anymore. Further registries were unmaintained. The research also showed that most available services in the consumer service domain were based on RESTful services. Amidst these surprising findings some service registries providing useful SOAP services could also be identified. The most useful registry for the purpose of this work turned out to be the web portal www.programmableweb.com.

The second goal of the market research was to find a number of functionally comparable SOAP services that were not too complex for the purpose of the examination. These functionally comparable services had to differ in various terms such as the interfaces used etc. This meant the services had to exist as implementations of various different providers. For this purpose, on the Web portal, suitable business categories were searched for suitable web services.

The most readily available services could be identified in the following business domains:

- Advertising, especially Online Marketing services and Ad Management services.

---

- E-Mail and Messaging Services, including E-mail/SMS Marketing, Notification, and Newsletter services.
- Financial Services, such as Stock Quote, Option Price, Commodity Price Services etc. as well as services that facilitate the trading of financial securities.
- Validation services for e-mail addresses, telephone, payment data, etc.
- Security services such as fraud detection services.

Every service provider website was analyzed to gather information about the availability and testability of available services. In addition to this service documentations, if provided, were also viewed to estimate the complexity and functional comparability of services. Unfortunately most provider websites were structured in a manner that was confusingly complex and required registration in order to access service documentations. Further information, such as pricing and the availability of trial memberships, was hard to find.

The further market research was limited to the domains of Data Quality/Validation services and Financial Information services. The marketing services in particular turned out to be too complex for the examination’s purposes. Moreover the services of other domains were also not available in sufficient numbers. Notable and testable service providers were “Xignite” in the financial services domain and “StrikeIron” providing data quality services. The interfaces of the testable services were invoked and tested in order to collect information about the data in- and output parameters. Finally, stock quote services were chosen as the most appropriate candidates for the examination. They provided more convenient output data than validation services and were also more freely available. The service logic of a stock quote service is more transparent and straightforward to understand; the common ingredients of a stock quote service can be derived from other sources such as financial web portals.

However, even the stock quote services were not available in all required variations to assess all possible hypotheses or scenarios. It was not possible to find services of different service providers that implemented the same interface. As a result of the limited pool of freely available or testable SOAP services, the original approach, to create one rather complex business process to examine all scenarios was abandoned. Instead the definition of various different scenarios was preferred. It was also decided to conduct the experiments within a simpler BPEL process to examine the predefined hypotheses. In order to examine all possible scenarios, especially those related to the service interface, it was decided to define “semi-mockup”-services that imitate the stock quote logic but do not implement a real stock quote service.

### 3.3 Definition of the Testing Scenarios

Several testing scenarios needed to be specified in order to enable the practical examination of the previously defined hypotheses. The scenarios should be examined within a composite application that invokes external services in a BPEL process. Within the process one service will be exchanged at once.

A scenario covers the interchange of a services operation in a BPEL process. Such an operation possesses certain characteristics like a different interface or a special grade of functional relation to another substituting service. These properties are described in more detail in the following scenario declarations.

As described in the hypotheses definition, services can differ in terms of their functionality. The functionality of a service can differ substantially, e.g. if a service implements the logic for a task of a different business domain. A service can be functionally different but related, for example if a service provides financial data for options instead
of stocks. In this case both services are in the financial services domain and their process-logic may also be similar. If two services implement basically the same functionality, such as to provide stock information data, they can still be functionally unequal. They can provide additional or different operations, use disparate identifiers for invocation or can provide distinct output parameters. Even if two services are totally equal, they might use different service interfaces, which also leads to interoperability issues. In order to systematically examine these integration issues scenarios are defined. The scenarios are related to the hypotheses. Further hypotheses such as WS-* standards and security aspects are not part of the scenarios, since they will be discussed in a different approach.

**Scenario 1:** A predefined service interface gets implemented by various different service providers. The service providers supply a copy of that interface definition as a WSDL file to their customers. The interface is identical to the other provider’s interfaces. The back-end however can differ slightly from the solutions of the competitors. This situation can occur in a scenario, where a company provides an interface to different service providers for implementation. A future scenario could also be the standardization of interfaces.

The scenario examines the effects of differences in interfaces on exchangeability. The scenario furthermore enables the testing of different data encodings/models in and output parameters. According to Hypothesis 1.1 it is expected, that equal services will not result in any interoperability issues, therefore it is expected that the impact of differing data models can be observed in an isolated way.

**Scenario 2:** Different service providers implement the same interface. The services are identical, the back-ends comparable. The interfaces, however, differ slightly, for example because two providers redefine the interfaces themselves by using their own namespace URIs or different identifier names.

This scenario is realistic in an environment where interfaces for certain subjects get standardized. Service providers however use their own names to name identifiers and namespaces within their domain. The scenario examines the impact of names to interoperability.

**Scenario 3:** A common business problem is addressed by various service providers within an open market without any restrictions. It is very likely that different providers implement their own, varying solutions and consequently functionally disparate services for the task. Therefore it is also likely that these providers use different interfaces.

The scenario examines the impact on interoperability created by functionally differing but related services and interfaces. As explained in Hypothesis 2, in this situation further adjustments are expected to maintain interoperability and exchangeability of services.

**Scenario 4:** If a service is situated in the same business domain such as the financial business, the process logic of a service might be somehow related. For example if a service needs an identifier to be invoked and provides information within parameters as simple types, then the differences between two services in some cases, can only be perceived by the semantics of the parameters. This also means that theoretically an interface can be implemented by services that provide a differing functionality.

The scenario examines the Hypothesis 1.2. The scenario is less likely to occur than the other scenarios, but it describes the pure technical aspect of interchangeability in BPEL and XML.

**Scenario 5:** A service is situated within a composite application that, as an API, is exposed to further customers. It should be possible to exchange services in order to optimize the composite services quality of use. The interchange has to follow additional rules, since the service output to the customer must stay the same.
This scenario is very important because it examines the need of a common interface that can be included in further composite applications. The scenario is related to Hypothesis 3.

3.4 Requirements
During the conduction of the web service market research requirements for applicable services were defined. The definition of requirements was influenced by the general market research and itself influenced the discovery of applicable services for the practical examination. The requirements definition was also influenced by the definition of scenarios, which were themselves based on the hypotheses.

Likewise the requirements concerning the testing process, the technical developing environment and the “semi-mockup” services had to be defined. The definitions of all requirements are contained within the following paragraphs.

3.4.1 Requirements for Services
Services need to be functionally related in order to be exchangeable\(^7\). Consequently functionally comparable operations have to be identified. Functionality that is different or not implemented in the substituting service cannot realize the business logic required and is consequently not sufficient to substitute a service. Functional differences between two services are fluid and already begin with one different parameter provided or required in an in- or output message. One must therefore specify which grade of functional difference is still acceptable and what is not.

To examine all scenarios there need to be various kinds of services available: (1) Two functionally related services that possess an equal interface are needed to compare a situation in which an interface is implemented by two different service providers. (2) Two services with logically equal interfaces, that employ different names and namespaces, are needed to analyze a situation where service providers implement an interface but use their own names and namespaces. And (3) finally an external service that employs a different interface, but related functionality, is required to examine a heterogeneous service supply.

The services that should be used to examine the situation in which two interfaces are alike need to be “semi-mockup” services that implement the logic of a stock quote service. The interfaces of these services should be easily editable in order to analyze the impact of differences within an interface, such as isolated testing of names, namespaces etc. Moreover these self-defined services should also provide an editable programming logic so that the effects of a different data output, in terms of the data model, format or encoding, can be analyzed separately.

The requirements for services are listed in Table 3.2.

### Requirements for Services - Overview

- Services candidates need to be functionally related
- The following services are required:
  - Two services with an equal interface to test scenario 1
  - One additional service that employs the same interface, but uses different names and namespaces to test Scenario 2

\(^7\) See Hypothesis 1.2
3.4.2 Requirements for "semi-mockup" Services

A “semi-mockup” services should copy the logic of a real stock quote service and provide stock information in a request-response way. It is sufficient for the examination to create services that return an output message data set after being invoked. It is not intended to build real stock quote services which provide real up to date stock data.

Mockup services should return a set of parameter values when invoked with a stock identification symbol. For testing purposes the services should be able to provide information for four different predefined stocks. The stock identifiers will also conform to the standard stock symbols used in the financial industry. The mockup services should provide two different operations that return different data sets.

The data responded should be editable to simulate a different data model. The service, however, does not need a database back-end, since simple switch case logic is better suited for the requirements to edit the data output.

A subsequent service, which is also a semi-mockup service, simulates the processing of the output data. It furthermore implements a second operation which requires an identifier provided by the previously invoked stock quote service.

The requirements for “semi-mockup” services are listed in Table 3.3.

### Requirements for “semi-mockup” Services - Overview

- Implement stock quote logic without being a real stock quote service
- Return a set of parameter values
- Identifiers and stock values correspond to the real data
- Interface needs to be easy to alter
- Data output should be easy to alter

### Table 3.3: Requirements for “semi-mockup Services

3.4.3 Requirements for the testing process

The scenarios are examined in a simple process. This process was chosen over a more complex end to end business process, because it more directly enables the conduction of practical testing. Furthermore the low availability of freely available and testable SOAP services limited the possibilities to define adequate business processes. Alternatives lead to the need to implement very complex service composites. The latter however do not improve the testing outcome of the actual service interchange examination as the technical integration pattern stays the same. From the technical perspective, it does generally not matter, whether services are free to use, commercial, or more complex. However the complexity of the integration process, especially concerning the interface- and data-transformation, plus the service authentication might differ. The authentication required using a commercial web services is shown in chapter 4.3.3 SOAP Header Authentication.

---

8 E.g. E-Commerce services require a complex environment
The process needs to be implemented within a SOA composite application. The goal is to examine the service interchange within a cross-cloud SOA environment. A SOA composite application enables the composition of BPEL processes, the creation of composed services and the provision of the service to external consumers. Additional elements can be implemented in a SCA, including adapters, mediators, business rules etc. [45].

The service should be implemented in BPEL as it allows a straight-forward orchestration of web services. It is also a common standard to orchestrate SOAP web services. BPEL provides elements to, for example, create variables, copy parameter content from one service to another and allow the inclusion of XSL transformation files. A further benefit of this approach is that different kinds of functions can be applied to the data [36].

The process should contain at least two different services that provide at least two operations. Two services are the minimum constellation in a process to examine the scenarios. A services operation gets invoked and returns a set of data. At least a part of the data output is then processed by a subsequent service to perform further activities. A service can contain multiple operations which perform the actual process activities. If a service contains two operations, and both of these operations are invoked, the process already contains three process steps.

One of those services shall be exchanged in each scenario. To conduct the examination, it is necessary to only interchange one service operation per test. If more service operations are interchanged, the effects of one particular interchange cannot be shown clearly.

The services should be invoked in a synchronous process, to get an immediate response from the service. Stock Quote services follow a simple request-response message exchange pattern; therefore they are well suited for this purpose.

Table 3.4 gives an overview about the requirements for the testing process.

- Requirements for the testing Process - Overview
  - Implementation and examination in a simple process
  - Realization within a SOA composite application using BPEL
  - At least two services within the process
  - Services shall provide two operations each
  - One exchange per testing scenario
  - Services shall be invoked in a synchronous process

### 3.4.4 Further Requirements

The composite application shall use a universal interface in order to assign and check every input and output of every possible parameter set.

The developing environment needs to be freely available. It should provide an SOA infrastructure with an application server and integrated development environment software that supports the developing process. The developing environment should also be easy to install and run.

### 3.5 The Implementation Context

The following section describes the technical infrastructure required to replicate the experiment. Oracle’s SOA Suite 11g was chosen as the developing environment to per-
form the practical examination. The software suite is available for download as a pre-configured java virtual machine, or as a VM on the basis of Linux. The Linux VM was downloaded and executed in Oracle’s Virtual Box 4.3.12. The Virtual Box was updated on a regular basis, so the version number changed over time.

Oracle SOA Suite 11g provides a comprehensive environment and can be downloaded and installed free of charge\(^9\). The software suite provides a “Weblogic” server as its application server and a BAM (Business-Activity-Monitoring) server. The SOA suite employs JDeveloper 11.1.1.6.0 as its integrated development environment (IDE) as well as further web-based management dashboards. The Enterprise Manager, the database server and the Oracle Service Bus, which is Oracle’s ESB implementation, can be accessed using the web browser.

### 3.6 The Implementation Approach

After the experimental approach has been outlined, the services and testing applications need to be implemented. The implementation has to be conducted in the Oracle environment described above. Only web services which comply with the defined web service requirements can be used. The web services are orchestrated in a BPEL process within an Oracle JDeveloper SOA composite application.

First the “semi-mockup” services need to be generated, which includes the creation of two services which both employ an identical interface. Thereafter another “semi-mockup” web service, that uses a slightly different interface, must be created. Finally, the consuming application’s universal interface has to be defined.

After the self-defined services are implemented, the external web service and a suitable operation have to be identified. The invocation prerequisites of the external service have to be complied. This includes the registration and request of a user token and the implementation of a convenient user authentication.

After the required services are available for use, a first “initial” implementation of the composite application has to be generated using a SOA application template. This implementation also includes the first implementation of the testing BPEL process, which uses the first stock quote service and links it to the subsequent processing service.

Finally, once the initial implementation is deployed and tested, the exchange process starts. The stock quote service is substituted by a second stock quote service which uses the same interface. The experiments are conducted in the order of the examination scenarios until all scenarios are examined. All implementation steps are documented.

The actual implementation is explained in more detail in the following chapter. The following Figure 3.1, also presents an overview of the course of action. The process is separated into four phases. The first preparation phase has been the content of this chapter. The subsequent phases are covered in the following fourth chapter.

---

An Example Scenario: Online Stock Brokerage Company
To establish a reference to a real world scenario, an example company is introduced in this section. The fictional company will be referenced in the following examinations.

The company is situated in the financial services domain and provides real-time financial news and data to its customers over the Internet. Furthermore it is a brokerage company that facilitates the trading of financial securities.

The company employs external web services to provide news and up-to-date security data on the company website. Furthermore the company has developed its own value-adding service that collects and condenses financial data to perform statistical evaluations. The results, as well as further expert analysts’ recommendations, can be accessed by paying customers.

In order to not rely on few external service providers, and in order to provide services with the best Quality of Service, external services shall be exchangeable within the internal application and process logic. The company itself wants to provide a service API to its customers.
4 The Practical Examination of Service Interchange

The previous chapter defined a conceptual approach for the examination of challenges that are expected during service interchange in a cross-cloud SOA environment. The approach included the definition of hypotheses, requirements, and testing scenarios.

This chapter covers the actual practical examination. It encompasses the definition of the initial testing process, the implementation of mockup services, the discovery and basic explanation of an external service, the creation of a testing application and the actual service interchange. The chapter covers only the practical examination of integration challenges. At the end of this chapter the identified integration challenges will be summarized and categorized. Further challenges that are not integration challenges will be discussed in chapter 5. During the practical examination, the Oracle SOA Suite 11g Documentation [39], and the Oracle SOA Suite 11g R1 Developer’s Guide [45] were used.

4.1 Specification of the Testing Process

To investigate the challenges during the interchange of services within a cross-cloud SOA environment, a practical examination was conducted. To execute this examination in a repeatable and scientific manner, a testing environment had to be established. The technical environment was explained theoretically, including a description of the used developing environment, in the previous chapter. For the actual testing purpose the detailed implementation needs to be defined and a process must be selected as the starting point of the actual service interchange.

As mentioned in the fundamentals chapter, within a SOA, BPEL is used to orchestrate web services. Exchanging a web service within an SOA application, is to exchange one service’s operation within a BPEL orchestration against another services operation. To examine this exchange process, and to define a starting point and environment for the practical service interchange, a simple but sufficient BPEL process within a composite application had to be defined. This process is described as follows:

The testing BPEL process employs a stock quote service, which is invoked using an identifier whose value is entered by the user. It returns stock information as a set of parameters and delivers them to a subsequent service. The subsequent service processes the stock data and hypothetically uses it as a basis for condensation and statistical evaluation. The subsequent service also implements another operation that provides the actual statistical data. It gets invoked by another identifier and returns a set of further financial data concerning the stock. The stock quote service shall be exchanged through other comparable services available.

The process was designed to simulate the impact of an interchanged service on subsequent services that require input data as a set or as a single identifier. The impact should be simulated within a single BPEL service. The process is kept simple but sufficient to examine the defined testing scenarios. This process was not designed for implementation in a real situation. To retain an overview the following figure (Figure 4.1) shows an abstract diagram of the BPEL process\(^\text{10}\).

\(^{10}\) The actual implementation of the BPEL process can be viewed in section 4.2.1.
The block arrows indicate the process steps and process flow. The black labeled block arrows can be seen as service invocations invoking different operations. The vertical black arrows indicate the message flow from an invocation activity to an operation and back. The messages are labeled according to their content. The connections between the services and the operations illustrate which operations a service provides.

The payload Parameters indicate the request-response style of this application. The Get Stock Information operation of Stock Quote Service is invoked by an Identifier. The operation returns a set of stock information as a set of parameters. The stock info parameters are copied into the Submit Data operation of the Financial Data Processing Service and status information is returned. After that the Get Statistics operation of the same service is invoked by the identifier that is also part of the Stock Quote Services output. In the end of the process, the output data is returned to the user or another service.

4.2 Implementation and Identification of applicable Services

The following text explains and justifies the creation and choice of web services that were used for the testing purposes. As described in the third chapter, one examination goal is to exchange services that provide the exact same service functionality and interface. Furthermore the examination should also show what happens to the exchangeability if an interface only differs slightly, for example if a service provider uses its own names and namespace definitions.

It is difficult to find freely available and functionally equivalent SOAP web services, and it is even more difficult to discover services which have the exact same interface\(^{11}\). Consequently it is necessary to build self-defined “semi-mockup”-services for this examination. These mockup services follow the rules of real services, but provide a limited amount of functionality and information.

For the sake of this examination four of these services were defined: two services were implemented using indistinguishable interfaces, one service was implemented with a logically identical interface using different names, and finally one service was implemented such that it ‘processes’ the output data of the other services and delivers ad-

\(^{11}\) This statement relates to the Web service market investigation.
ditional statistical stock information for a consumer. All services provide realistic content that is based on the stock information of the “Bloomberg L.P.” web site.\footnote{An example of this data can be viewed on the following URL, which shows the stock information for the Oracle stock: \url{http://www.bloomberg.com/quote/ORCL:US}.}

Finally, to test the services and orchestrate them into a process, another composite application had to be defined. The “ConsumingApplication” serves this purpose and also implements the starting process described above. The composite application uses a generic interface in order to test all kinds of relevant scenarios.

