INSTITUT FÜR INFORMATIK
der Ludwig-Maximilians-Universität München

Diplomarbeit

Selected Languages for Web Services Composition: Survey, Challenges, Outlook

Stephan Leutenmayr

Aufgabensteller:	Prof. Dr. François Bry
Betreuer:	Michael Eckert
Abgabetermin:	31. Januar 2007

Erklärung

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbstständig verfasst habe und keine anderen als die angegebenen Hilfsmittel verwendet habe.

München, den 31. Januar 2007 Stephan Leutenmayr

Abstract

An increasing adoption of e-commerce requires application systems and their business processes to be opened across organizational boundaries for combination with those of other business partners. Businesses offer more and more of their services by means of Web Services which are a standardized way of integrating applications over the Internet in a loosely coupled fashion. Building upon these offered functionalities one is then able to create new functionality by combining existing services into new ones. This combining of Web Services into a new course of action is called *composition*. The term is used to denote both the process of combining and the resulting description. Thereby the participating entities and their interactions are regarded. The descriptions are typically written in languages which provide special constructs for the modeling of compositions.

This thesis surveys languages that are currently developed for the modeling of Web Service composition. In particular we consider the languages BPEL, WS-CDL, and OWL-S. BPEL is emerging as the standard for the modeling of business processes from one participant's point of view, whereas WS-CDL and OWL-S represent areas of active research in modeling from an overarching viewpoint and in adding semantics to Web Services, respectively.

The topic of composition is introduced by differentiating its different aspects and delineating the terms that are used for their description. As the modeling of business processes is one of the main application areas of Web Services composition we then classify the term business process in the broader field of Business Process Management and give an overview of related terminology. Languages for the modeling of business processes require capabilities to describe control flow, data flow, participants, transactions, compensation, and exception handling. We initially address these in an abstract way without regard for any particular language.

After introducing the three languages BPEL, WS-CDL, and OWL-S themselves we compare their goals with respect to Web Services composition and examine the constructs they offer for business process modeling with regard to control flow, data flow, participants, transactions, compensation, and exception handling as well as message correlation.

Zusammenfassung

Mit dem zunehmenden Einsatz von Electronic Business wird es für Anwendungen und deren Geschäftsprozesse notwendig sich über Unternehmensgrenzen hinweg für die Interaktion mit anderen Anwendungen zu öffnen. Organisationen bieten immer mehr ihrer Dienstleistungen mit Hilfe von Web Services an, welche einen standardisierten Weg darstellen, um Anwendungen über das Internet lose miteinander zu verbinden.

Auf diesen Funktionalitäten aufbauend können nun Neue geschaffen werden durch die Kombination von bereits existierenden Dienstleistungen. Diese Kombinierung von Web Services zu einem neuen Prozeßablauf wird Komposition genannt. Dieser Begriff bezeichnet sowohl den Vorgang des Zusammenstellens als auch die daraus resultierende Beschreibung dessen. Dabei werden die teilnehmenden Partner und deren Zusammenspiel berücksichtigt. Die Beschreibung wird typischerweise in Sprachen vorgenommen, die spezielle Konstrukte zur Kompositionsmodellierung bieten.

Gegenstand dieser Arbeit ist die Untersuchung von Sprachen die gegenwärtig für die Kompositionsmodellierung von Web Services entwickelt werden. Im besonderen untersuchen wir die Sprachen BPEL, WS-CDL und OWL-S. BPEL entwickelt sich gerade zum Standard für die Modellierung von Geschäftsprozessen vom Standpunkt eines Betrachters aus gesehen. Die Sprachen WS-CDL und OWL-S repräsentieren Bereiche aktiver Forschung über die Modellierung von einem globalen Standpunkt und über die Verbindung von Semantischem Web und Web Services.

Wir führen ein in den Themenkomplex der Komposition durch eine Abgrenzung seiner verschiedenen Aspekte und der Begriffe die zu deren Beschreibung verwendet werden. Da die Modellierung von Geschäftsprozessen eines der Hauptanwendungsgebiete für die Komposition von Web Services ist, ordnen wir den Begriff Geschäftsprozeß in den größeren Zusammenhang des Geschäftsprozeßmanagements ein und stellen die dabei verwendete Terminologie vor. Sprachen für die Modellierung von Geschäftsprozessen müssen bestimmte Eigenschaften erfüllen. Diese bestehen aus der Beschreibung des Kontrollflusses, des Datenflusses, der Teilnehmer, der Transaktionen, der Kompensationen und der Ausnahmebehandlung. Ohne Betrachtung einer speziellen Sprache stellen wir zunächst mögliche Konstrukte für deren Realisierung vor.