### 4.2.1 Self-defined “semi-mockup” Services

This subchapter contains a brief description of the creation of the self-defined mockup services. A more detailed insight into the creation process can be viewed in the digital appendices. The understanding of the mockup-services’ creation process will help the reader to understand the characteristics of each particular service.

**StockQuoteService1**

The StockQuoteService1 is an ordinary stock quote information service that is based on the stock information provided on Bloomberg.com. Consequently it’s in- and output parameters were defined accordingly. For testing purposes, the service provides two operations. The first operations “getSimpleStockInfo” provides a stock price, the stock identifier and the name of the company. The second operation “getComplexStockInfo” provides more complex information, such as the stock’s ‘day range’ (high and low share values), the traded stock volume, and price changes which indicate the development of a stock’s value with time. In the following description the creation process of this web service is summarized.

To create the service, a new SOA composite application was created in Oracle’s JDeveloper. Within the new application an individual interface had to be defined, which suited the purpose of a stock quote service and was designed according to the model of in- and output data defined earlier. To create the interface, new XML-Schema and WSDL definition files were created automatically. These were then altered accordingly. Figure 4.2 shows the important element definitions of the XML-Schema definition.

```xml
<?xml version="1.0" encoding="utf-8"?>
<schema [...namespaces...]>  
  <element name="simpleStockInformationRequest">  
    <complexType>  
      <sequence>  
        <element name="symbol" type="string"/>  
      </sequence>  
    </complexType>  
  </element>  
  <element name="simpleStockInformationResponse">  
    <complexType>  
      <sequence>  
        <element name="symbol" type="string"/>  
        <element name="name" type="string"/>  
        <element name="last" type="float"/>  
      </sequence>  
    </complexType>  
  </element>  
</schema>
```
This is a simple XML-Schema Definition (XSD) file, which does not use any restrictions or facets. The XML-Schema consists of four complex elements, which were used in four different message types. The two input “request” elements contain just a single identifier, which is called ‘symbol’ for the stock symbol. The output elements consist of a set of various simple elements for every up-to-date element derived from the website. The elements were given data types. Decimal numbers were defined as floats to obtain another data type variance - other external interfaces use double and decimal.\(^\text{13}\) The type int was chosen as a 32bit integer that allows the record of large integral numbers.

In contrast to online available services, JDeveloper creates separate XSD files, which are then imported into the WSDL file within the <types> tags. This is acceptable in this situation, but not optimal when providing the WSDL via the Internet, since the XSD elements and types cannot be viewed by external users that access the WSDL file\(^\text{14}\).

To create the actual interface, the created StockQuoteService.wsdl file was edited according to the defined XML-Schema elements. The following WSDL file excerpt (Figure 4.3) shows the definition of the message and port types that were used.

\(^{13}\) The data type float should not be used for real financial data, since it approximates values instead of saving the real value. During calculations and rounding this lead to inaccurate values being presented [Float].

\(^{14}\) This is important to distinguish differences in the content of messages, i.e. parameters.
The WSDL file defines four message types, which adopt the four complex elements defined earlier in the XSD file. The message types are then used in the port type definitions as in- and output messages to implement the operations. As can be seen, the “StockQuoteServicePort” had to be extended by a second operation “getComplexStockInformation”. After the definition of elements, messages and port types, a (SOAP-) binding and the service tag had to be created. This can be done using a wizard within the WSDLs design view in JDeveloper. Finally, the interface was defined successfully.

For each of the defined operations within the WSDL file a BPEL process had to be created inside the composite application. Both processes were created using the WSDL interface defined above. Within each BPEL process, a simple switch case element was used to individually react to certain requests. An external data source, such as a database, was not implemented since this was not necessary to examine the situation and data formats could also be altered easily. The switch element was extended by further cases to realize requests for the stocks of Google (GOOG), Oracle (ORCL), IBM (IBM), Goldman Sachs Group Inc. (GS) and Others. The condition was e.g. set to:

`bpws:getVariableData('receiveInput_getSimpleStockInformation_InputVariable','payload','/ns1:simpleStockInformationRequest/ns1:symbol')='GOOG'

If a condition was met, a BPEL Assign activity was processed to set the output variables of the service with the particular values of a stock.

A 'Mediator' element was added to the SCA to route the messages from the composite application’s interface to the particular BPEL processes. The external interface was wired to the Mediator and the Mediator was wired to both BPEL
processes. Within the Mediator the routing was defined using XSL maps, which was done using a wizard. Finally, the service was deployed and tested.

**StockQuoteService2 and StockInfo Service**

The service StockQuoteService2 was created manually using the same WSDL and XSD definition as StockQuoteService1. The creation procedure was identical, so it will not be repeated here. The process is identical with the exception of a different <soap:address location=""/> definition, which refers to the actual endpoint of the service on the application server. It has to be set in the <service> tag of the WSDL file after deployment.

The StockInfo service was created accordingly with the exception that the service uses different WSDL and XSD files in order to provide an altered interface. The file structure remains identical to StockQuoteService1 and 2 but uses different element-, variable-, message-, and port-type names as well as its own namespace. The services logic also remains identical.

**FinancialDataProcessing**

The Financial Data Processing Service simulates a service which provides additional financial and statistical data for customers. In the scenario this service is one of the core business activities of the online stock brokerage company. It hypothetically collects stock information, condenses the information and enriches it with further company data. Internally the company conducts sophisticated, close to real-time analysis on actual financial data.

In this examination the service is subsequently connected to the stock quote services and uses their data as its input in the “submitData” operation. Furthermore it provides its additional information to the customers via its output in the operation “getStats”.

The FinancialDataProcessing service was created in almost the same manner as the services mentioned before, consequently these steps are not explained further. The service uses a different interface, which also makes new XSD and WSDL file definitions necessary. The service provides two operations “submitData” and “getStats” whose back-ends are, again, accessed via a Mediator. This Mediator connects the external interface with two separate BPEL processes and implements the routing. The BPEL processes again implement the service logic via “switch cases” that contain Assign-activities. The output differs from the other services output since the service provides a different functionality. Still, simple data is provided using simple types.

The composite.xml SCA contains two BPEL Processes. “ProcessData” processes the data submitted to the operation whereas “submitData” checks if the invocation symbol is valid and returns its status. It simulates a function that validates inputted data. The operation returns a true() or false() variable as Boolean, as well as a related status name variable providing either the strings 'Status: OK' or 'Status: Error'. The BPEL Process 'StatisticsProvider' returns additional stock information data for a particular stock symbol. The data is again based on Bloomberg stock data that can be accessed via the Bloomberg website[^15] in connection with a stock symbol.

The following source documents shows the XML-Schema (Figure 4.4) and WSDL files for the FinancialDataProcessing service (Figure 4.5).

<?xml version="1.0" encoding="UTF-8"?>
<schema [...Namespaces... ]>
  <element name="submitStockInformationData">
    <complexType>
      <sequence>
        <element name="symbol" type="string"/>
        <element name="name" type="string"/>
        <element name="last" type="float"/>
        <element name="changeAbs" type="float"/>
        <element name="changeRel" type="float"/>
        <element name="previousClose" type="float"/>
        <element name="open" type="float"/>
        <element name="dayRange" type="string"/>
        <element name="Range-52wks" type="string"/>
        <element name="volume" type="integer"/>
        <element name="Rtn-1yr" type="float"/>
        <element name="daysRange" type="string"/>
      </sequence>
    </complexType>
  </element>
  <element name="returnStatus">
    <complexType>
      <sequence>
        <element name="status" type="boolean"/>
        <element name="statusName" type="string"/>
      </sequence>
    </complexType>
  </element>
  <element name="getStatisticsForStock">
    <complexType>
      <sequence>
        <element name="symbol" type="string"/>
      </sequence>
    </complexType>
  </element>
  <element name="complexStockInformationResponse">
    <complexType>
      <sequence>
        <element name="CurrentPERatioTTM" type="float"/>
        <element name="EstimatedPE" type="float"/>
        <element name="RelativePEvsSPX" type="float"/>
        <element name="EPS_TTM" type="float"/>
        <element name="EstimatedEPS" type="float"/>
        <element name="EstimatedPEGRatio" type="float"/>
        <element name="MarketCap" type="float"/>
        <element name="SharesOutstanding" type="float"/>
        <element name="DaysAvgVol30" type="integer"/>
        <element name="PriceBook" type="float"/>
        <element name="PriceSale" type="float"/>
        <element name="DividendIndXYield" type="float"/>
        <element name="CashDivident" type="float"/>
        <element name="DividentExDate" type="date"/>
        <element name="DividentGrowth5Yrs" type="float"/>
        <element name="NextEarningsAnnouncement" type="date"/>
      </sequence>
    </complexType>
  </element>
</schema>
The testing Application: ConsumingApplication

The ConsumingApplication - composite implements the testing environment to conduct the service interchange experiments. The experiments are conducted in the ‘ConsumingApp’ BPEL process, which is directly connected to the external service interface. The interface is defined to be as generic as possible, in order to invoke every service without having to adjust the interface to special in- and output parameters. The message payload elements are sets of various string, integer, float, and boolean variables, which can be mapped individually before invoking a service. The XML-Schema indicator 'minOccurs="0"' was added to allow empty parameters during service invocation. The following code snippets show excerpts of the Consuming Application’s XSD (Figure 4.6) and WSDL (Figure 4.6) files:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<schema ![...Namespaces...]> 
  <element name="inputData">
    <complexType>
      <sequence>
        <!-- [further elements] -->
        <element name="string0" type="string" minOccurs="0"/>
        <element name="string1" type="string" minOccurs="0"/>
        <!-- [further elements] -->
        <element name="integer0" type="integer" minOccurs="0"/>
        <element name="integer1" type="integer" minOccurs="0"/>
        <!-- [further elements] -->
        <element name="float0" type="double" minOccurs="0"/>
        <element name="float1" type="double" minOccurs="0"/>
        <!-- [further elements] -->
        <element name="boolean0" type="boolean" minOccurs="0"/>
        <element name="boolean1" type="boolean" minOccurs="0"/>
        <!-- [further elements] -->
      </sequence>
    </complexType>
  </element>
</schema>
```
As can be seen in the above schema and WSDL definitions, the interface consists of only one simple operation, which returns a message after a request. The two complex elements ‘inputData’ and ‘outputData’ consist of a set of optional simple element types to facilitate the individual in- and output of any data type. Since the elements should apply to any relevant service, they possess universal names such as ‘string0’, ‘string1’
etc. Consequently, output variables provide no semantic information about the meaning of variables. To distinguish variable values in the Enterprise Manager during testing, further parameters called “string labels” have been added to the interface, which enabled the developer to add a description label to each output parameter. The WSDL file shown above is short compared to the WSDL files defined earlier, since it does not need any bindings or service tag definitions. It is only deployed to the application server to be tested in the Oracle Enterprise Manager and does not need to be invoked out of other composite applications. Binding and service elements can be added if needed.

The ConsumingApplication -composite contains the ‘ConsumingApp’ BPEL process that was described earlier since it is the initial process implementation and starting point for the service interchange. The process invokes the operation “getComplexStockInformation” of StockQuoteService1 with a stock symbol identifier. To copy the input from the interfaces input to the service, an Assign activity is used. The invoked service synchronously returns the complex stock information. This then becomes the input data for both operations of the following FinancialDataProcessing service.

A ‘Transformation’ activity is used to map the output of StockQuoteService1 to the input of FinancialDataProcessing’s operation “submitData”. The Transformation activity in JDeveloper is an XSLT mapper, which creates and imports an XSL document [39]. Thereafter the FinancialDataProcessing service is invoked. The copy operation to assign the symbol output to the symbol input of the “getStats” operation, however, is realized in an additional Assign activity. Subsequently this service is invoked again, calling the “getStats” operation, which then returns additional statistical data of the selected financial asset.

The input format of “submitData” and “getStats” can be validated by XML-Schema restrictions and fractions so that it has to follow a certain format in order to be integrated/accepted by the service [22]. Note that this justifies the Transformation activity instead of an ordinary Assign activity, since it facilitates the use of XSLT transformations to fit the needs of the validation restrictions of certain services.

Since the FinancialDataProcessing service is invoked twice in order to call both available operations, its output is then returned to the ConsumingApp BPEL process’s output variable by another Transformation Activity's XSL file. This maps the output of both service invocations to the output variable of the invoking BPEL-process ‘ConsumingApp’. Again a Transformation activity is used to map the data to the output variables. The implemented BPEL process can be viewed in the following figure 4.8. The process elements are additionally labeled.16

---

16 Unfortunately the screenshot is a bit fuzzy, also due to scaling problems within the software. Though the details in the graphics Figure 4.8 and Figure 4.9 are not that important. The graphics are intended to give the reader a glimpse of how a BPEL process and a composite application in JDeveloper generally can look.
4.2.2 Xignite Global Quote (External Service)

The following section describes and justifies the selection of Xignite’s [53] Global Quote service and the choice of operations that are suitable for an exchange. Additionally, the service’s interface is analyzed to learn about the functionality of the service and commercial stock quote services in general. Basic invocation requirements and steps, that had to be performed in order to use the service, are also explained.

**Xignite Global Quote Service**

Xignite is a financial data vendor which offers a comprehensive amount of financial data APIs. The company offers a trial option for many of its services, which makes them useful for this examination. Due to the functional comparability to other stock quote
services, such as the self-defined StockQuoteService1, the GlobalQuote API of Xignite was selected as a candidate for testing the service interchange between two services that implement a close but different functionality while using a structurally different interface.

The GlobalQuotes service provides several operations, of which the operation “GetGlobalDelayedQuote” is the most applicable for the purpose of this examination. The other operation “GetGlobalDelayedQuotes” provides information for a whole set of stocks, but is basically the same operation as it uses the same output type. However, to completely document the service, and to further justify the choice of those two particular operations, the other available operations are listed below. The text shown in Table 4.1 is cited from the API website. The operations mentioned above are underlined.

**Table 4.1: List of Operations of: Xignite GlobalQuote (Legacy); Source: [53, 54]**

- GetGlobalDelayedQuote: Returns a global delayed quote.
- GetGlobalDelayedQuotes: Returns a collection of global delayed quotes.
- GetGlobalTick: Returns a single tick for a global security at specific time.
- GetGlobalTicks: Returns a set of tick-by-tick financial quotes for a stock and a time range during the trading day.
- GetDelayedChart: Returns a standard delayed intra-day chart for a security.
- GetDelayedChartPreset: Returns a preset delayed intra-day chart for a security.
- GetDelayedChartCustom: Returns a customized delayed chart for a security.
- GetDelayedChartDesign: Returns the default Design specification for the standard financial quotes chart.”

Since, when performing a service integration or exchange, the service implements various operations that have to be regarded, the corresponding WSDL file is relatively large and therefore only partially presented in this document. Removed sections of the WSDL are marked with comments. The full version can be accessed online.

The following WSDL data (Figure 4.10) shows an excerpt of Xignites GlobalQuotes WSDL-File. The testing, and consequently also the presented WSDL file, is limited to the relevant operation “GetGlobalDelayedQuote” and “GetGlobalDelayedQuotes”. The latter operation is also included as it implements a way of providing a full set of stock information within an array for multiple stocks, and is closely related to the first operation.

```xml
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions [...]>
  <wsdl:documentation [...]>This web service provides global delayed stock quotes and for U.S. and international equities.</wsdl:documentation>