Nach der Vorstellung der drei Sprachen BPEL, WS-CDL und OWL-S vergleichen wir deren Ziele in Bezug auf die Komposition von Web Services. Anschließend untersuchen die Konstrukte, die diese Sprachen anbieten um den Kontrollfluß, Datenfluß, die Teilnehmer, Transaktionen, Kompensationen und die Ausnahmebehandlung sowie die Nachrichtenzuordnung eines Geschäftsprozesses zu modellieren.

Acknowledgments

Several people helped and supported me during my work on this diploma thesis. First of all I would like to thank *Prof. Dr. François Bry* of the University of Munich for his dedication and fruitful suggestions to the topic of this thesis and his encouraging motivation throughout my work.

I also would like to express my gratitude to my supervisor *Michael Eckert* for the fruitful discussions during my work on this thesis, for thoroughly proof-reading the entire thesis, and for his valuable suggestions that helped on writing this thesis.

My thanks also go to the fellow researchers of the *Programming and Modeling Languages research group* at the University of Munich for providing the pleasant working atmosphere and additional suggestions.

Last but not least I am very grateful to my family, especially my parents, and my partner *Jeannette* for their continuous support and encouragement.

x

CONTENTS

CHAPTER

ONE

Introduction

1.1 Motivation

The adoption of e-commerce in both business-to-business and business-to-customer relationships is gaining more and more importance. Not so long ago, business collaborations between companies were realized mostly by humans. If for example a shop was running low on a certain product, an employee would have contacted the wholesaler by telephone or fax for reordering. Then communication media shifted to the use of e-mail or, more advanced, to filling out a web formular. This led to decreased order processing time and more reliable ordering. However, in a more and more globalized world, pace and efficiency are among the key success factors for businesses. Therefore the automation of business interactions is proceeding. An increasing number of businesses is offering their functionalities as computer-accessible services. This allows for the integration of them by applications across organizational boundaries.

The technology of Web Services and their composition is under development to address this integration of applications. Its aim is to support peer-to-peer interactions without the need for a centralized controller and to offer standards for commonly accepted communication based on widely adopted Web standards like HTTP and XML.

Based on the services that businesses offer as standardized Web Services one is then able to create new functionality by combining them into new courses of actions. This aggregation of various Web Services is referred to as *composition*, which in turn can be offered as a new, now composite, Web Service. There are two main viewpoints one can take up when considering Web Services composition. The viewpoint of *orchestration* deals with the aggregation of Web Services as seen from a single participant's point of view, whereas the viewpoint of *choreography* considers the aggregation of the Web Services of all participants in a business collaboration.

The composition of Web Services can be realized by using standard programming languages like C or Java. The usage of specialized composition languages however promises advantages in simplicity of development, automation capabilities, and maintainance, and is expected to provide for a more seamlessly integration. Three languages are currently attracting attention: BPEL for

the modeling of orchestration, WS-CDL for describing choreography, and OWL-S for adding semantics to Web Services.

1.2 Goals of this Thesis

This work surveys selected languages for the composition of Web Services. They are especially developed to offer constructs for modeling control flow, data flow, participants, transactions, compensation, and exception handling.

The area of Web Services composition comprises different aspects and has a quite complex terminology. This thesis aims at clarifying the terms collaboration, conversation, coordination, orchestration, and choreography, that occur in conjunction with composition.

Business Process Modeling is one of the largest application areas of Web Services composition. This thesis will relate it to the enclosing field of Business Process Management and examine its modeling principles for control flow, data flow, participants, transactions, compensation, and exception handling.

Several languages are currently developed for the composition of Web Services. In this thesis the languages BPEL, WS-CDL, and OWL-S will be surveyed as three prominent representatives of the fields of orchestration, choreography, and semantic Web Services. The thesis will identify their common and complementary goals in the area of Web Services composition and analyze their abilities for realizing the principles of business process modeling. Web Services can be invoked concurrently several times thus creating multiple instances.The messages exchanged between different Web Services need therefore to be correlated to the proper instances of the Web Services. This thesis aims at elaborating this problem and identifies three types of message correlation as well as the solutions the languages provide for them.

1.3 Outline

This thesis at hand is composed of 7 chapters. In the second chapter we give a quick overview on Web Services and their essential technologies. The third chapter addresses the composition of Web Services. We introduce the terms used for describing the different aspects and the different viewpoints of composition and give analogies for a better understanding of the terms choreography and orchestration.

The fourth chapter introduces the concepts of Business Process Management, whereupon the phase of Business Process Modeling and its modeling principles are considered more deeply. In this chapter we also outline the relationship of terms in the field of business processes.

In the fifth chapter we provide an elaboration of concrete languages for the composition of Web Services. The following languages are covered: Business Process Execution Language (BPEL), Web Services Composition Language (WS-CDL), and Web Ontology Language for Web Services (OWL-S). We deal the constructs they offer for the modeling of Web Services composition in preparation for the next chapter.

The comparison of the selected languages is in the focus of the sixth chapter. We consider their respective goals and their ability to realize the modeling principles comprising control flow,

data flow, participants, transactions, compensation, and exception handling. Then the problem of message correlation is introduced and the languages are examined regarding their respective solutions.

Chapter seven summarizes the work and gives an outlook on other possible business process realizations.

CHAPTER

TWO

Web Services in a Nutshell

In a more and more globalized world, where businesses are interacting across the globe and pace and efficiency are key success factors, the integration of applications of different businesses represents one of the cornerstones of successful business-to-business (B2B) relationships. The effort of connecting businesses and their software applications in order to facilitate seamless collaborations is not new. Enterprise application integration (EAI) already aimed at providing a means to integrate applications within an enterprise and also across organizational boundaries. But for some reasons the solutions that were developed in this area did not gain widespread adoption for B2B interaction. Among the reasons was a lack of widely accepted standardization of message formats and communication and the dependency on centralized controllers, requiring businesses to agree on a cooperatively managed central platform.

The technology of Web Services and their composition is leveraged to overcome these difficulties and advance application integration. The development of Web Services aims at providing functionality for peer-to-peer interactions without central control and standards that form the basis for a commonly accepted communication. In this chapter we give an introduction on the nature of Web Services and the basic technology for describing, accessing, and publishing them. The chapter is aimed to be a quick introduction to the subject, not a comrehensive overview.

2.1 Web Services

A Web Service can be seen as an entry point to the local information system. It exposes functionality that is provided by the system. The usage of a Web Service is independent from the interal realization of the offered service, the operating system used, or the hardware platform the system runs on. The Web Services Glossary of the World Wide Web Consortium defines a Web Service as follows [WSG]:

> A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in

> a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

This definition designates the goals of the development of Web Services. Its main goal is to facilitate the interoperation between applications without the need for humans to engage in the flow of work. The interaction takes place over a network, which is nowadays the Internet representing an ubiquitous and easily accessible medium for communication.

The term "service" in Web Service denotes that a Web Service resembles a collection of procedures, methods or generally an object with a stable, published interface that can be invoked by clients. The term "web" in Web Service designates that the service offered is accessible over the Internet using Web technologies like HTTP and XML. Web Services are intended to allow applications to communicate directly with one another. There is no requirement for a graphical user interface for human based interaction. Web Services do not offer a user interface, for example in the form of web pages, possibly containing formulars, that are accessed by humans. Such pages might also provide "services" to humans, but this is not what Web Services are for.

When invoking services across the internet in a loosely-coupled way there are some obstacles that need to be overcome:

- **Common Syntax:** A common syntax is needed for the description of data structures and formats to be exchanged. In the area of Web Services XML is used for this as a widely adopted standard.
- **Service Description:** A mechanism is necessary that allows for a standardized description of the interface of a service. For this WSDL is used as the interface definition language.
- **Service Interaction:** Once a service is described a standardization of interactions between services is necessary. The SOAP standard specifies formats for the content of a message and transport protocols to be used for actual transmission of the message.
- **Name and Directory:** A standardized way for publishing, searching, and locating services has to be put up. Web Services rely for this task on the standard UDDI.