  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://www.xignite.com/services/">
      <s:element name="GetGlobalDelayedQuote">
        <s:complexType>
          <s:sequence>
```
<s:element minOccurs="0" maxOccurs="1" name="Identifier" type="s:string" />
<s:element minOccurs="1" maxOccurs="1" name="IdentifierType" type="tns:IdentifierTypes" />
</s:sequence>
</s:complexType>
</s:element>
<s:simpleType name="IdentifierTypes">
<s:restriction base="s:string">
<s:enumeration value="Symbol" />
<s:enumeration value="CIK" />
<s:enumeration value="CUSIP" />
<s:enumeration value="ISIN" />
<s:enumeration value="Valoren" />
<s:enumeration value="SEDOL" />
</s:restriction>
</s:simpleType>
<s:element name="GetGlobalDelayedQuoteResponse">
<s:complexType>
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="GetGlobalDelayedQuoteResult" type="tns:GlobalQuote" />
</s:sequence>
</s:complexType>
</s:element>
<s:complexType name="GlobalQuote">
<s:complexContent mixed="false">
<s:extension base="tns:Common">
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="Security" type="tns:Security" />
<s:element minOccurs="0" maxOccurs="1" name="Date" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="Time" type="s:string" />
<s:element minOccurs="1" maxOccurs="1" name="Open" type="s:double" />
<s:element minOccurs="1" maxOccurs="1" name="High" type="s:double" />
<s:element minOccurs="1" maxOccurs="1" name="Low" type="s:double" />
<s:element minOccurs="1" maxOccurs="1" name="Last" type="s:double" />
<s:element minOccurs="1" maxOccurs="1" name="Volume" type="s:double" />
<s:element minOccurs="1" maxOccurs="1" name="PreviousClose" type="s:double" />
<s:element minOccurs="1" maxOccurs="1" name="Change" type="s:double" />
<s:element minOccurs="1" maxOccurs="1" name="PercentChange" type="s:double" />
<s:element minOccurs="1" maxOccurs="1" name="Bid" type="s:double" />
<s:element minOccurs="1" maxOccurs="1" name="Ask" type="s:double" />
<s:element minOccurs="0" maxOccurs="1" name="Currency" type="s:string" />
</s:sequence>
</s:extension>
</s:complexContent>
</s:complexType>
</s:element>
<s:complexType name="Common">
<s:sequence>
<s:element minOccurs="1" maxOccurs="1" name="Outcome" type="tns:OutcomeTypes" />
<s:element minOccurs="0" maxOccurs="1" name="Message" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="Identity" type="s:string" />
<s:element minOccurs="1" maxOccurs="1" name="Delay" type="s:double" />
</s:sequence>
</s:complexType>
<s:simpleType name="OutcomeTypes">
<s:restriction base="s:string">
<s:enumeration value="Success" />
<s:enumeration value="SystemError" />
<s:enumeration value="RequestError" />
<s:enumeration value="RegistrationError" />
</s:restriction>
</s:simpleType>
<s:complexType name="Security">
<s:complexContent mixed="false">
<s:extension base="tns:Common">
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="CIK" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="Cusip" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="Symbol" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="ISIN" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="Valoren" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="Name" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="Market" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="CategoryOrIndustry" type="s:string" />
</s:sequence>
</s:extension>
</s:complexContent>
</s:complexType>
<s:element name="Header" type="tns:Header" />
<s:complexType name="Header">
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="Username" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="Password" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="Tracer" type="s:string" />
</s:sequence>
</s:complexType>
<s:element name="GetGlobalDelayedQuotes"/>
<s:complexType>
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="Identifiers" type="s:string" />
<s:element minOccurs="1" maxOccurs="1" name="IdentifierType" type="tns:IdentifierTypes" />
</s:sequence>
</s:complexType>
</s:element>
<s:element name="GetGlobalDelayedQuotesResponse">
<s:complexType>
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="GetGlobalDelayedQuotesResult" type="tns:ArrayOfGlobalQuote" />
</s:sequence>
</s:complexType>
</s:element>
<s:complexType name="ArrayOfGlobalQuote">
<s:sequence>
<s:element minOccurs="0" maxOccurs="unbounded" name="GlobalQuote" nillable="true" type="tns:GlobalQuote" />
</s:sequence>
</s:complexType>
<!-- more schema definition -->
</wsdl:types>

<wsdl:message name="GetGlobalDelayedQuoteSoapIn">
<wsdl:part name="parameters" element="tns:GetGlobalDelayedQuote" />
</wsdl:message>
<wsdl:message name="GetGlobalDelayedQuoteSoapOut">
<wsdl:part name="parameters" element="tns:GetGlobalDelayedQuoteResponse" />
</wsdl:message>
<wsdl:message name="GetGlobalDelayedQuoteHeader">
<wsdl:part name="Header" element="tns:Header" />
</wsdl:message>
<wsdl:message name="GetGlobalDelayedQuotesSoapIn">
<wsdl:part name="parameters" element="tns:GetGlobalDelayedQuotes" />
</wsdl:message>
<wsdl:message name="GetGlobalDelayedQuotesSoapOut">
<wsdl:part name="parameters" element="tns:GetGlobalDelayedQuotesResponse" />
</wsdl:message>
<wsdl:message name="GetGlobalDelayedQuotesHeader">
<wsdl:part name="Header" element="tns:Header" />
</wsdl:message>
<!-- More Message Definitions -->
<wsdl:portType name="XigniteGlobalQuotesSoap">
<wsdl:operation name="GetGlobalDelayedQuote">
<wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Returns a delayed quote for a global security.</wsdl:documentation>
<wsdl:input message="tns:GetGlobalDelayedQuoteSoapIn" />
<wsdl:output message="tns:GetGlobalDelayedQuoteSoapOut" />
</wsdl:operation>
<wsdl:operation name="GetGlobalDelayedQuotes">
<wsdl:input message="tns:GetGlobalDelayedQuotesSoapIn" />
</wsdl:operation>
</wsdl:portType>
The <binding> and <service> sections have been removed from this WSDL excerpt, because they do not provide any new information about the functionality of this service. The binding is realized with SOAP using the document type and a literal-encoding.

Above the schema definition part, the WSDL contains a <description> tag which briefly defines the purpose of the service.

In the schema definition it can be seen that the potential input issue concerning different stock identifier types is solved by an additional element “IdentifierType”. It uses the simple type string as a restriction base for the valid expressions: ‘Symbol’, ‘CIK’, ‘CUSIP’, ‘ISIN’, ‘Valoren’ and ‘SEDOL’. These expressions define standardized ways to identify a stock. One of these expressions has to be submitted as a string during service invocation, alongside the actual stock identifier string.

As Output, the element ‘GetGlobalDelayedQuoteResponse’ is defined. It uses the complex type ‘GlobalQuote’ which encompasses the ordinary stock info payload as simple types. Additionally it is extended by the complex type ‘Common’ that contains an element ‘Security’. The complex type Common consists of metadata about the invocation of the service. This includes the elements ‘Outcome’, ‘Message’, ‘Identity’ and ‘Delay’. Outcome uses a simple type string restriction base and restricts to the expressions ‘Success’, ‘SystemError’, ‘RequestError’, ‘RegistrationError’. The output of the other variables can be viewed in the XML output file provided further below. The type ‘Security’ contains information about the financial security as well the values for all available Identifier types. The complex type security is further extended by the complex type ‘Common’. The element Header is of complex type Header, and defines the parameters being sent in the SOAP header. The complex type comprises the elements Username, Password and Tracer.

The second operation “GetGlobalDelayedQuotes” provides information for a set of stocks. It uses the same input as the “GetGlobalDelayedQuote” operation. The output ‘GetGlobalDelayedQuotesResponse’ contains a complex element of type ‘ArrayOfGlobalQuote’ which moreover contains an unbound element of type ‘GlobalQuote’ (the type of output like in the first operation) and therefore allows the arrangement of multiple stock information data sets. The type is “Nillable”, which means that the content of GlobalQuote type can also be NULL/empty. The rest of the WSDL is an ordinary WSDL file, which in also defines a message type for the Header element.

The following Figure 4.11 shows an example XML output of the operation GetGlobalDelayedQuote. This output is necessary in order to fully understand the semantics and data formats provided by the service operation.

17 The identifier type might differ from user to user. For example, within European markets, the ISIN number is more commonly used than the (Stock) symbol. (In Non-US Markets the Symbol has to be further Qualified by prefixes e.g. ‘XETRA.BMW’ in order to be able to identify an nearly unlimited amount of stocks, whereas the ISIN is unique but more abstract)
The XML output describes the semantics and encoding of the operations output. This is important additional information to the interfaces static description and facilitates the work with the API. This message provides, for example, insights to the content of the \(<message>\) and \(<identity>\) tags. It is apparent, that the service responds to every stock identifier and therefore it can be used for translation purposes between different identifiers. The tags and data outputs are further described in the online documentation [54].

4.3 Service Interchange and Testing

This part documents the actual testing of the testing scenarios. The tests were conducted in the order of the predefined testing scenarios, starting with scenario 1.

4.3.1 Scenario 1: Same Interface, Same Service

Within this section, the service interchange of two functionally equal services is examined, where both use the same exact interface.

The first Scenario 1 was examined by exchanging the service StockQuoteService1 against the service StockquoteService2. Both services implement the same functionality.
and interface. The data output format of both services were also identical in the first test.

Because the WSDL files are totally identical, they also use the same names and namespaces for schema elements, types, messages, port types etc. Consequently the services were easily interchangeable by just altering the WSDL file URL within the partner link definition.

However if the data encoding format of the output between StockQuoteService1 and StockQuoteService2 differs, for example if a different database back-end is used, the exchange still will not technically be a problem, but it will lead to a different data output. This causes problems later, for example if a subsequent service such as the FinancialDataProcessing expects data in a certain encoding format. Consequently the data output needs to be further transformed or translated to meet the expected data requirements. This is a general issue which is applicable to all scenarios. It is explained within the context of this scenario, because the technical exchange process is not a problem in itself.

4.3.2 Scenario 2: Different Interface, Same Service

Within this section, the service interchange of two functionally equal services is examined. The StockQuoteService1 was exchanged for the StockInfo service. Both services use interfaces which provide the same structure. Only the identifier names and namespace definitions differ. Again the testing was performed by exchanging the service’s WSDL URL within the partner link definition of the BPEL process. Since the referencing in WSDL and XML-Schema is completely text-based, all relevant identifier names, had to be adjusted, including the URI of the namespace definition.

The adjustments to be made are comprehensive. Every activity which uses a reference from the WSDL or XSD needs to be adjusted. The definition of variables has to be altered if a variable uses a message type defined in the WSDL file. If the variable name does not suit its purpose anymore, it should be changed as well, which leads to further changes everywhere the variable is used, such as in the ‘Invoke’ activities. The following Figure 4.12 shows the variables definition within BPEL and necessary points of adjustments.

```
<variables>
  <variable name="inputVariable1" messageType="ns1:nameOfMessageType"/>
  <variable name="outputVariable1" messageType="ns1:nameOfMessageType"/>
  <variable name="simpleString" type="xsd:string"/>
</variables>
```

*Figure 4.12: Variables definition in ConsumingApp BPEL*

The port type definition in the Invoke activity must be changed too, if the WSDL uses a different port type name. This is shown in Figure 4.13.

```
<invoke name="Invoke1"
  inputVariable="inputVariable1"
  outputVariable="outputVariable1"
  partnerLink="PartnerLink1" portType="ns2:StockQuoteServicePort"
  operation="getSimpleStockInformation" bpelx:invokeAsDetail="no"/>
```

*Figure 4.13: Invoke activity definition in ConsumingApp BPEL*
Furthermore XPath expressions are used in the Assign activities to copy data from one parameter to another. These path names, which also are used within XSL files, have to be changed as well. Figure 4.14 shows an Assign activity.

```xml
<assign name="Assign4">
  <copy>
    <from [...]/>
    <to variable="InvokeStockInfo_getComplexStockInformation_Input Variable" part="payload" query="/ns3:complexStockInformationRequest/ns3:symbol"/>
  </copy>
</assign>
```

Figure 4.14: Assign activity definition in ConsumingApp BPEL

Oracle automatically generates a reference of the new integrated service in the SCA canvas, because the Invoke activity had to be altered and rewired. This new reference has to be wired manually to the BPEL process in order to define a required “wire target”. The reference of the original service will still be displayed in the SCA. Figure 4.15 shows the wire within the SCA.

```xml
<wire>
  <source.uri>ConsumingApp/StockInfo</source.uri>
  <target.uri>StockInfo</target.uri>
</wire>
```

Figure 4.15: Definition of Wires in the ConsumingApplication SCA

Since this is an almost all-encompassing alteration process, it can be more practical to remove the original elements and variables and substitute them by newly generated pendants, related to the new service. This approach is explained further in Scenario 3.

### 4.3.3 Scenario 3: Different Interface, Related Service

This section examines the service interchange of functionally related services that use a different interface. The scenario describes, according to previous market research, the most common expected scenario in the cloud service market today. Different cloud service providers deliver their own solutions for business problems - as a service. Different financial market data providers, for example, provide varying financial data information sets on their web portals. Consequently it can be assumed that their API interfaces also vary, since there are no standards for the semantics of financial data API interfaces (yet).

In this example the self-created StockQuoteService1, which is based on Bloomberg’s stock quote data, will be replaced by a commercial stock quote service of Xignite\(^1\). The service can be used for testing purposes by registering for a trial period at Xignite which enables the user to call the service 250 times within a seven-day trial period. Xignite furthermore provides a full catalogue of different financial APIs to choose from. This subchapter documents the changes that have to be performed in order to exchange this kind of service within the ConsumingApplication composite.

**Testing the Xignite Web Service GlobalQuote**

In order to be able to test the Xignite stock quote service, a trial account was set up at

---

\(^1\) The service GetGlobalDelayedQuote was introduced in sub-chapter 4.2.2
the Xignite website\(^{19}\). To obtain access to the GlobalQuote stock service, a subscription had to be made for the particular service. After subscription, the website provides a Verification Key: 8630e2e8-d856-4520-9185-27d8ceaba53 (example) and a 32 digits alphanumeric API Token '5C5GGGACDF746B892BDD468B2EF085A' (example). The token acts as the authentication username to access the API. The testing token is limited to 250 hits per API and expires after a period of seven days. An authentication password can be set up in the account management section of the website. Both username and password have to be sent via the SOAP header in order to invoke the service. Only one API can be tested at a time.

To invoke the operation “GetGlobalDelayedQuote”, a Username/Password combination, as well as a valid stock identifier, had to be inputted. The service allows the input of various standardized identifier types such as the ordinary (stock) Symbol, or the ISIN (International Stock Identification Number), further formats are supported as well. As mentioned before, the input format has to be declared in the second required input variable ‘Identifier Type’.

The Exchange

First, the ConsumingApplication composite was loaded into JDeveloper, which implemented the process in the examination of the first scenario - using StockQuoteService in connection with the FinancialDataProcessing service. To ensure that this application was implemented correctly as a starting point, the application was deployed again and then functionally tested in the Oracle Enterprise Manager. The testing input variable 'String0' was set to ‘ORCL’ and the service was invoked – the expected information was returned successfully.

Now the actual service interchange started. There are two ways of manually interchanging a service, either by altering and adjusting everything in the source code manually (this was the case in Scenario 2) or by generating every relevant activity anew. The latter method was chosen in order to prevent reference errors and to be able to reverse the changing process quickly. To remove the original activities from the process all relevant definitions were put into comments <!-- … --> within the source code. If these elements are not removed properly, JDeveloper returns an error messages during compilation and deployment.

The following documentation of the exchange process also shows all activities in the source view. This is important to understand the consequences of the tasks performed in JDeveloper. Moreover the shown BPEL source code is mostly independent of developing environments.

First the service WSDL URL\(^{20}\) was copied into a new partner link, which was created in the ‘ConsumingApp’ BPEL process. After entering the URL, JDeveloper generates a wrapper WSDL file with new partner link definitions, since these are not included in the provider’s file. The WSDL is then referenced in the wrapper file and can be accessed via the WSDL design view.

To invoke the created partner link, an Invoke activity had to be created by dragging it into the BPEL sequence. It then was wired to the partner link. As the Invoke activities dialogue opened, the name of the activity was set and the correct operation to be invoked was chosen. After that in- and output variables were created automatically. The

---

\(^{19}\) https://www.xignite.com/

\(^{20}\) http://www.xignite.com/xGlobalQuotes.asmx?wsdl
composite.xml (SCA) now contained a new reference to the GlobalQuotes service. This reference needed to be wired manually to the BPEL process.

A new Namespace definition was created automatically for the service in the BPEL process source code. A new Partner Link and new Variables were added to the source code by performing the activities explained above. Partner link type and message types were qualified with the namespaces prefix ‘ns3’.

The Figure 4.16 shows the ConsumingApp BPEL Process after the Xignite service integration.

```xml
xmlns:ns3="http://www.xignite.com/services/
...
<partnerLinks>
  ...
  <partnerLink name="GlobalQuotes"
    partnerLinkType="ns3:XigniteGlobalQuotesSoap_PL"
    partnerRole="XigniteGlobalQuotesSoap_Role"/>
</partnerLinks>
...
<variables>
  ...
  <variable name="InvokeGlobalQuotes_GetGlobalDelayedQuote_InputVariable"
    messageType="ns3:GetGlobalDelayedQuoteSoapIn"/>
  ...
  <variable name="InvokeGlobalQuotes_GetGlobalDelayedQuote_OutputVariable"
    messageType="ns3:GetGlobalDelayedQuoteSoapOut"/>
</variables>
...
```

*Figure 4.16: ConsumingApp BPEL process after Xignite integration*

**SOAP Header Authentication**

The GlobalQuote service requires an authentication during its invocation. In order to invoke the service, an authentication via the SOAP Header had to be implemented. The services WSDL contains a special message type for this task. The details can be viewed in the schema definition part of the WSDL file. The WSDL file can be accessed by clicking the import button within the design view of the GlobalQuotesWrapper.wsdl file. The Figure 4.17 below shows the design views of the WSDL wrapper and the actual imported WSDL. In the latter, the particular message type is marked.

*Figure 4.17: WSDL design view and Header Message Element*
To actually use the header message type, a variable had to be created, which uses this message type. The variable was given the declarative name “SoapHeaderAuthentication”. To send the variables payload within the service invocation, the expression ‘bpelx:inputHeaderVariable="SoapHeaderAuthentication"’ had to be copied into the source code of the Invoke activity, right after the in- and output variables definitions.

The service was now usable but still needed to be integrated into the BPEL process. In order to be invokable, the service needed an input to be assigned. Furthermore the service should deliver data to the subsequent FinancialDataProcessing service, which, however, has a different interface. Consequently the existing Assign activities had to be substituted. Because of different data types, further transformations had to be performed.

Comparing the two Services

In this experiment 'StockQuoteService1' has to be substituted by the 'GlobalQuotes' service. The 'FinancialDataProcessing' service is well-aligned to the output-parameters of 'StockQuoteService1'.

To maintain the functionality of the BPEL process, corresponding parameters have to be identified in order to substitute the other service’s parameters and data. This process has to be performed manually, as there is no semantic information connected to the interface definition.

The data model might also not be identical to the ones of the previous service, therefore they might have to be transformed. The data requirements of the consecutive service have to be met, firstly to enable an error-free invocation of the services, and secondly to avoid possible data integrity issues, which would be particularly likely to occur in combination with a database backend.

The following Table 4.2 compares the input data of FinancialDataProcessing’s “submitData” operation with output data of the GlobalQuotes service. This particular constellation assumes that the submitData operation is an operation to save and process financial data provided by a service.

<table>
<thead>
<tr>
<th>FinancialDataProcessing - submitData</th>
<th>GlobalQuotes - GetGlobalDelayedQuote</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbol (String)</td>
<td>Symbol (String) – element</td>
</tr>
<tr>
<td></td>
<td>“Security“ type=&quot;tns:security&quot;</td>
</tr>
<tr>
<td>Name (String)</td>
<td>Name (String) – element</td>
</tr>
<tr>
<td></td>
<td>“Security“ type=&quot;tns:security&quot;</td>
</tr>
<tr>
<td>Open (Float)</td>
<td>Open (Double)</td>
</tr>
<tr>
<td>DayRange (String) – Higher Value</td>
<td>High (Double)</td>
</tr>
<tr>
<td></td>
<td>Low (Double)</td>
</tr>
<tr>
<td>Last (Float)</td>
<td>Last (Double)</td>
</tr>
</tbody>
</table>

21 Relating to the hypothetical scenario of the financial information provider company, this might be the case, because in the past only 'StockQuoteService1' was used to deliver stock information. The service 'FinancialDataProcessing', a self-developed application, was adapted accordingly. Now the company wants to be able to substitute the service with other external services.

22 This might not be “best practice” and is simply a hypothetical scenario used to explain the issues of different interfaces which need more than one standardized identifier for a valid invocation.
<table>
<thead>
<tr>
<th>Volume (Integer)</th>
<th>Volume (Double)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PreviousClose (Float)</td>
<td>PreviousClose (Double)</td>
</tr>
<tr>
<td>ChangeAbs (Float)</td>
<td>Change (Double)</td>
</tr>
<tr>
<td>ChangeRel (Float)</td>
<td>PercentChange (Double)</td>
</tr>
<tr>
<td>Range-52wks (String)</td>
<td></td>
</tr>
<tr>
<td>Rtn-1yr (Float)</td>
<td>Bid (Double)</td>
</tr>
<tr>
<td></td>
<td>Ask (Double)</td>
</tr>
<tr>
<td></td>
<td>Currency (String)</td>
</tr>
</tbody>
</table>

Table 4.2: Comparison of submitData and GetGlobalDelayedQuote

The Xignite GlobalQuotes service provides parameters which are not included in the interface of the target service, furthermore it does not provide parameters such as a 52 week range or a 1 year return parameter. Variables which do not exist in an exchanging operation can consequently not be substituted and have to be implemented alternatively, so as not to lose functionality. Because the exchange is GlobalQuotes for StockQuoteService1, Range-52wks and Rtn-1yr cannot be provided. In reverse it would be an issue, if the target process would use the information provided in the ‘Bid’, ‘Ask’ etc. parameters that StockQuoteService1 does not.

The interface also contains parameters which have different data types or even use different approaches to deliver information. Noticeable distinctions in data types in this case are different decimal numbers such as floats in service1 and doubles in service2, integers in service1 and doubles in service2, or strings for decimal numbers. The interoperability depends on the situation and is not a problem in this particular case. It could be a problem, however, if, for example, large numbers with many decimal places had to be calculated exactly. In this case, a conversion from double to float would cause a loss in accuracy and consequently maybe also a loss of money. Generally speaking, there are functions to convert data types, but the payload of data might be affected.

The parameters such as ‘High’ and ‘Low’ are examples of a different approach, instead of a single ‘dayRange’ parameter, which condenses two decimal numbers as a string data type. Another example, which is not included in the table above, is the date as a string type instead of a date type. This issue is especially relevant if the date string has a different format from the date type format e.g. ‘DD/MM/YYYY’ instead of ‘YYYY-MM-DD’. The date example is not included in the table above, since this table only analyzes the “functional payload” in connection to the subsequent Financial-DataProcessing service. The Xignite service however, includes further meta-data where this particular example is included. If the parameters of StockQuoteService1 and GlobalQuotes are compared, the first service uses a date type whereas the second uses a string.

Handling of disused BPEL Elements

BPEL elements of the original services were not altered but substituted for this service interchange. Adjustment works related to the exchange process are far more complex and error prone than the creation of new elements. Original elements that are not used

---

\(^{23}\) During the testing, the IDE allowed e.g. the assignment of doubles to float values
anymore needed to be removed from the process so as to not generate error messages during deployment. They could either be deleted in the design view or source code of the BPEL process, or alternatively set into comment tags ‘<!—[...]-->’. This also included the definition of generated variables which were no longer used.

**Substitution of Assign Activities**

After the Xignite service was implemented in the BPEL process, the existing Assign-Activities had to be substituted. If a service is exchanged and an Assign activity references parameters of this service, all references either <from> or <to> within the <copy> tag need to be adjusted. In this case the part names did not have to be adjusted, but a different service may also use a different part name. Furthermore, if the Assign activity had a describing name, it should have been changed too, even if this was not required. The Assign activity is shown in Figure 4.18.

```xml
<assign name="Assign1">
  <copy>
    <from variable="inputVariable" part="payload" query="/client:inputData/client:string0"/>
    <to variable="InvokeStockQuote_getComplexStockInformation_InputVariable" part="payload" query="/ns1:complexStockInformationRequest/ns1:symbol"/>
  </copy>
</assign>
```

*Figure 4.18: Original Assign activity “Assign1”*

An Assign activity can contain various <copy> operations, copying from and to different variables. During changes, that occur, for example during a service interchange, adjustments have to be applied individually. One Assign activity is theoretically not limited to the copy operations between two particular services but can also contain “mixed” copy operations between various services. In addition, if two subsequent services within a BPEL process are changed, every statement <from> and <to> in the <copy> operation might have to be changed. Changes comprise variable names and paths and namespace qualifications. There cannot be any copy operations that are no longer used. In certain scenarios, besides the <copy> operations, transformations might also have to be altered within an Assign activity. Due to the issues mentioned above, new Assign activities were created for the integration of the Xignite service. The actual Assign activities used to integrate the process are shown below in Figure 4.19:

```xml
<assign name="AssignInputForXignite">
  <copy>
    <from variable="inputVariable" part="payload" query="/client:inputData/client:string0"/>
    <to variable="InvokeGlobalQuotes_GetGlobalDelayedQuote_InputVariable" part="parameters" query="/ns3:GetGlobalDelayedQuote/ns3:Identifier"/>
  </copy>
  <copy>
    <from expression="Symbol"/>
    <to variable="InvokeGlobalQuotes_GetGlobalDelayedQuote_InputVariable" part="parameters" query="/ns3:GetGlobalDelayedQuote/ns3:IdentifierType"/>
  </copy>
</assign>
```

*Figure 4.19: Assign activity "AssignInputForXignite"*
The second <copy> operation of this Assign activity contains an expression to assign the String 'Symbol' to the variable 'IdentifierType'. The expression 'Symbol' was set within the Assign activity. Alternatively it could also be copied from a second input variable of the composite application. This method was preferred since only Symbols were used in the tests.

**Assign Authentication**
To authenticate during the invocation it was also necessary to assign the actual credentials to the previously defined variable “AssignAuthInfo”. To implement this, another Assign activity was set up. In this example the authentication credentials were simply added by using expressions. The token provided by Xignite for this particular service was used as the Username. In reality, the password should not be sent like this but be encrypted and signed. Figure 4.20 shows the Assign activity that assigns the credentials.

```xml
<assign name="AssignAuthInfo">
  <copy>
    <from expression="'5C5GGGGACDF746B892BDD468B2EF085A'"/>
    <to variable="SoapHeaderAuthentication" part="Header" query="/ns3:Header/ns3:Username"/>
  </copy>
  <copy>
    <from expression="'MyPasswordsWildcard'"/>
    <to variable="SoapHeaderAuthentication" part="Header" query="/ns3:Header/ns3:Password"/>
  </copy>
</assign>
```

*Figure 4.20: Assign activity "AssignAuthInfo"*

**Transform Activities**
To transmit the output data of the “GetGlobalDelayedQuote” operation, its output-variables had to be mapped to the input variables of FinancialDataProcessing. For this task the comparison between the two services, described above, could be used. Another Assign activity could be used for this purpose, but this solution would be rather confusing, especially when using many expressions for transformation purposes. For the latter, Oracle provides the Transformation activity which implements an XSLT mapper, which generates an XSL file, which is then referenced in BPEL.

In this case a transformation element was dragged into the canvas from the component palette (Oracle extensions)\(^{24}\). To define the basics for the transformation file, the source variables and the target variables had to be defined. Within the XSLT mapper, the variables were mapped to the submitData input variables. Available XSLT functions can be dragged to the center to the connecting wires to perform a transformation. Figure 4.21 shows the transformation.\(^{25}\)

\(^{24}\) There are alternative XSLT mappers available.
\(^{25}\) Since this is a simple/sample transformation, the details are not important. It is however important to get an idea of what has to be done manually, which is to connect the parameters from left to right and in addition add, if required, operations.
This mapping contained a 'concat' operation, which joined the ‘High’ and ‘Low’ values of the GlobalQuote service to the expected string ‘dayRange’. The concat operation is implemented like this in the XSLT file (Figure 4.22):

```xml
<client:dayRange>
</client:dayRange>
```

**Assign Input for GetStats**

To assign the needed input data to the “getStats” operation of the FinancialDataProcessing service, the 'symbol' output variable of the GlobalQuotes service was mapped to the 'symbol' variable of the FinancialDataProcessing “getStats” operation input variable. The mapping is not further documented because it did not differ from the others.

**Assign Output to the process’ Output Variables**

To complete the process, the output-variables of the invoked service operations had to be mapped to the output variables of the BPEL process. This also returns the data back to the Mediator and, consequently, back to the interface of the composite application.

For this purpose, again, a Transform activity was chosen to facilitate the assignment in a clear manner. Alternatively it would also be possible to use an Assign activity, but in order to deliver the output data to a potentially fixed composed service interface, a Transform activity is the preferred method, because it enables the developer to transform the assigned data into the formats required.

The Figure 4.23 shows the dialogue ‘Edit Transform’ and the definition of three sources from three different operations. The target variable is the ‘outputVariable’ of the BPEL process. The generation of such a XSL file resulted in a relatively long loading time.
The following Figure 4.24 show the XSLT and the finished BPEL process. The parameters of the defined sources were all mapped to the output parameters of the process, which are also equal to the output parameters of the composed service. Since the Consuming Application uses a neutral testing interface, the mappings were enriched by the string labels. Finally the composite application was saved, deployed and tested in the Enterprise Manager.

As notable, almost the whole BPEL process had to be recreated, because the exchange of only one service caused adjustments that affected almost all employed BPEL
activities. A manual service interchange within BPEL seems consequently, without further measures, to be too complex to be sensible.

4.3.4 Scenario 4: Different Interface, Different Service
Irrespective of the semantics of a service or service interface, it is technically possible to allocate any service parameters to the input parameters of a service. There are services which are functionally different but related in terms of their schema or structure, especially synchronous services that follow a request-response method of data delivery. If the services are moreover in the same business domain, as, for example, a stock quote service and an option price service. The data outputs become even more comparable. Technically speaking, an option price service could consequently deliver data to a stock quote service’s interface, because its parameters allow any decimal and string values without checking the meaning of the content.

This scenario is not particularly realistic and the practical examination is not further explained, but it shows the importance of a semantic analysis during the service interchange. As there are no semantic annotations commonly available to WSDL interface descriptions, services need to be integrated by humans, which can be very time-consuming.

4.3.5 Scenario 5: Service Interchange within a composed Application
The previously examined scenarios comprised the analysis of challenges during service interchange with the focus on service interaction. This interaction occurred within a BPEL process embedded to a testing application, whose output was not important. The composite application used a generic interface in order to test any service constellation and enable any output. However, it is a part of the service-oriented idea, that services can be composed into new composed services. These services then need a steady interface and data output to meet data integrity needs and provide the predefined Quality of Service. This scenario documents the additional characteristics, and requirements, of a service interchange within a composed service that is externally or internally exposed as an API. Further requirements have to be satisfied, especially if provided to external consumers.

Provide an API for external Consumers
The financial services company planned to provide its business services as an API to external customers. It planned to use a SOA composite application that utilizes external APIs, adds its own statistical services and assembles all parts into its own composed service. To achieve the best Quality of Service, the company intended to exchange its internally employed services individually. To facilitate this exchange, in connection with a steady flow of information, a well-designed interface had to be defined.

To generate such an interface, a company has to learn about the nature of common stock quote services and the in- and output parameters they provide. The interface of the composite application has to be a compromise of the usable stock quote services. Strictly speaking, an interface has to contain the intersection of all parameters which are offered by the different providers. Since particular parameters are less important than others, some parameters could be omitted in order to increase the number of service candidates that conform to the interface of the composite application. There must be a trade-off between a large variety of exchangeable services and the information that the API can provide.

After the interface has been defined, it subsequently creates a certain profile for the future selection of additional stock quote service candidates, in the context of a service interchange. Services which are not able to provide or derive the required information,
i.e. to serve the set output parameters, can thus not be used for an interchange. In the following section the definition of such an interface is presented.

**The Definition of a common Interface**

To define a common interface for the composed services API, further stock quote services were analyzed. In this particular case, three additional services were identified and taken into account, together with the two services examined before.

The three additional services are described briefly in the subsequent section, following a comparison of all five services output parameters:

**Web Service X StockQuote – Operation: “GetQuote”**

The service is freely available but provides neither up-to-date, nor complete data. Furthermore the output XML data is wrapped in a string and needs to be parsed. Further XSD information is not provided. As such, this service is only used as an example to show different output parameters in functionally related services. The following graphic (Figure 4.25) shows the services XML output for the ‘ORCL’ stock symbol as it is.

```xml
<string><StockQuotes><Stock><Symbol>ORCL</Symbol><Last>44.19</Last><Date>1/23/2015</Date><Time>4:01pm</Time><Change>+0.14</Change><Open>43.98</Open><High>44.54</High><Low>43.92</Low><Volume>14150264</Volume><MktCap>194.1B</MktCap><PreviousClose>44.05</PreviousClose><PercentageChange>+0.32%</PercentageChange><AnnRange>35.44 - 46.71</AnnRange><Earnings>2.404</Earnings><P-E>18.32</P-E><Name>Oracle Corporation</Name></Stock></StockQuotes>
</string>
```

*Figure 4.25: XML Output of Web Service X StockQuote*

For the purpose of this examination, the XML nodes were taken as individual parameters and suitable data types were added. The identified parameters are illustrated in the subsequent Table 4.3:

<table>
<thead>
<tr>
<th>Symbol – String</th>
<th>Volume – Int</th>
</tr>
</thead>
<tbody>
<tr>
<td>Last – Double</td>
<td>MktCap – String</td>
</tr>
<tr>
<td>Date – String</td>
<td>PreviousClose – Double</td>
</tr>
<tr>
<td>Time – Time</td>
<td>PercentChange – String</td>
</tr>
<tr>
<td>Change – Double</td>
<td>AnnRange – String</td>
</tr>
<tr>
<td>Open – Double</td>
<td>Earnings – Double</td>
</tr>
<tr>
<td>High – Double</td>
<td>P-E – Double</td>
</tr>
<tr>
<td>Low – Double</td>
<td>Name – String</td>
</tr>
</tbody>
</table>

*Table 4.3: List of Parameters of WebServiceX StockQuote; Source: [55]*

**cdyne DelayedStockQuote – Operation: “GetQuote”**

This service is also freely available. It provides three operations of which the operation “Get Quote” is comparable to the operations examined and implemented before. It provides a set of information for the input of a stock identifier. The service requires a ‘LicenceKey’ which, for testing purposes, can be set to ‘0’. The service uses the data type decimal for decimal numbers and a dateTime data type for the last trade date and time. The in- and output parameters can be viewed in the following XSD excerpt (Figure 4.26). The second part of the figure shows an example XML output to show the actual data output.
Schema excerpt for Input:

```xml
<s:element minOccurs="0" maxOccurs="1" name="StockSymbol" type="s:string"/>
<s:element minOccurs="0" maxOccurs="1" name="LicenseKey" type="s:string"/>
```

Schema excerpt for Output:

```xml
<s:element minOccurs="0" maxOccurs="1" name="StockSymbol" type="s:string"/>
<s:element minOccurs="1" maxOccurs="1" name="LastTradeAmount" type="s:decimal"/>
<s:element minOccurs="1" maxOccurs="1" name="LastTradeDateTime" type="s:dateTime"/>
<s:element minOccurs="1" maxOccurs="1" name="StockChange" type="s:decimal"/>
<s:element minOccurs="1" maxOccurs="1" name="OpenAmount" type="s:decimal"/>
<s:element minOccurs="1" maxOccurs="1" name="DayHigh" type="s:decimal"/>
<s:element minOccurs="1" maxOccurs="1" name="DayLow" type="s:decimal"/>
<s:element minOccurs="1" maxOccurs="1" name="StockVolume" type="s:int"/>
<s:element minOccurs="0" maxOccurs="1" name="ChangePercent" type="s:string"/>
<s:element minOccurs="0" maxOccurs="1" name="FiftyTwoWeekRange" type="s:string"/>
<s:element minOccurs="1" maxOccurs="1" name="EarnPerShare" type="s:decimal"/>
<s:element minOccurs="1" maxOccurs="1" name="PE" type="s:decimal"/>
<s:element minOccurs="0" maxOccurs="1" name="CompanyName" type="s:string"/>
<s:element minOccurs="1" maxOccurs="1" name="QuoteError" type="s:boolean"/>
```

Output SOAP Message of Request ‘ORCL’ ‘0’