The Webopedia, which claims to be the number one online encyclopedia dedicated to computer technology, nicely summarizes the relations between core standards of Web Services [web]:

> The term Web services describes a standardized way of integrating Web-based applications using the XML, SOAP, WSDL and UDDI open standards over an Internet protocol backbone. XML is used to tag the data, SOAP is used to transfer the data, WSDL is used for describing the services available and UDDI is used for listing what services are available.

The usage of already established standards (HTTP, XML) or emerging standards (SOAP, WSDL, UDDI) aims at increasing the adoption by both vendors and users. The basic technology of SOAP, WSDL, and UDDI is introduced in the next section.

2.2 Basic Technologies

In this section we give a brief overview of the technology that is used for the application of Web Services.

2.2.1 SOAP

The interaction of Web Services is based on the exchange of messages. In order to have a basic common understanding of the contents of these messages, the World Wide Web Consortium (W3C) develops the SOAP standard. SOAP is an XML-based protocol for specifying how to format and package the information contained in these messages, and how to transmit them through the network [SOA].

A SOAP message is specified by an envelope that consists of a SOAP header and a SOAP body. This structure follows the approach of standard communication protocols. The header contains data that is necessary for intermediate processing of the message or infrastructure protocols (like security or transaction). The application data that the sender wants to transmit to the receiver is placed into the body part of the SOAP envelope (see figure 2.1).

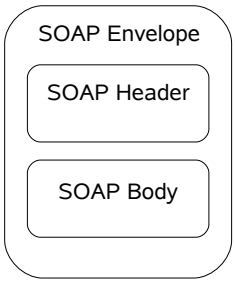

Figure 2.1: SOAP envelope

Though the SOAP standard does not specify further structuring of the header or body parts, there are two patterns that are typically used for the construction of the body part. These are: *document-style* usage and *RPC-style* usage [ACKM04]. In document-style usage, the interacting Web Services exchange XML documents containing the necessary information. These documents are enclosed within the SOAP body part for transmission. Consider the example of a buyer ordering goods from a seller. The buyer would send a document to the seller, containing the ID and quantity of the wanted good (see listing 2.1). The seller in turn would answer by sending another document, containing an element that indicates the confirmation of the order. Of course, both buyer and seller need to agree on the structure of the documents beforehand in order to be able to process them accordingly.

```
1 <PurchaseOrder>
2     <ProductItem>
3         <name>NameOfProduct</name>
4         <quantity>42</quantity>
5     </ProductItem>
6 </PurchaseOrder>
```

Listing 2.1: SOAP Body in document-style

In RPC-style usage, the content of the body part is different. Though the Web Service composition languages which are surveyed in this thesis adopt the document-style usage, we also introduce the RPC-style for completeness. There the body contains actual method calls. In the buyer seller example this means that the body contains the name of the operation and the input and output parameters (see listing 2.2 for an example of the request message).

```
1 <orderGoods>
```

```
2       <param1>NameOfProduct</param1>
3       <param2>42</param2>
4 <orderGoods>
```

Listing 2.2: SOAP Body in RPC-style

Discerning between document-style and RPC-style is somewhat artificial as the body of a SOAP message in both cases actually contains a document. In the first case the document describes what is ordered, in the second case the document describes a method call for actual ordering. What makes a difference is how the document is further processed. In RPC-style usage the document containing a method name and parameters can be mapped directly to a local method call. This functionality can be provided by an infrastructure level of a SOAP implementation. In document-style usage, the content of the document is totally free to the developer of the Web Service. This means that there cannot be a generic mapping to local methods, but there has to be a specific processing.

The SOAP standard does not only specify the format of the exchanged messages but also the transport protocol to be used for transmission through the network. This specification of a transport protocol is called *binding*. A binding defines how messages are to be packaged within the transport protocol and which protocol primitives are to be used. Though the SOAP standard does not impose any transport protocol, HTTP is emerging as the de facto standard for transmission. Thereby, the SOAP envelope is sent in a HTTP request, typically by using the GET or POST functionality. The respective response message is then contained in the HTTP response.

2.2.2 WSDL

The Web Services Description Language (WSDL) is an XML-based language for the specification of the interface of a Web Service. Originally it was created by Microsoft, IBM, and Ariba and is now maintained by the World Wide Web Consortium (W3C) [WSD]. A WSDL document describes the functionality of a Web Service, how to interact with it, and its input and output messages.