```xml
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body>
<GetQuoteResponse xmlns="http://ws.cdyne.com/">
<GetQuoteResult>
<StockSymbol>ORCL</StockSymbol>
<LastTradeAmount>40.25</LastTradeAmount>
<LastTradeDateTime>2014-08-14T09:37:00</LastTradeDateTime>
<StockChange>0.01</StockChange>
<OpenAmount>40.29</OpenAmount>
<DayHigh>40.40</DayHigh>
<DayLow>40.23</DayLow>
<StockVolume>296991</StockVolume>
<PrevCls>40.24</PrevCls>
<ChangePercent>+0.02%</ChangePercent>
<FiftyTwoWeekRange>31.56 - 43.19</FiftyTwoWeekRange>
<EarnPerShare>2.38</EarnPerShare>
<PE>16.91</PE>
<CompanyName>Oracle Corporation</CompanyName>
<QuoteError>false</QuoteError>
</GetQuoteResult>
</GetQuoteResponse>
</soap:Body>
</soap:Envelope>
```

Figure 4.26: cdyne DelayedStockQuote XML-Schema excerpt and SOAP output; Source [56]
Markitondemand – Stock Quote

This service is used to describe a further stock service with differing output parameters. This service does not provide a WSDL, but in this case the sole description of possible output data for a service is sufficient. The description of the in- and output parameters can be accessed from on the Markitondemand website [57]. Table 4.4 shows the parameters of the service.

<table>
<thead>
<tr>
<th>Status – String</th>
<th>MarketCap – Double</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name – String</td>
<td>Volume – Int32</td>
</tr>
<tr>
<td>Symbol – String</td>
<td>ChangeYTD – Double</td>
</tr>
<tr>
<td>LastPrice – Double</td>
<td>ChangePercentYTD – Double</td>
</tr>
<tr>
<td>Change – Double</td>
<td>High – Double</td>
</tr>
<tr>
<td>ChangePercent – Double</td>
<td>Low – Double</td>
</tr>
<tr>
<td>Timestamp – String</td>
<td>Open – Double</td>
</tr>
<tr>
<td>MSDate – Double</td>
<td></td>
</tr>
</tbody>
</table>

*Table 4.4: Markitondemand Parameters*

Comparison of Output Parameters

The following Table 4.5 shows an overview of all output parameters provided by the different APIs. In addition, the input of the “submitData” operation of FinancialDataProcessing is shown. The parameters are sorted to show the conformity between the different services’ output data.
Table 4.5: Comparison of Output Parameters

<table>
<thead>
<tr>
<th>WebserviceX StockQuote (GetQuote)</th>
<th>Cdyne DelayedStockQuote (GetQuote)</th>
<th>Markitondemand (StockQuote)</th>
<th>XigniteGlobalQuotes (GetGlobalDelayedQuote)</th>
<th>StockQuoteService1 - FinancialDataProcessing - (submitData - input)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbol (String)</td>
<td>StockSymbol (String)</td>
<td>Symbol (String)</td>
<td>Symbol (String)</td>
<td>Symbol (String)</td>
</tr>
<tr>
<td>Name (String)</td>
<td>CompanyName (String)</td>
<td>Name (String)</td>
<td>Name (String)</td>
<td>Name (String)</td>
</tr>
<tr>
<td>Last (Double)</td>
<td>LastTradeAmount (Decimal)</td>
<td>LastPrice (Double)</td>
<td>Last (Double)</td>
<td>Last (Float)</td>
</tr>
<tr>
<td>Change (Double)</td>
<td>StockChange (Decimal)</td>
<td>Change (Double)</td>
<td>Change (Double)</td>
<td>ChangeAbs (Float)</td>
</tr>
<tr>
<td>PercentageChange (String)</td>
<td>ChangePercent (Double)</td>
<td>PercentChange (Double)</td>
<td>ChangeRel (Float)</td>
<td>ChangeRel (Float)</td>
</tr>
<tr>
<td>Open (Double)</td>
<td>OpenAmount (Decimal)</td>
<td>Open (Double)</td>
<td>Open (Double)</td>
<td>Open (Float)</td>
</tr>
<tr>
<td>High (Double)</td>
<td>DayHigh (Decimal)</td>
<td>High (Double)</td>
<td>High (Double)</td>
<td>DayRange (String)</td>
</tr>
<tr>
<td>Low (Double)</td>
<td>DayLow (Decimal)</td>
<td>Low (Double)</td>
<td>Low (Double)</td>
<td>DayRange (String)</td>
</tr>
<tr>
<td>PreviousClose (Double)</td>
<td>PrevCl Price (Decimal)</td>
<td>PreviousClose (Double)</td>
<td>PreviousClose (Float)</td>
<td>PreviousClose (Float)</td>
</tr>
<tr>
<td>Volume (Int)</td>
<td>StockVolume (Int)</td>
<td>Volume (Int32)</td>
<td>Volume (Double)</td>
<td>Volume (Integer)</td>
</tr>
<tr>
<td>AnnRange (String)</td>
<td>FiftyTwoWeekRange (String)</td>
<td>X</td>
<td>X</td>
<td>Range-52wks (String)</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Rtn-1yr (Float)</td>
</tr>
<tr>
<td>MktCap (String)</td>
<td>MarketCap (Double)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Earns (Double)</td>
<td>EarnPerShare (Decimal)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>P-E (Double)</td>
<td>PE (Decimal)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>QuoteError (Bool)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>Status (String)</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Date (String)</td>
<td>LastTradeDateTime (dateTime)</td>
<td>Timestamp (String)</td>
<td>Date (String)</td>
<td>X</td>
</tr>
<tr>
<td>Time (String)</td>
<td>MSDate (Double)</td>
<td>Time (String)</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>Bid (Double)</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>Ask (Double)</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>Currency (String)</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

The table above shows, that the first ten parameters have equivalents in all of the alternative services. These parameters should consequently be used to define the interface of the API. Many parameters differ in terms of data types and formats and will need to be adjusted accordingly. The APIs interface however shall use data types, which are reasonable and require the least adjustments. These differences are analyzed below and a possible selection for the data types used in the API interface will be made.

It can be seen that several different number types are used: float, double and decimal. To prevent the loss of information, no matter what numbers are processed, the largest data type decimal would be the best choice for the APIs interface. This would, however, also be a tradeoff, because big data types perform worse than smaller types and are not
always necessary. Due to the limited complexity of the numbers, a double type is used for the interface. Decimal values will need to be converted.

In one particular case the Cdyne service provides a string instead of a decimal number for the ‘ChangePercent’ parameter. The data is provided in the following format: “+0.02%”. Such a format has positive and negative consequences for the data. A positive aspect is, that the percentage data can be clearly identified by a human user as 0.02% instead of 0.02, which also could be 2% if the figure was a simple ratio. The negative aspect is, again, the loss of information and the inability to calculate with a string, without using further transformations. However, to process changes in percent the interface shall use a number.

As mentioned before, some services assemble two parameters to create a condensed one. A range instead of two separate high/low values is such an example. An Interface could either use a string or transform the range back into two decimal values. To be able to further work with the data, two separate numbers shall be provided via the API instead of a string.

The PreviousClose parameter is not provided by the service Markitondemand, but could, for example, either be derived by the last ‘Last’ value of the previous trading day, or be calculated by using the existing ‘Last’ and ‘Change’ values. The ‘Change’ value is calculated on the basis of the previous close value. It will therefore be used in the interface definition.

Several integer numbers use different integer types and in one case also a double. A larger integer number, such as Int32, shall be used, at least for the interface, in order to process large numbers. An integer that is defined as a double should not lose any information when converted to an integer since it should not have decimal places anyway.

There are different date formats in use. Compared with the SOAP message outputs, the dates are formatted differently. A solution for this issue could be the parsing of a date string back to a date type, followed by a transformation and (if desired), a conversion back to a string. The interface parameter would in this case be a string. If a service does not provide a date, it has to be clarified whether the date is the invocation time or a stock date. In the first case the time could be added as an expression, whereas in the second case this would be a technical challenge, since it could cause legal problems.

Several parameters are only provided by some services. They cannot be substituted in every service constellation since they do not exist. These parameters need to be excluded, if they cannot be derived correctly.

Operations of the API
To complete the approach and to supply stock information in various levels of detail, several different operations shall be provided by the API. The definition of operations is crucial for the XML-Schema and WSDL definition, which will then be used to describe messages in the WSDL file. Three different operations are described as follows:

“GetSimpleQuote”
The first operation provides a simple Stock Quote which will be returned for an identifier. An identifier can be a stock symbol as it is e.g. commonly used in the USA, an ISIN number or any other accepted identifier type. Accordingly there needs to be an

---

26 This also indicates that the output is intended to be delivered to a person and not a machine.
27 The ‘Change’ value derived from the ‘PreviousClose’ not from the ‘Open’ value.
input for an identifier and an input for an identifier type. Table 4.6 shows the in- and output parameters of the operation.

<table>
<thead>
<tr>
<th>Input:</th>
<th>Output:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Identifier (String)</td>
<td>• Date/Time</td>
</tr>
<tr>
<td>• Identifier type (String)</td>
<td>• Symbol</td>
</tr>
<tr>
<td>o Restrictions</td>
<td>• Quote</td>
</tr>
<tr>
<td>• Symbol</td>
<td></td>
</tr>
<tr>
<td>• ISIN</td>
<td></td>
</tr>
<tr>
<td>• ...</td>
<td></td>
</tr>
</tbody>
</table>

*Table 4.6: “GetSimpleQuote” of API*

“GetStockInformation”

The operation “getStockInformation” returns the parameters that can be provided by all stock quote services. Additional data, which is only provided by some services, could be provided as a string on an optional, non-binding basis. This could cause problems when consumers further process this information. The operations’ core output consists of the ten identified parameters mentioned above, as well as ‘dayRange’ and a ‘date’, whereas the date in this case only indicates the time of the request. Table 4.7 shows the in- and output parameters of the operation.

<table>
<thead>
<tr>
<th>Input:</th>
<th>Output:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Identifier (String)</td>
<td>• Symbol (String)</td>
</tr>
<tr>
<td>• Identifier type (String)</td>
<td>• Name (String)</td>
</tr>
<tr>
<td>o Restrictions</td>
<td>• Last (Double)</td>
</tr>
<tr>
<td>• Symbol</td>
<td>• Change (Double)</td>
</tr>
<tr>
<td>• ISIN</td>
<td>• ChangePercent (Double)</td>
</tr>
<tr>
<td>• ...</td>
<td>• Open (Double)</td>
</tr>
<tr>
<td></td>
<td>• High (Double)</td>
</tr>
<tr>
<td></td>
<td>• Low (Double)</td>
</tr>
<tr>
<td></td>
<td>• DayRange (String)</td>
</tr>
<tr>
<td></td>
<td>• PreviousClose (Double)</td>
</tr>
<tr>
<td></td>
<td>• Volume (Int32)</td>
</tr>
<tr>
<td></td>
<td>• Date (String)</td>
</tr>
</tbody>
</table>

*Table 4.7: Operation “GetStockInformation” of API*

“GetStockStatistics”

The operation “getStockStatistics” returns the same information as “getStockInformation”, as well as additional statistics of certain stocks. It requires the same input as “getStockInformation”. The additional statistics data is provided by the service FinancialDataProcessing, which is based on the data provided by the Bloomberg website. Table 4.8 shows the in- and output parameters of the operation.
Table 4.8: Operation “GetStockStatistics”

<table>
<thead>
<tr>
<th>Input:</th>
<th>Output:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Identifier (String)</td>
<td>- CurPERatioTTM(Double)</td>
</tr>
<tr>
<td>- Identifier type (String)</td>
<td>- EstimatedPE(Double)</td>
</tr>
<tr>
<td>o Restriction</td>
<td>- RelativePE_SPX(Double)</td>
</tr>
<tr>
<td>- Symbol</td>
<td>- EPS_TTM(Double)</td>
</tr>
<tr>
<td>- ISIN</td>
<td>- EstimatedEPS(Double)</td>
</tr>
<tr>
<td>- ...</td>
<td>- EstimatedPEGRatio(Double)</td>
</tr>
</tbody>
</table>

If a service provides a different set of parameters, and fails to provide the required data for this interface, then the missing data could be provided by an additional source, in order to maintain the same functionality and service quality. From a technical point of view, the BPEL process would have to integrate and invoke an additional service, which provides the required data and maps it to the output parameters of the composite service. However, this approach increases the complexity of the application in general and leads to more required effort for service integration and management. The supplement would furthermore not be reasonable if the supplementary service would also provide the remainder of data which the original service provides. Furthermore, if invoking multiple services instead of one, the non-functional qualities of the composite decrease while, in a pay-per-use scenario, the costs also increase.

Finally, if a substituting service provides a different output format for parameters that are standardized, and used as an identifier in consuming services, the data has to be transformed or translated in order to maintain that expected data format. Otherwise this would lead to unbearable integration issues on the consumer side.

4.4 Generalization of Identified Challenges

The following section summarizes and categorizes the challenges that were identified during the practical examination. It was notable that most identified challenges of the service interchange concerned the service integration. This, however, was also the main

---

28 In that case, the comprehending service could also replace the service completely.
subject of the practical examination. Other challenges also concerned information security.

**Challenge 1: Semantic Discovery and Integration**
The first significant challenge was to discover functionally equivalent or comparable services that might be candidates for a service interchange. The identification focused on the semantic interpretation and comparison of service operations and their in- and output parameters. The discovery of services was conducted manually by using metasearch engines and exploring service registries.

A service candidate had to meet a certain predefined minimum of functional conformity, in order to be classified as functionally interchangeable. This conformity could be assessed by the semantic match of operations and parameters of a service. In some cases missing parameters could be derived from others; however, this required a substantial understanding of the service’s business logic. It is also worth noting that some technical approaches to provide the same functionality were different. For example some messages contained assembled parameters, instead of separate parameters. Such different approaches had to be recognized and the data provided had to be interpreted accordingly.

The semantic information of a service was manually identified either by descriptive variable names (if provided), by analyzing example in- and output messages, or by viewing the service’s documentation. At least two of these methods were combined in order to avoid misinterpreting the semantics of a parameter or an operation. The developing environment is not capable of recognizing service semantics without semantic extensions in a service definition.

As a result, from a semantics perspective, the functional discovery and integration of services needs to be performed manually for each individual service. This is a great task and very time-consuming. Services that do not fulfill a predefined conformance to functionality cannot be interchanged.

**Challenge 2: Interface Differences**
Functionality is provided technically via the service interface, consequently the connection between different interfaces and the functionality of a service is closely related. This means that an interface has to deliver at least a subset of semantically equal parameters to be functionally related. Requirements to this subset have to be specified.

After functionally related service candidates for a service interchange had been identified, the service interfaces had to be technically compared to analyze the need of possible data manipulations and transformations. One had to determine how parameters and operations were named, which namespaces and prefixes were used, and which data types the parameters employed. The required technical information for these tasks was derived from the services’ interface description (WSDL) file and its XML-Schema definition.

As described above, parameters use different approaches on how to provide information. From a technical perspective, this results in different service interfaces. A service interface is already different if only the parameter names or namespaces are changed, since in BPEL every element is referenced by names. These differences, however, have to be overcome in order to exchange services, therefore adjustments in the form of transformations have to be conducted. In most cases parameters either need to be mapped to a subsequent service to meet its invocation requirements, or the parame-
ters must be mapped to a set of output parameters that serve a composite application/service interface. These have to provide a steady data output.29

**Challenge 3: Data Model/Encoding**

If a service is exchanged within a composite application and a subsequent service or the composite interface require a certain encoding format of data, the substituting service will need to provide this particular data too. If the data formats between the available output and required input differ, then the data will have to be transformed or translated.

Stock quotes can be identified in various accepted ways. The encoding of the stock identifier follows certain generally accepted standards such as “Symbol” or “ISIN”. The Xignite Global Quotes service resolves the disparity of various accepted standards through an additional input variable that allows the translation and definition of the standardized type used.

However, this solution only concerns this particular scenario. Services situated in other domains might use their own identifier names, without the availability of standards. Assuming a stock had no standardized identifier, the identifier encoding could be anything. The company name could be expressed differently, for example by using different name suffixes. The name could also be case sensitive, or different abbreviation could be used such as “ORCL” or “ORAC”. Furthermore, an artificial key, which is not related to the stock at all, could be used internally to identify the stock. Consequently the encoding of identifiers needs to be well known and the identifiers might need to be translated.

Other values can be provided differently too. A value indicating a change in percent for example can either deliver a ’10’ for 10% or a ’0.1’ which could also be interpreted as 0.1%. The consequences would involve severe calculation errors when the data is further possessed.

**Challenge 4: Different WS-* Specifications**

As described in the fundamentals chapter, WS-* standards are crucial for the implementation of important non-functional requirements.

WS-* standards, however, also have an impact on web service interoperability. Today over 100 WS-* specifications are available and it can be assumed that many specifications require the implementation of standards on both the requestor and the responder side. Furthermore, the implementation of standards needs to be varied in some scenarios. As a consequence, this can lead to interoperability problems. Services that implement particular WS-* standard cannot be exchanged without any limitations.

**Challenge 5: Security**

Within the experiments conducted above, security issues where a notable concern. In Scenario 3, for example, the authentication token and password were assigned to the SOAP header of the Xignite service without any encryption. This unencrypted approach of exchanging a message in general, and authenticating in particular, could be exploited by a “man-in-the-middle”. The abuse could result in direct damage and costs for the company, or act as an entry point for further attacks and hacks. In this situation the provided point-to-point HTTPS encryption might be sufficient, but in other scenarios, in-

---

29 There can also be a data output that is processed internally and does not need any mapping. Certain differences of numerical types may lead to a loss of data accuracy but may be tolerated by the environment.
cluding many intermediaries and more sensitive data, a more sophisticated approach is required [34].
5 Solution Approaches

The following chapter addresses each of the previously defined challenges and distributes them into four sub-chapters. The first four challenges all concerned integration issues. The discovery of web services is also included. The sub-chapter 5.1 covers the discovery and integration of web services. Challenge No. 5 regards security issues. Due to the high complexity and importance of the subject, the latter is divided into two parts. The first part is related to special SOA security requirements, and is presented in the sub-chapter 5.2. The second part covers the subject of trust in sub-chapter 5.3. Issues related to trust are especially significant in a cross-cloud environment. The last challenge included the interoperability between REST and SOAP web services in BPEL. This subject is covered in sub-chapter 5.4. It should be emphasised that this chapter covers solution approaches to ease the beginning of additional, future research. The chapter does not include complete solutions to the stated challenges and cannot substitute a literature review about these advanced topics.

5.1 Integration and Semantics

In the practical chapter of this thesis, several integration challenges concerning heterogeneous web services were identified. These challenges are now further discussed, and solution approaches to these challenges will be introduced. The subject of integration and interoperability is important, since it is a prerequisite for interchangeability.

The first challenge is related to the discovery of functionally equal services. During the examination, the process of service discovery had to be conducted manually using filtering methods and queries of a service registry. The discovery of services was very time-consuming. To find solution approaches that facilitate an automatic approach of service discovery, semantic methods, such as OWL-S, will be presented briefly. Furthermore, approaches to the dynamic discovery of web services will be introduced.

Other challenges concerned more technical, rather than functional, aspects of interoperability issues. The heterogeneity of web service interfaces leads to the need for many adjustments during the implementation process. A semantic approach could also be a solution for this issue. However, less sophisticated solution approaches will also be introduced. In addition, solution approaches concerning different data models will be given.

Because of the fact that semantic solutions are a driver for the development of comprehensive solution approaches to the integration of web services, less sophisticated solution approaches concerning the practical examination will be discussed first. Furthermore, semantic web services still remain somewhat uncommon.

5.1.1 Integration of heterogeneous Web Services

Besides the discovery of functionally equal web services, the integration of heterogeneous web service has been a major challenge during the practical examination. Different interfaces had to be integrated by altering almost all elements in the whole integrating BPEL process. The following section describes solution approaches to solve these challenges. Some drawbacks of these approaches are also introduced.

Different Interfaces

Different web service providers almost inevitably employ different interfaces. Even if the functionality of a service is almost equal, interfaces are still heterogenic. Even varying the name and namespace definitions within an interface description will lead to a
different interface. XSD elements, and consequently WSDL, messages can consist of different parameters and types.

Because WSDL and XSD elements are all referenced by names, adjustments have to be made within all BPEL activities in which they are included. This leads to the need for a manual integration of services, which can, however, not become a total recreation of a BPEL process\(^{30}\).

A result of the practical Scenario 1 was the fact that equal interfaces are technically interchangeable if they implement the same WS-* standards. This means that the creation of a common interface, which all interchangeable services employ, enables the interchangeability of web services by simply altering the endpoint reference.

**Common Interfaces**

Within the fifth scenario of chapter 4, a composite application was designed. The composite application was intended to provide an API to external users. The API’s interface was created as an intersection of commonly used stock quote interfaces, provided within the financial services domain. The creation process of that API’s interface can be compared to the creation of a common interface, the term common interface was also used in that relation. The selection of parameters for the common interface lead to the creation of a semantic profile, which was used to manually further validate the suitability of yet unknown service candidates for an exchange.

As services can be exchanged within BPEL, when their interfaces use equal names and namespaces, all services that employ the same common interface can be exchanged, simply by exchanging the URL within a Partner Link stub definition. Since interfaces behind a common interface are still unequal, differences need to be overcome by transformation. When using a common interface, only two transformations have to be specified: the transformation from a service’s interface to a common interface and vice versa [45]. The common interface can be realized in BPEL, just like the composite application in scenario 5, with the difference that only one service is “wrapped” within a composite application that exposes the common interface.

**The Wrapping of Services**

To implement the common interface, the service can be encapsulated in a composite application. A Mediator element can be used to route and transform messages from the common interface, through the Mediator, to the actual service interface and back. Simple disparities, like different names and namespace definitions, can be easily mapped in XSLT style sheets; complex transformations have to be conducted if services differ in a structural way [39].

Inside the composite application, all adjustments have to be performed to meet the requirements of the common interface. The translation of data encoding formats also has to be performed.

**Dynamic Binding of Web Services**

Encapsulated web services, that expose a common interface, can now be exchanged simply by altering the URL in the Partner Link definition within BPEL. Without further extensions, however, this needs to be done manually at design-time.

To implement a dynamic binding of web services at runtime, Oracle introduces methods to make BPEL dynamic in its BPEL Cookbook [58]. Reynolds and Wright also

\(^{30}\) This was the case in the practical examination.
propose this method [45] in their developers guide. The WS-* standard WS-addressing possesses an Endpoint Reference mechanism that implements an alternative for the static service endpoint binding within the service tag of the WSDL file. The standard provides an XSD type that can be used to define variables of the type EndpointReference. The variables can be used instead of a WSDL URL within a Partner Link. The decision of when to exchange a service URL within an EndpointReference variable could be made by using business rules that are enforced by a rule engine.

**ESB integration, an Alternative to BPEL**

Another alternative approach is the use of the Enterprise Service Bus (ESB) to integrate services into a process flow. The service bus tested within the scope of this work was the Oracle Service Bus (OSB). The Bus can be accessed using a web browser. The service integration is conducted and configured using a web environment that is abstracted from any code.

According to Oracle, the OSB provides a seamless integration of services. It possesses features to modify web service endpoints at runtime, and furthermore allows the validation of messages in order to evaluate their conformance to interface contracts. The OSB allows the abstraction of services by using proxy and business services. These two services are connected with pipelines that allow the invocation of further services and activities, such as the transformation of messages. The use of virtual service addresses moreover decouples the service invocation from the physical location of a service. A dynamic routing component allows the dynamic selection and integration of services from a service pool. Furthermore the OSB provides, for example, intelligent message brokering, service monitoring, administration functionalities, SLA compliance, and service lifecycle capabilities. Moreover, it integrates with other Oracle components [59], [60]. A drawback of the OSB, in Oracle SOA Suite 11g, is that it cannot be included into a composite application in JDeveloper [45]. One must consider which approach better suits a situation, OSB or a BPEL solution.

**Different Data Models**

The probability, that different service providers employ different data models that cause different data in- and outputs, is high. If services are exchanged within a composite application, the data in- and outputs however have to be steady. A common data model needs to be specified that uses standardized identifiers, such as a stock symbol, if available. Disparate data in- and outputs from different domains have to be mapped to that common data model using data mappings [61].

There are various possibilities to solve this issue of data integration: Data mappings can be implemented by using associative arrays as dictionaries, XSLT transformations, or graphical tools. Oracle provides Domain-Value Maps as a solution that allows one to map two vocabularies of different domains to each other. An example would be the mapping of the expressions VXO to Växjö, or KA to Karlsruhe [62].

Furthermore, sophisticated solution approaches allow the automatic mapping of data. The concepts use semantic vocabularies and languages, such as OWL and RDF to identify synonyms. The most advanced approaches, however, are data-driven mappings, which evaluate data sets to identify complex data relations. The identification can then also be a basis to derive complex rules for an automatic data transformation [63][61].

A drawback of the described method is that, without additional technologies, service transformations and data mappings to and from a common interface have to be created manually. Furthermore, services and their interfaces have to be known and integrated at design-time [51].
Different WS-* Standards
Web services are extendable. This includes extensions that concern the SOAP protocol and extensions that extend the WSDL interface description. It can be assumed that most standards have to be known to both parties on the server and client side. Since the protocols are extendable, there is no limit to additional WS-* standards; vendor specific standards are also in use. It can be assumed that the employment of different standards therefore inevitably leads to interoperability issues and, consequently, also to issues during service interchange.

It is highly recommended to build and use WS-I compliant services. Furthermore, the use of Meta-Languages like WS-Policy and WS-SecurityPolicy helps to enforce interoperability [34].

5.1.2 Semantic Approaches of Service Integration
As mentioned above, the discovery of functionally exchangeable web services was conducted manually in the preparation phase of the practical examination. The available services only possessed syntactical descriptions within their WSDL and XML-Schema definitions. Semantic information was not directly available and had to be manually derived from example service in- and output messages, or from service documentations on the service provider websites.

However, web services can be enriched with semantic information. The semantic Web extends the common web with additional methods to enable machines to interpret the semantic information of data. The semantic functionality becomes machine-readable [64]. OWL-S, the Web Ontology Language (OWL) for Services, provides methods to enrich web services with semantic annotations [65].

Semantic Web Services with OWL-S
Semantic technologies shall enable software agents to access Web resources by their content or, in terms of services, by their functionality. They furthermore shall enable the automatic discovery, selection, employment, composition and monitoring of Web-based services [66]. To make this possible, Semantic Web languages like OWL (Web Ontology Language) have been developed. OWL enables the specification of concepts to represent a knowledge base of a particular domain. These ontologies are required for software agents to interpret the applied semantic information [66][67].

The Web Ontology Language for Services (OWL-S) is based on OWL and enables the enrichment of web service descriptions with semantic information. The goal of OWL-S is to make semantic characteristics of a web service machine-readable and allow entities, such as software agents, to dynamically discover and compare web services according to their functionality. Software agents are able to automatically read the description of web services, including in- and outputs parameters, invocation preconditions etc. [66]. Martin defines the high-level principles of OWL-S as follows: “[…] (1) to provide a general-purpose representational framework in which to describe web services; (2) to support automation of service management and use by software agents; (3) to build, in an integral fashion, on existing web service standards and existing Semantic Web standards; and (4) to be comprehensive enough to support the entire lifecycle of service tasks.” [49, p.246]

The setup of OWL-S
The structure of an OWL-S description consists of different parts. A ServiceProfile describes the functionality of a service. The service profile is then advertised at an OWL-S matchmaker entity. A Service Model then describes how a client can interact with a service. After that the model is published by the service provider via the web. A Service-
**Grounding** specifies the technical details to map a semantic OWL-S service to an actual web service. All three parts can be implemented using elements of the OWL-S vocabulary, in connection with RDF [68] and RDFS [69]. Examples of the implementation can be viewed at OWL-S specification [66]. The parts are described further in the following sections. The following Figure 5.1 shows a top level overview.

![Figure 5.1: OWL-S High Level Overview; Source: [66]](image)

**The Service Profile**

The service profile itself consists of different facets: A functional description defines the classes: *Parameters, Input, Output, Preconditions, and Results* of a service. Furthermore, relations between classes are specified. The in- and output classes are aggregations of the parameters specified in the parameter class. Preconditions define constraints that have to be satisfied.

A *Category* class, which is also part of the service profile, relates the service to a predefined business specific *ontology* or *business taxonomy*. URIs that are used, for example in the specifications of in- and output messages, refer to elements in a related ontology. The semantics of parameters can consequently be interpreted. Since an ontology provides a linked knowledge base [67], syntactical failures can be overcome.

Finally, a facet concerning non-functional properties is part of the profile to distinguish between differences of QoS.

**The process model sub-ontology**

After the basic service discovery was conducted, the process model provides a detailed model to further determine the functionality of the service. It is important to determine whether a service is functionally suitable, according to the predefined requirements of a consumer. In addition, one must consider, which constraints have to be satisfied in order to invoke the service [49].

The process model encompasses the definition of *atomic processes, simple processes, and composite processes*. An atomic process represents a single service request-response operation that can be further orchestrated to a simple process. A simple process, however, is only a concept and cannot be executed. A composite process can be compared to a composite application such as the one created in chapter 4. All processes are children of the process model. Inputs, outputs, preconditions etc. are specified via the atomic process. Again the code of OWL-S is implemented in a machine-readable style using XML-based code with RDF and OWL vocabulary [49]. The following Figure 5.2 shows the structure of a service model.
The ServiceGrounding sub ontology

The ServiceGrounding sub-ontology technically specifies the way how information between a service requestor and a service consumer is exchanged. It furthermore defines how the abstract semantic information specified in the atomic process’s model is mapped to the actual web service [49].

Every atomic process specified in OWL-S is mapped to a WSDL operation of the target web service. The atomic process’s in- and output variables are mapped to the in- and output elements defined within the WSDL operation [49].

The grounding furthermore describes how the in- and output messages of the semantic OWL-S specification are translated to the in- and output parameters of the WSDL file and vice versa. The grounding specification can be included at runtime to allow a dynamic invocation of web services [49].

The following image (Figure 5.3) illustrates the mapping and translation between OWL-S and WSDL. Both actions take place at the two bidirectional arrows, marked bold in the center.
The following Figure 5.4 shows a typical semantic web services interaction model. The model shows and describes how OWL-S could be employed to realize the discovery of a functionally suitable web service for a service interchange.

A service provider generates a semantic service profile and a process model for its web service. The process model is registered at an “OWL-S Matchmaker” entity. The process model is published online.

A service consumer wants to consume a service, which provides a particular functionality. It creates a service profile and queries the OWL-S Matchmaker entity. The Matchmaker entity matches the profiles to discover a suitable service and returns a set of candidates to the service requestor.

The service requestor selects a web service. During this selection process profiles can eventually differ and a number of trade-offs have to evaluated.

The service requestor requests the service model that contains the definition of the process models and matches the information with the own goals and preferences.

The service requester invokes the web service. The grounding translates the semantic information to the actual web service interface and invokes the service.

The service provider receives the message and decides if it can perform the request. It then replies to the message. If it cannot process the request an error message is sent.

The service consumer’s grounding sub-ontology translates the incoming message back to its abstract semantic form. The response can now be interpreted by the service requestor.

Paulraj et al. [70] and Brogi et al. [71], amongst others propose approaches of service discovery and composition using OWL-S.

### 5.1.3 Comprehensive Approaches to the Integration and Discovery of Web Services

There are more sophisticated and comprehensive approaches for tackling the challenges related to the semantic and technical integration of web services. Two projects that work on such approaches are concisely introduced within this section, to give a quick insight of what is, may, or will be possible in the future of Service Oriented Computing.
Intelligent business processes composition based on multi-agent systems

García et al. [50] proposes (IPASCI), a comprehensive model for the agile, and automatic creation of business processes in cloud computing environments. The realization of business processes shall be triggered by an abstract specification of the business process in a text format. The model includes the use of semantic methods, such as ontologies, to facilitate the automatic discovery of web services. Furthermore, the proposed method includes the automatic composition of web services in BPEL to create higher level services, and the automatic invocation of web services. The functionality is facilitated by a multi-Agent system. The work furthermore encompasses a casestudy that includes the implementation of the model as a tool.

A Dynamic Composition and Stubless Invocation Approach for Information-Providing Services

The work proposes a method to a stub-less specification and execution of services that is based on the Graphplan algorithm. Paganelli et al. [51] states that a dynamic binding and invocation of web services needs to be possible at runtime, even when using services that were unknown to the invoking entity before deployment. Several solution approaches might allow such a dynamic binding, but only while using a vendor specific methodology. To invoke a web service in a process, a stub entity needs to be present as a proxy. The stub can, for example, be a java class, depending on the implementation, and is, in its BPEL representation, commonly a Partner Link. The entire process engine requires knowledge about a service and about its interface at the design stage. Paganelli et al. therefore proposes a method that does not use pre-compiled service-related classes. In contrast the method uses the Graphplan\textsuperscript{31} algorithm to specify service plans at runtime. A stub-less service execution is realized by identifying and exploiting semantic and structural characteristics of the service description.

It can be stated that the trends of SOC related service integration is going towards more agile and automated approaches, that are commonly based on semantic methods like ontologies and semantic languages, such as RDF, OWL and, for web services in particular, OWL-S. Furthermore, data driven methods represent a sophisticated approach for the development of automated integration, including automatic transformations and mappings.

5.2 SOA Security

SOA applications are exposed to the same threats as traditional distributed applications. Furthermore, they rely a lot on external service providers, especially in a cross-cloud environment. Just as in traditional applications, information security goals have to be secured. This includes the assurance of confidentiality, integrity, and non-repudiation goals, and also the implementation of mechanisms for authentication and authorization. There are various security standards available to approach the fulfillment of these goals, however different situations require different measures, and therefore it is quite likely that developers also implement different approaches to solve a problem. This, however, may lead to interoperability problems and, consequently, to interchangeability problems.

The following subchapter is based largely on the book SOA Security by Kanneganti et al. [34].

\textsuperscript{31} The Graphplan algorithm is a planning algorithm that was proposed by Blum and Furst in 1997 [72].
5.2.1 Basic SOA Security

According to Kanneganti [34] there are three approaches to manage SOA security: Message-level security, Security as a service, and Policy-driven security.

Message-level Security is intended to enable end-to-end security on the message level. To achieve this task the web service standard WS-Security can be used. The standard will be introduced later in this subchapter. Security as a service employs an infrastructure web service that deals with security. It can be implemented in the ESB and processes all security related tasks. SAML, WS-Trust etc. are standards to support such a service, both will also be explained later. A policy-driven security approach allows the separation of the service logic and the security implementation by providing a framework to assert security standards. An advantage of such a separation is that the implementation of security standards can be conducted by security specialists. Furthermore, policy-driven security fosters interoperability as it declares which services employ which standards. It enables different services to compare their policies and negotiate an intersection, on which both parties can agree on. Finally, it facilitates a consistent application of web service standards. WS-Policy and WS-SecurityPolicy are two standards to define policies [34], [46].

Which one of those methods is employed depends on the scenario: Kanneganti [34] suggests that the security of a single service within a web service toolkit should be ensured using pre-crafted policies offered by the toolkit. Oracle’s Enterprise Manager, for example, offers such policies. If pre-crafted policies are not available, a self-defined policy should be created. The policy-driven approach also includes the employment of suitable security WS-* extensions to enable message-level security. However, to secure a large number of services within an enterprise, it is suggested to augment the ESB with a security infrastructure, to implement the Security as a Service approach.

5.2.2 Authentication and Authorization

As observed in the practical examination, a user needs to authenticate in order to use an external web service. In the practical examination, the Username token and password were either submitted unencrypted or using HTTPS. As previously stated, both approaches are not optimal, since the first one does not use encryption at all, and the second is only encrypted until it reaches the external service’s endpoint in a synchronous invocation, or a mediator service.

Authentication on different Layers

Authentication of SOAP services is often conducted via the transport layer using HTTPS. This way is not recommended, since the method does not provide end-to-end security. Within the infrastructure of a service provider the username, password data is available in plain text. Furthermore many web service software suites allow basic HTTP authentication, also Oracle’s Enterprise Manager. The authentication is conducted by the request response of a username and password via the HTTP header. The authentication could also be implemented in the actual implementation of the service. This, however, would create further integration issues, because the authentication could be implemented completely individually, using various security mechanisms. Consequently, the best way to implement security mechanisms is in the protocol itself [34].

WS-Security

The SOAP protocol’s header is extendable. As mentioned in the fundamentals chapter, the extensibility of SOAP also leads to many different WS-* specifications, this also includes security extensions.
A standardized header extension to SOAP is the security standard WS-Security. The standard allows the inclusion of security claims and the employment of further standards such as XML-Encryption. The following code snippet (Figure 5.5) shows an excerpt of a SOAP message that is extended by WS-Security claims:

```xml
<soapenv:Envelope …>
  <soapenv:Header>
    <wsse:Security…>
      <wsse:UsernameToken wsu:Id="1">
        <wsse:Username>
          <xenc:EncryptedData>…</xenc:EncryptedData>
        </wsse:Username>
        <wsse:Password>
          <xenc:EncryptedData>…</xenc:EncryptedData>
        </wsse:Password>
      </wsse:UsernameToken>
      <wsse:Security>
        </soapenv:Header>
      <soapenv:Body>…
    </soapenv:Body>
  </soapenv:Envelope>

Figure 5.5: XML-Encryption; Based on: [34]
```

The WS-Security specification provides a standardized number of Tokens that solve a certain purpose. An example can be seen in the code snippet above. The snippet contains the UsernameToken that allows the transmission of username and password in the message header. This token, and further tokens, are specified in the WS-Security specifications that are available in various versions [WS-Security spec]. The specification also describes how to implement WS-Security and gives examples and explanations. All tokens are additionally explained in the token profile files provided at OASIS\(^{32}\). This includes, for example, X.509 or SAML token profiles. WS-Security furthermore also allows the use of self-defined security tokens. It is, however, not recommended to use these as they could be not understood by other services. If a WS-Security claim is not understood by the message receiver, a SOAP fault message is sent, that reports the cause of failure. WS-Security claims or in general SOAP header extensions are processed by special implemented software handlers [34].

5.2.3 Confidentiality

Data confidentiality can usually be provided using SSL/TLS on the transport layer. This ensures a point-to-point encryption of the whole communication on the wire between sender and receiver. Due to the fact that SOA normally involves multiple intermediaries, a point-to-point encryption like in HTTPS is not enough. If a mediator can access all information that is not addressed to that mediator, this is a serious security issue. An end-to-end security implementation is needed [34].

As in general information security, confidentiality can be ensured using symmetric and asymmetric encryption. In addition, encryption in SOA needs to meet further requirements. It needs to be selective (different parts encrypted for different clients), it needs to respect the syntax of SOAP documents (SOAP documents need to be identified

\(^{32}\) https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss
as such), and it needs to provide metadata do explain what parts have been encrypted and how. For symmetric encryption the XML-Encryption standard is used. The standard uses an EncryptedKey and an EncryptedData element. The EncryptedData element contains the encrypted data itself and further information about the encryption algorithm and a message ID. The EncryptedKey element contains all information about the key such as the method it has been encrypted with, plus a certificate to guarantee its authenticity. The specification of XML-Encryption describes the standard in detail [73]. It must be noted that the standard has been broken once and was fixed by the W3C [74].

Different encryption algorithms cause interoperability issues if one style is not known to another party. Furthermore, too much encryption can block the monitoring functions of a SOA infrastructure [34].

5.2.4 Integrity and Non-Repudiation
Data integrity is important to ensure that a message has not been manipulated, for example, by a “man-in-the-middle”. Furthermore, non-repudiation needs to be established to ensure that a sender is not able to deny having sent a message. Both goals can usually be ensured with digital signatures. Because transport-layer security is not appropriate to provide SOA security, a different solution has to be implemented in SOA [34].

XML-Signature can be used as a method to sign XML messages. The message is digested by using a hash-function and encrypted using the private key of the sender. A certified public key is openly available. It can be used to decrypt the digital signature. The receiver can digest the transmitted message himself using the same hash function and compare the two hash values. The public’s key authenticity can be certified using X.509 certificates over WS-Security. The use is specified in the X.509 Certificate Token Profile [75], [34].

A varying layout of an XML file has no impact on its parsing. However, using a hash function on two differently formatted files will result in two different hash values. Consequently, XML files have to be transformed to a common, canonical form prior to being digested by a hash function. A transformation standard needs to be used to transform such an XML file, in order to prevent interoperability problems [34].

5.2.5 Password Digest – One Time Passwords
Even if XML-Encryption allows the authentication via the SOAP header in an encrypted way, there is still the risk of a “man-in-the-middle” attack. The latter could, for example, use a replay attack to authenticate himself with the service provider [34].

Password Digest offers a way to implement a one-time password. A password is normally either transmitted as plain text or as a hash value. The hash value never changes if the password does not change. An attacker could simply intercept an authentication message and use it for his own authentication. Password Digest uses a function that adds a Nonce, a number used only once, and a timestamp to the password, to generate a triple that is then hashed by a hash function such as SHA-1. The Nonce makes the password unique every time it’s used. The timestamp helps the server to block every password used within a certain time range. The server saves every password in that range and compares it to incoming passwords. Replay attacks can thus be prevented. In order to not be required to save all passwords ever sent, the range is changed periodically after a predefined amount of time. Because a timestamp is used, to implement a Password Digest the system times of services have to be close, and different time zones, daylight savings time etc. have to be taken into account [34].
5.2.6 The Authentication System Kerberos

A service provider can steal the username or password credentials of a service consumer and try to use them with other service providers. Kerberos provides a system of preventing this scenario by providing individual tickets to registered service providers [34].

To ensure SOA security, an authentication system needs to satisfy various requirements. Kerberos does this: Kerberos allows the collection of identities in different ways; it does not require that a shared secret such as a password needs to be exchanged over a network. It implements Single Sign On to multiple services and facilitates a delegable authentication. Not only a client has to authenticate himself, also a server that provides a service - the authentication is mutual. Finally Kerberos provides protection against replay attacks [34].

Kerberos enables the user to authenticate in various applications that do not trust each other. Hostile service providers cannot abuse credentials any more, since they are not exchanged. Kerberos uses a central authority that authenticates with users and services. A user only has to authenticate once per session with an authentication authority. He can request a ticket to use different service providers. Tickets are sent encrypted and asymmetrically using signatures and certificates to guarantee the authenticity. The standard can be used in connection with WS-Security as described in the Kerberos Token Profile [76], [34].

The detailed Kerberos process is not further explained in the scope of this thesis. There are several web sites and books that explain Kerberos. The reader is referred to a collection of Papers and Documentation provided by the Massachusetts Institute of Technology (MIT)33.

5.2.7 Security as a Service

Security can be implemented as a service. There are different ways to invoke a security service. It can be invoked by the receiving service’s endpoint, by the sending endpoint, or by both endpoints. Furthermore messages can be routed via the security service, either explicitly or transparently, with help of a smart network device. Protocols to facilitate these tasks are SAML, WS-Trust, WS-Addressing, Transparent routing and AON [34].

SAML

The Security Assertion Markup Language (SAML) allows a security entity, such as an identity provider, to create security assertions that other entities can rely on. SAML aims to enable Single-Sign-On (SSO) and Identity Federation [34].

Assertions in SAML can contain one to multiple security statements. There are three different kinds of security statements. An Authentication statement states that a user was authenticated with an authority at a certain time, an Attribute statement states that a subject is related to a particular attribute, and an Authorization Decision statement asserts decisions applied on a subject, such as if a subject has been granted or denied the access to certain resources. The structure of a SAML assertion includes information about the issuer, information about the subject, a signature, the conditions and the actual authentication statement [34].

The following figures (Figure 5.6 and Figure 5.7) show the principle of Single Sign On in the context of SAML and the content of a SAML assertion.

33 http://web.mit.edu/kerberos/papers.html
WS-Trust
WS-Trust allows a trusted and brokered exchange of security tokens. This includes the issuing, validation, renewal, and cancellation of security tokens [WS-Trust Spec]. Security tokens can consist of the SAML assertions that were described above. WS-Trust describes an interface for a security service that is further called a Security Token Service (STS) [34]. If a client wants to access a web service it authenticates with the STS. The service endpoint to be invoked also authenticates with the STS. If both parties are authenticated, a security token is exchanged. The standard is further examined in the standard’s specification.

SAML, WS-Trust and also Kerberos introduce principles that could be used as a basis to implement an identity management system in a cross-cloud environment. According to Kanneganti [34] all provide the requirements for authentication systems. As other approaches are to name OpenID, OAuth etc. Further information can be accessed at the web site of OpinID [34]. Identity management however will not be introduced in the scope of this thesis.

---

34 http://openid.net/developers/specs/
5.2.8 Security Policies

Security solutions can be implemented using different security standards. The employment of such standards can cause interoperability issues when different standards are employed. Even if the same standards are used, interoperability issues are likely to occur, because more complex security standards allow the choice between a set of possible approaches to deal with different security issues. For example, if standards use different algorithms to encrypt, sign, or authenticate, and if these standards may not be supported by a counter party, the latter cannot understand the standard. Consequently a service cannot be used [34].

WS-I developed the Basic Security Profiles to limit the choices of implementation for developers. The profile provides recommendations and best practices for the implementation of SOA security related solutions and can be accessed on the WS-I websites [79]. However, even the WS-I Basic Security Profiles cannot comprehensively ensure the interoperability of security standards, consequently further standards like WS-Policy and WS-SecurityPolicy were introduced. These describe the implementation of security solutions and furthermore allow the enforcement of standards [34].

Security policies are a good approach to specify the employment of security standards. The policies enable security experts to independently work on security subjects while application developers can focus on the implementation of the business logic. For simple scenarios, there are packaged policies, which are usually provided by the developing environment suite. Oracle’s Enterprise Manager, for example, provides a set of predefined policies that can be applied to a service. Policies can be used internally, to ensure a consistent application of security standards; they can also be employed to describe requirements to external services. This can help to ensure interoperability between services of different service providers, since policies can be compared and negotiated to specify a policy accepted by all parties involved. The software can then use further standards to fetch policies of other service providers. The WS-MetadataExchange protocol is used to request a policy from a service provider, the standard WS-PolicyAttachment defies how policies are attached to WSDL, UDDI etc. [34].

5.2.9 WS-SecurityPolicy

WS-SecurityPolicy [80] provides a framework to express security related assertions. This is important to further enforce interoperability between different services. The assertions can encompass requirements, capabilities, and constraints of security implementations for standards, such as WS-Security, WS-SecureConversation, and WS-Trust. Policy assertions can be set at endpoint level, per operation, at the message level, or they can be nested into other assertions [34].

Security Assertions for the Endpoint

The task of assertions for the endpoint is to describe security assertions that are suitable for a set of commonly used variations of standards. Assertions concerning the endpoint can further be classified into three different security patterns. These describe how security patterns can be bound to the endpoints between sender and receiver in different ways, for example using a transport-based way, symmetrically, and asymmetrically. For each of these security usage patterns, WS-SecurityPolicy defines assertions. This includes information about the type of tokens required, about used encryption algorithms or the strength of keys. The layout of WS-Security header elements, the order of encrypting, the signing and further aspects, like the management of timestamps, can also be asserted. Moreover, WS-SecurityPolicy specifies an additional assertion about the
conformance between WS-Security and WS-Trust, and also about supporting tokens that are required to specify further usage patterns [34].

**Security Assertions for Messages and Operations**

WS-SecurityPolicy provides further assertions that concern message layer security. These assertions include statements about several ways to sign or encrypt messages, such as if the message as a whole should be signed or just specific parts, if the whole SOAP body is encrypted or if every element is allowed selective encryption etc. Moreover, required elements can be specified using XPath. Operations can be asserted with support tokens [34].

### 5.3 Trust related SOA Security

The last sub-chapter introduced special SOA related security requirements plus several security related WS-* specifications. However, in a cross-cloud SOA environment, basic SOA security is still not sufficient to guarantee information security [81].

A relevant aspect is the fact that actual service implementations of external services are not transparent to service consumers. Web services only provide an endpoint, often referred to as a “black-box”. Internally, external services may be themselves composed of further services that are provided by additional external service providers. These services could be further composed etc. This may lead to a chain of service invocations that might include multiple unknown external service providers. Some of these service providers could be deceitful and manipulative and consequently cause security breaches. As a consequence according to Li [82], in a cross-cloud SOA environment, particularly the security of unknown external service providers needs to be evaluated. Employed services need to be trustworthy in order to deal with sensitive data. A trust evaluation system needs to be introduced to identify security risks.

Trust, in the context of web services, describes the willingness of a service entity to believe in another service entity. To measure trust, a concept needs to be developed to quantify a level of trust, this is rather complex [83][84]. Armbrust et al. [85] and Buyya et al. [86] state that the assessment and management of trust especially within a cross-cloud environment is very challenging. Sheth et al. [87] proposed a framework in which reliable “trust management service nodes” were employed to manage and maintain the trust of cloud services. The trust services discover and evaluate external services and collect feedback from the service consumers. Other service consumers can then request trust figures from the trust management services. Various solutions to implement such a trust management have been proposed. Some of them are introduced in the following sub-section.

#### 5.3.1 Different Approaches of Trust Evaluation

There are various different approaches to evaluate trust. Proposals include reputation, network, certification-based trust evaluation approaches and more.

A very common approach of trust evaluation is a reputation-based approach. A reputation-based trust evaluation facilitates service selection for clients in a comprehensive market of services [82]. A reputation can be expressed by adding a trust score to a service provider. Services are evaluated by service consumers that interacted with a service before. The consumers transmit their experiences with particular services to a trust management authority. The authority manages the creation of a reputation score, which can also be called a trust level of services [88]. The trust management authority is responsible for calculating, aggregating and managing the trust level of services [89]. It provides information about the trust ratings of particular services to the authority’s cus-
customers. Trust figures can be presented online or as a response on particular service queries [88].

There are various approaches to evaluate the reputation of a service that indicates a level of trust. Some approaches are presented in the following section:

One method to measure the reputation of a service is based on the comparison between the advertised non-functional properties and the actual properties that are measured by the service requester. A service is only reputable if the real quality is representative of the advertised level of quality [90]. Another approach by Wang [91] evaluates a set of various metrics that are further aggregated to finally rank a service according to its reputation value against other services. Conner [92] comprehends that approach by adding a method for service consumers that allows them to enter preferences according to their individual trust requirements.

Li et al. [82] summarize the fact that most trust evaluation models use a single trust level figure, which is commonly referred to as the “Final Trust Level” (FTL) for every service. The figure is calculated as an aggregation of average trust values over a specified period of time. It is meant to indicate the trustworthiness of a service in the future, using a past value. A single value to access a trust level however is not optimal according to [88], because it only shows the actual value and does not include any development figures. Consequently, [88] proposes the extension of the approach by using a trust vector instead of a single value. The vector contains three numbers, the FTL and in addition a “Service Trust Trend” (STT) to measure the dynamics of a trust value, plus the “Service Performance Consistency Level” (SPCL), which indicates the variation of the STT value.

5.3.2 A Trust Model for Service Certification

Another approach to maintain trust and information security within a cross-cloud SOA environment is to implement a certification approach. According to Anisetti [93], the implementation of a certification-based service evaluation system has an impact on the traditional trust evaluation systems as it further complements the selection process of services, by adding an additional level of rating.

The security level of a service is assessed by a certification authority (CA) that creates, collects, and validates security assertions. The security assertions are further assessed and supported with evidence and then they are certified. Evidence can be derived from the results of actual security tests [94][95][96].

The CA itself needs to be a trusted entity; therefore it needs to be accredited by further authorities in order to be trustworthy. The trust of a client in a service consequently also depends on the CA that assessed the service, provided certificates can be traced back to a CA by a service consumer. Furthermore, the method of assessment can vary. The trust of a service consumer in a service provider is also affected by the method of evaluation and of the evidence available to assure the certification. The evidence is also specified by assertions. A CA uses a predefined certification schema that ensures a reliable service evaluation under a clearly defined quality system. The schemas also document how a service was validated [93].

The validation of a service’s security characteristics can be processed on the focus of the security functionality itself, but also on the development process of a service. A service provider can request the certification of its services at a certification authority (CA). The service provider needs to provide all the required information to the authority, which then starts with the evaluation process. The process also includes the collection of evidence. The service provider benefits from the certification as a selling argument. Furthermore, a certification may help to improve the service as it can identify weak spots. The service gets certified by the CA according to its results. A service con-
sumer can now query for certified services and evaluate the certificates to finally choose a suitable service. The client can define preferences for properties, models, and evidence to find suitable services for his needs. Service assertions can be matched and ranked [82].

5.3.3 End-to-End Security Approach

The following approach of Azarmi et al. [97] goes further than the other introduced approaches. It combines several aspects of other approaches, such as reputation and certification, but intends to extend the approach of certification by adding a software module to the ESB of a service provider. Azarmi states that an illegal invocation of services within a composed service cannot be detected - within or without a trusted domain. Consequently, services can still manipulate data. The approach shall increase the selection of trusted and secure services.

The work introduces a framework that consists of a “Taint Analysis” (TA) software module and a “Trust Broker” (TB). The software module recognizes nested service invocations and reports them to the broker. The broker monitors and manages all certified services.

The TA module enables the system to track all service invocations within BPEL processes. It reports invocations to the TB and detects misbehavior of external services. The Trust Broker evaluates the appropriateness of service invocations. It furthermore monitors and manages all deployed TA modules.

A service provider is labelled “certified” if it deploys a TA module within its ESB and furthermore is compliant to a set of predefined WS-* specifications. If a service provider is not certified, it is rated as in a reputation-based trust evaluation approach. This reputation-based evaluation approach includes the history of service interactions, the support of various WS-* standards, and the trust levels of employed services within the service provider’s composition.

In practice a client looks for a service by querying a registry. The results are sent to the Trust Broker, which categorizes the results into various levels of trust. The client chooses a certified service that is situated in the trusted domain and registers the service for monitoring by the Trust Broker. If the monitored service invokes another service illegally, the Taint Analysis module intercepts the invocation data and reports it to the TB. The service response is sent back to the client.

5.4 REST and BPEL

During the web service market research it was interesting to note, that a lot of services, especially user-centric services, were RESTful services using JSON. It is also known that RESTful services cannot be integrated into a BPEL sequence [42] without additional measures. Determining how a RESTful service can be included into a BPEL orchestration is thus a useful investigation.

Pautasso [98] proposed a solution approach to this issue in 2009. He proposed the extension of BPEL with four new activities, corresponding to the four HTTP verbs that are employed by RESTful services. Consequently, the activities should be named <get> <put> <post> <delete> and be further extended by an attribute to address the URI of a particular resource, as well as two more attributes, to specify variables that contain the message payloads of the request and response messages. The use of a variable in the uri-attribute should facilitate a dynamic late binding of unknown RESTful services in connection with a decision logic. The activities should furthermore contain elements to perform fault handling, as well as header information.

Also in 2009, Peng et al. [42] introduced a framework “REST2SOAP” to encapsulate RESTful services within a SOAP service. The method uses the Web Application De-
scription Language (WADL [99]) to describe a RESTful web service, the XML User Interface Language (XUL [100]), and the web service standard technologies BPEL and WSDL.

A RESTful service provider needs to provide a WADL file for his RESTful service. The service provider can create such a file using a WADL Designer tool that is part of the framework. To access the framework, the service provider connects to the module “RESTful Service Publisher” that manages all further activities.

The sub-module WADL convertor validates an applied WADL file, and creates a WADL content tree that acts as the internal representation of the service. The “Proxy Generator” then generates a proxy class in the Java back-end and maps operations to HTTP verbs and URIs. Furthermore, XML data types are mapped to Java data types. Finally the “REST2SOAP Deployment Assistant” generates a WSDL file to describe the SOAP service. The proposal uses the tool WADL2Java which is available online; the service is deployed on Apache Axis35. Figure 5.8 shows the REST2SOAP framework.

Oracle JDeveloper facilitates the integration of RESTful services by using an HTTP Adapter. The adapter is new with SOA Suite version 11g and can be configured using a Configuration Wizard. The wizard requires a schema of the RESTful service or sample data e.g. as Comma Separated Values (CSV) to automatically generate a schema. Furthermore the HTTP verb, URL and operation type have to be specified [39].

5.5 Evaluation of Hypotheses

Finally, in this section (Table 5.1), the predefined hypotheses of chapter 3 are evaluated against the results of the practical and theoretical research.

<table>
<thead>
<tr>
<th>Hypothesis 1 (Equal Interface)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hyp0thesis 1.1 (Technical View):</strong> Different implementations of the same service interface are always technically interchangeable if they apply the same standards.</td>
</tr>
<tr>
<td>This statement is true, because if the abstract part of an interface is equal to another interface’s abstract parts, all referenced elements are still valid if the interface is ex-</td>
</tr>
</tbody>
</table>

changed. The service address in the concrete part of the interface, however, can differ and point to a different service endpoint.

**Hypothesis 1.2 (Service Functionality):** Different implementations of the same service interface cannot always be interchanged functionally. Services which implement the same interface can implement a different functionality and behavior. Services are then not interchangeable.

**True**, because a process requires a particular functionality from a service. If the service does not provide this functionality, it consequently cannot be used to achieve the goal a process should achieve.

**Hypothesis 1.3 (Data Model):** Services which implement the same interface, and provide the same functionality, can differ according to their data back-end. This includes the encoding of data, calling conditions and output data. Integration challenges will occur.

This hypothesis is also **true**, concerning the field of data integration. Different data models have to be integrated, especially if a common data model is required. Data mappings need to be applied.

**Hypothesis 2 (Different Interface Technically):** Functionally equivalent services that employ different service interfaces need to be technically adjusted in order to be interchangeable. Examples of these differences include disparate parameters, data types or solution approaches.

The statement is **true**, different schemas lead to a different data structure. To be interoperable, schemas need to be mapped in schema mappings.

**Hypotheses 3 (Composite Applications):** Composed Services need to provide a steady interface and steady data to the outside world.

A **true** hypothesis, because external service users require a steady data in- and output, in order to integrate the service into their own environment. If a service changes its in- and output data structure and content, a service consumer would have to make adjustments himself to maintain interoperability.

**Hypothesis 4 (WS-* Standards):** Web services have to fulfill the same standards in order to be interoperable.

This hypothesis cannot be tested, because of the high number and complexity of WS-* specifications. There might be standards that do not require a counterpart, without affecting the interoperability of services. Service developers should follow the WS-I profiles.

**Hypothesis 5 (Semantics):** The pure functional content and logic of a web service cannot be identified automatically by a machine without semantic annotations. Services have to be searched individually by hand to guarantee interoperability.

This statement is **true**, because ordinary schema and interface definition files only provide syntactical information about the service.

Hypothesis 6: (Security and Trust)

**Hypothesis 6.1:** The service interchange within a cross-cloud SOA environment leads to additional security challenges.

A **true** statement, because SOA includes many mediators and parties that “traditional” distributed systems do not include. SOA security consequently requires measures that provide, for example, end-to-end security including special mechanisms, such as selec-
Hypothesis 6.2: It cannot be guaranteed that a service provider implements the same security measures and standards within its compositions.

The hypothesis is **true with some limitations**. To evaluate external services only methods to evaluate the trust level of a service are available. More sophisticated approaches require the active cooperation of external service providers.

**Hypothesis 7 (SOAP and REST):** SOAP based web services and RESTful web services are not combinable without special measures in a BPEL process.

This is a **true** statement, because RESTful services can only be integrated into a BPEL process with the help of tools or BPEL extensions. IDEs, such as JDeveloper, provide adapters to integrate disparate technologies. The HTTP Adapter in JDeveloper can, for example, be used to integrate a RESTful service into the SCA.

Table 5.1: Evaluation of Hypotheses
6 Conclusion

Within the scope of this work, challenges of service interchange in a cross-cloud SOA environment were identified. In the last chapter, the identified challenges were further examined to present solution approaches to the reader. In addition, the previously defined hypotheses were evaluated. The following chapter will now appraise and conclude the insights of this work in relation to the general objectives.

6.1 Review

The subject and research context were introduced in the first chapter. A fundamentals chapter presented the principle technologies that are crucial for a basic understanding of Service Oriented Computing, Service Oriented Architectures, web services and the practical examination. The latter was prepared in the conceptual approach in chapter 3; furthermore, hypotheses were specified as a guideline for the examination approach. The practical examination was covered in chapter 4. It showed how a cross-cloud SOA could possibly be implemented in practice, by using BPEL in Oracle JDeveloper. The examination moreover examined the service interchange within a BPEL process embedded in a stock quote scenario. Various testing scenarios were examined to cover all facets concerning the interchange of web services. In the end of chapter 4, identified challenges were categorized. These challenges were addressed in chapter 5, in order to find solution approaches. The approaches are based on the practical examination and furthermore on a theoretical investigation. This investigation gave further insights into current projects and future scenarios of SOC.

6.2 Evaluation of Objectives

The goal of this work was to identify challenges during the interchange of services within a cross-cloud SOA environment. The thesis should furthermore generate a foundation for future academic research. This included the introduction of fundamental technologies related to the field of SOC, the conduction of a practical examination, the presentation of the practical service interchange in BPEL, and the finding and presentation of solution approaches to the identified challenges. All these objectives were accomplished. The paper accelerates the entry to the subject of SOC and leads the way towards more advanced works in the domain of SOC.

6.3 Core Insights

It was particularly noteworthy that the discovery and integration of web services still requires extensive manual effort. Semantic methods that allow the functional description of web services were much less applied than it could be expected, considering that these technologies were standardized years ago, such as OWL-S in 2004.

Service solutions from different web service providers are still very heterogenic. Services from the same business domain employ different interfaces and also different functionalities. Service documentations can be hard to find on partially unstructured web sites. The standardization of web services functionalities remains an interesting challenge.

Maintaining a secure cross-cloud SOA requires all conventional measures to meet information security goals. In addition, measures have to be applied that implement additional security requirements that come with SOA. Such requirements are, for example, the implementation of end-to-end security and selective encryption. It is notable that several security standards, such as SAML, WS-Trust, or Kerberos etc. represent fundamental approaches for the implementation of an identity management.
It was recognizable that the subject of trust is a very important aspect of security in cross-cloud environments. Service providers implement services and provide them as black-boxes to their customers. Composite applications can be composed of further composite applications; consequently non-transparent chains of composite services may lead to the participation of unknown service providers that may not be trustworthy. Furthermore, internally applied security standards in a service provider’s composite application cannot be reconstructed without the cooperation of the service provider.

Many services in the World Wide Web are RESTful services. RESTful services can be integrated with SOAP services, even if BPEL and REST are incompatible. REST can be integrated into BPEL with the help of adapters or other methods that encapsulate REST in a SOAP service. IDEs facilitate the integration of RESTful services into SCAs, and allow the connection to BPEL processes by using vendor specific adapters.

### 6.4 Transfer of Insights to a Service Interchange Scenario

Today, in order to implement a dynamic service interchange, functionally equal services need to be identified. The service providers need to be trustworthy. Semantic web services are not yet commonly in use, consequently a lot of discovery and integration work has to be done manually. Due to the lack of semantics, transformations from and to a common interface will still need to be created manually. Services that are unknown at runtime can consequently not be used. Instead a service pool needs to be maintained. If a new service is added, it needs to be deployed. If a dynamic binding of services within that service pool is intended, one must determine which technology shall be used for the task - BPEL or ESB functionality. Both approaches have advantages and disadvantages.

### 6.5 Possible Subjects for future Research

Future papers within this research domain should develop concrete solutions to the solution approaches introduced in this thesis. An example subject could be the identification and implementation of a concrete solution on how to identify functionally equal, or closely related, services that are suitable for a service interchange. Related to this subject it would be interesting to research how mature and how ready for mainstream deployment semantic web services are, and if frameworks and ontologies are available and could be applied. A further topic for study could be determining why semantic methods are still not commonly used. The same approach could be applied to the trust subject, which is an important but difficult subject, as it also is dependent on the individual service providers.

### 6.6 Future Outlook

Current projects in the domain of SOC show a trend towards more automation, fundamental use of semantic methods, the inclusion of non-functional service properties and the application of sophisticated heuristic and statistical methods to, for example, discover logical relations. Semantic methods are used to facilitate an automatic discovery and binding of web services. Heuristic and statistical methods could be used as an alternative. They furthermore intend to facilitate the automatic transformation and integration of web services. Due to these facts, it is not unlikely that, in the future, applications like the hypothetical one mentioned below may become reality.

Future scenarios might contain frameworks that allow a standardized discovery, integration, and orchestration of SOAP and RESTful web services. The integration applications may employ a combination of semantic technologies and data-driven approaches, which use heuristics and statistical measures to derive transformation logic etc.

The processes could be applied further to higher-level applications, which only employ the best-fit services in terms of QoS. Furthermore, the application could allow the
automatic composition of complex composite applications to support and automate complex business problems at runtime. Such business problems could be specified directly by business users that use abstract, text-based formulations with few constraints.

This research introduced the reader to the complex topic of SOC. It revealed multiple challenges and presented solution approaches with varying degrees of sophisticated to resolve various issues. The service interchange may bring challenges, but for all challenges a somewhat mature solution approach exists. Furthermore, sophisticated approaches to dynamically discover, integrate, and execute services, are currently being researched and developed. These insights indicate an interesting future for Service Oriented Computing and Software Engineering in general.

6.7 Reflection on ethical and social Aspects of the examined Subject

The following part of the conclusion contains a brief reflection on ethical and social aspects of the examined subject, also in relation to the given visions/innovation and prospect. Potential influences on society will be discussed.

The findings of this thesis are in essence challenges and solution approaches, that are related to the individual broker supported realization of business processes, with help of external cloud services.

To individually orchestrate, compose, and integrate external IT-services into business processes, while following customer needs, services and their interfaces need to be harmonized or standardized. A standardization of functional properties will follow a homogenization of services that might consequently decrease the range of services offered. It will also intensify the competition as service providers lose their unique selling points and become easily exchangeable. Service providers will, as a consequence, have to merge or diversify in order to maintain enough market share to remain profitable. Some service providers will have to leave the market or give up their business.

Besides the standardization of services, the automation of service discovery and service integration is an interesting vision. Whether this vision will be realized by using semantic methods, or if it is driven by heuristics, in both cases, the automation will lead to high cost cuts and therefore, from a business administrative perspective, will prevail. If assumed that in the future, certain business users might be able to formulate their business problems themselves directly to an IT system, the amount of required IT professionals that used to generate IT solutions for business users will consequently be less in this environment. Another issue though with such a scenario might be a proliferation of specialised software following the IT autonomy of business users.

Two other important challenges identified in this thesis are the subjects of IT security and trust. If the deployment of external IT services within a company’s IT System increases, also the external risks increase. It is commonly accepted that the more external services a company uses, the higher the risk becomes of becoming a victim of e.g. cybercrime, hacking attacks, or corporate espionage. External services are normally still black boxes to their consumers and might be corrupted. Accordingly a possible IT security “arms race” will force companies to spend a lot of money on prevention or force them to disappear from the market. The subject of trust might in the future require service providers to certify themselves in order to be perceived as trustworthy. To achieve a certification, providers will need to reveal their inner security mechanisms to certification authorities (CAs) that need to be trustworthy themselves. Theses CAs might save sensitive security related data about service providers. The solution of a software module monitoring the internal activities of a service provider, as it was described in the solution approaches chapter, shows parallels to the employment spyware software and can also be seen critical. Such a module could be corrupted as well. Not to imagine what happens, if a CA is overtaken by attackers.
However it can be said, that innovation always comes with change, and change is usually connected to a certain amount of risk. Jobs might be rationalized, whereby rationality is also a word for reason. By innovation, activities that were so far carried out by hand, might now be automated, saving costs and also (human) resources. Waste is avoided. This of course in many cases also inflicts labour. Usually jobs are translocated into other fields of activity, especially within the IT sector.

Summarizing, it can be said that through innovation, not only since the industrial revolution, similar processes have always taken place and have been criticised ever since. However the technological development and standard of living nowadays had never been achieved without innovation. Altogether the negative aspects of a lack of innovation or no innovation at all would overweight the negative aspects of most innovations and paradigm changes, such as the industrialization, or the digital revolution etc., by far.
References


[74] L. Constantin, Widely used encryption standard is insecure, say experts: XML encryption, used to secure communications between Web services, can be exploited so that sensitive information is decrypted (2014, Sep. 29).


### Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 2.1:</td>
<td>SOA Layered Model</td>
<td>5</td>
</tr>
<tr>
<td>Figure 2.2:</td>
<td>IT and Services Stack; Source: [8]</td>
<td>6</td>
</tr>
<tr>
<td>Figure 2.3:</td>
<td>Web Service invocation, Source: [11]</td>
<td>8</td>
</tr>
<tr>
<td>Figure 2.4:</td>
<td>Extended Web Service Stack, Source: [28]</td>
<td>11</td>
</tr>
<tr>
<td>Figure 2.5:</td>
<td>Example XML File</td>
<td>11</td>
</tr>
<tr>
<td>Figure 2.6:</td>
<td>Example XML-Schema File</td>
<td>12</td>
</tr>
<tr>
<td>Figure 2.7:</td>
<td>XPath Example</td>
<td>13</td>
</tr>
<tr>
<td>Figure 2.8:</td>
<td>XSLT Application; Source: [25]</td>
<td>13</td>
</tr>
<tr>
<td>Figure 2.9:</td>
<td>Example SOAP Message; Source: [28]</td>
<td>15</td>
</tr>
<tr>
<td>Figure 2.10:</td>
<td>Variables in BPEL</td>
<td>16</td>
</tr>
<tr>
<td>Figure 2.11:</td>
<td>Invoke Activity in BPEL</td>
<td>17</td>
</tr>
<tr>
<td>Figure 2.12:</td>
<td>Assign activity in BPEL</td>
<td>17</td>
</tr>
<tr>
<td>Figure 2.13:</td>
<td>Fusion Order Demo application SCA in Oracle JDeveloper; Source: [38]</td>
<td>18</td>
</tr>
<tr>
<td>Figure 3.1:</td>
<td>Course of action. Source: Own Diagram</td>
<td>30</td>
</tr>
<tr>
<td>Figure 4.1:</td>
<td>Abstracted visualization of the testing BPEL process. Source: Own Diagram</td>
<td>32</td>
</tr>
<tr>
<td>Figure 4.2:</td>
<td>XML-Schema excerpt StockQuoteService1</td>
<td>34</td>
</tr>
<tr>
<td>Figure 4.3:</td>
<td>WSDL excerpt StockQuoteService1</td>
<td>35</td>
</tr>
<tr>
<td>Figure 4.4:</td>
<td>XML-Schema excerpt FinancialDataProcessing</td>
<td>37</td>
</tr>
<tr>
<td>Figure 4.5:</td>
<td>WSDL excerpt FinancialDataProcessing</td>
<td>38</td>
</tr>
<tr>
<td>Figure 4.6:</td>
<td>XML-Schema excerpt ConsumingApplication</td>
<td>39</td>
</tr>
<tr>
<td>Figure 4.7:</td>
<td>WSDL excerpt ConsumingApplication</td>
<td>39</td>
</tr>
<tr>
<td>Figure 4.8:</td>
<td>The ConsumingAPP BPEL process</td>
<td>41</td>
</tr>
<tr>
<td>Figure 4.9:</td>
<td>ConsumingApplication -composite</td>
<td>41</td>
</tr>
<tr>
<td>Figure 4.10:</td>
<td>WSDL excerpt Xignite GlobalQuotes</td>
<td>46</td>
</tr>
<tr>
<td>Figure 4.11:</td>
<td>XML Output: Visualization and description of the output data of GlobalQuotes – GetGlobalDelayedQuote; Source: Service Invocation</td>
<td>47</td>
</tr>
<tr>
<td>Figure 4.12:</td>
<td>Variables definition in ConsumingApp BPEL</td>
<td>48</td>
</tr>
<tr>
<td>Figure 4.13:</td>
<td>Invoke activity definition in ConsumingApp BPEL</td>
<td>48</td>
</tr>
<tr>
<td>Figure 4.14:</td>
<td>Assign activity definition in ConsumingApp BPEL</td>
<td>49</td>
</tr>
<tr>
<td>Figure 4.15:</td>
<td>Definition of Wires in the ConsumingApplication SCA</td>
<td>49</td>
</tr>
<tr>
<td>Figure 4.16:</td>
<td>ConsumingApp BPEL process after Xignite integration</td>
<td>51</td>
</tr>
<tr>
<td>Figure 4.17:</td>
<td>WSDL design view and Header Message Element</td>
<td>51</td>
</tr>
<tr>
<td>Figure 4.18:</td>
<td>Original Assign activity “Assign1”</td>
<td>54</td>
</tr>
<tr>
<td>Figure 4.19:</td>
<td>Assign activity &quot;AssignInputForXignite&quot;</td>
<td>54</td>
</tr>
<tr>
<td>Figure 4.20:</td>
<td>Assign activity &quot;AssignAuthInfo&quot;</td>
<td>55</td>
</tr>
<tr>
<td>Figure 4.21:</td>
<td>XSLT mapping of ‘GetGlobalDelayedQuotes’ to ‘submitStockInformationData’</td>
<td>56</td>
</tr>
<tr>
<td>Figure 4.22:</td>
<td>&quot;Concat&quot; operation in XSLT map</td>
<td>56</td>
</tr>
<tr>
<td>Figure 4.23:</td>
<td>Edit Transform Dialogue</td>
<td>57</td>
</tr>
<tr>
<td>Figure 4.24:</td>
<td>Output map and BPEL Sequence. For a larger version of the BPEL Sequence see Appendix</td>
<td>57</td>
</tr>
<tr>
<td>Figure 4.25:</td>
<td>XML Output of Web Service X StockQuote</td>
<td>59</td>
</tr>
<tr>
<td>Figure 4.26:</td>
<td>cdynes DelayedStockQuote XML-Schema excerpt and SOAP output; Source [56]</td>
<td>60</td>
</tr>
<tr>
<td>Figure 5.1:</td>
<td>OWL-S High Level Overview; Source: [66]</td>
<td>73</td>
</tr>
<tr>
<td>Figure 5.2:</td>
<td>Service Model in OWL-S; Source: [49]</td>
<td>74</td>
</tr>
</tbody>
</table>
Figure 5.3: The Mapping and Translation between OWL-S and WSDL; Source: [66] ................................................................. 74
Figure 5.4: Typical OWL-S Semantic Web service interaction model; Source: [49] . 75
Figure 5.5: XML-Encryption; Based on: [34] ................................................................. 78
Figure 5.6: Web SSO High-Level Use Case, Source: [77] .................................................. 81
Figure 5.7: SAML assertion, Source: [78] ........................................................................ 81
Figure 5.8: REST2SOAP Framework; Source: [42] ............................................................ 86
List of Tables

Table 2.1: XML Languages........................................................................................................ 12
Table 3.1: Definition of Hypotheses ...................................................................................... 23
Table 3.2: Requirements for Services - Overview ................................................................. 27
Table 3.3: Requirements for „semi-mockup Services............................................................... 27
Table 3.4: Requirements for the testing Process - Overview .................................................. 28
Table 4.1: List of Operations of: Xignite GlobalQuote (Legacy); Source: [53, 54] .......... 42
Table 4.2: Comparison of submitData and GetGlobalDelayedQuote ................................... 53
Table 4.3: List of Parameters of WebServiceX StockQuote; Source: [55] ............................ 59
Table 4.4: Markitondemand Parameters ................................................................................. 61
Table 4.5: Comparison of Output Parameters ......................................................................... 62
Table 4.6: “GetSimpleQuote” of API ..................................................................................... 64
Table 4.7: Operation "GetStockInformation" of API ............................................................. 64
Table 4.8: Operation "GetStockStatistics" .............................................................................. 65
Table 5.1: Evaluation of Hypotheses ....................................................................................... 88
Statement of Authorship

I hereby confirm that this master thesis and the research work to which it refers are the product of my own work and that it has not been submitted for a degree at any other university. Any ideas or quotations from the work of other people published or otherwise are fully acknowledged.

<table>
<thead>
<tr>
<th>Place, Date</th>
<th>Signature</th>
</tr>
</thead>
</table>

102
ConsumingApp BPEL process