An XML document specifying a Web Service interface encloses two parts of the WSDL description: an *abstract part* and a *concrete part*. The abstract part defines *types*, *messages*, *operations*, and *port types* of a Web Service without specifying concrete bindings or addresses, whereas the concrete part defines *bindings* and *ports*.

In the following we briefly introduce the abstract part of a WSDL document:

For the definition of **types** WSDL relies on XML Schema, which has built-in basic types such as String or Integer and allows for the definition of more complex types by the user. For every data structure that is exchanged as part of a message there is a type defined, in order to be correctly treated by the sender and receiver.

For every **message** that is exchanged with a Web Service its format is defined in the WSDL document. Each message is made up of parts describing its content. A part is described by a name and a type. In listing 2.3 a message named “OrderMsg” is defined. It encloses two parts: a part called “productName” which has the predefined type string of XML Schema, and a part called “quantity” which type is also predefined.

A **portType** groups the **operations** that a Web Service offers, as shown in listing 2.3. The operations specify the functionality of a Web Service and correspond to procedures or methods.

Additionally to the name of the operation the input and output messages are specified. There are 4 kinds of operations discerned:

- **Request:** A service request consists of a single message sent to the service without a reply. So, for request operations there is only the input message specified, as there is no response.
- **Request-Response:** This is the most widely used pattern which consists of a service requesting message and a corresponding response message. For this, both an input and an output message are defined.
- **Solicit-Response:** The first message is sent by the Web Service and the client responds with another message. For this pattern also both an input and an output message are specified.
- **Notification:** This pattern describes that only the Web Service sends a message, that is, it notifies the client on something. There only the output message is given.

The concrete part of a WSDL document contains the following:

The **bindings** specify what protocols are to be used when contacting the Web Service, for example SOAP for message formatting and HTTP for transmission. **Ports** provide the concrete network address (given as an URI) at which the service resides.

```
1 <definitions xmlns:xs="http://www.w3.org/2001/XMLSchema">
2     <message name="OrderMsg">
3         <part name="productName" type="xs:string" />
4         <part name="quantity" type="xs:int" />
5     </message>
6
7     <portType name="procurementPortType">
8         <operation name="OrderGoods">
9             <input message="OrderMsg" />
10        </operation>
11    </portType>
12 <definitions>
```

Listing 2.3: WSDL

2.2.3 UDDI

UDDI stands for Universal Description Discovery and Integration. This specification was first proposed by Microsoft, IBM, and Ariba and is now supervised by the standards organization OASIS [UDD]. The goal of UDDI is to allow for publishing, discovering, and locating Web Services.

The information stored in a UDDI registry can be parted into 3 categories. To understand what kinds of information are contained in a UDDI registry often there is an analogy with a telephone directory used: The *white pages* are listings of organizations, their contact information, and the Web Services they provide. This category allows a client to seach for Web Services on the basis of businesses. The *yellow pages* describe a classification of companies and Web Services according to taxonomies. By using this category of UDDI information, a client can search for Web Services based on a wanted category. The *green pages* provide information on the way of invocating Web

Serivces. A client seeking Web Services by their invocation methods can use a UDDI registry in this manner. As being used in the Web Services World, a UDDI registry is accessible as a Web Services and described by a WSDL document.

2.3 Summary

In this section we have introduced the main concepts of Web Services, how they can be described, published and discovered, as well as the message format in use. They are emerging as the standard for facilitating business-to-business integration. A single Web Service may offer more than one operation. For example a Web Serivce of a book store could provide operations for login, browsing and adding items to a shopping cart, and finally confirming the order. The basic technologies that we have covered so far do not provide a means to define a certain ordering that the operations have to be invoked in. Such means are in the focus of the next chapter.

CHAPTER

THREE

Web Services Composition

In the emerging Web Services world companies are proceeding to offer more and more functionalities of their services as standardized Web Services, especially in the business-to-business (B2B) area. Consider the following example scenario: A company enables business partners to trace the processing of business orders online by invoking Web Services. Service consuming partners on the other hand may want to combine this service with other services they need in order to accomplish a certain goal. This aggregation of various services into a larger course of action is referred to as *composition*. This resulting composition may then again be offered as a new, now *composite*, Web Service. Two important aspects of composition are *orchestration* and *choreography*. Furthermore the term composition is also used to describe different viewpoints of the aggregation of Web Services.

In this chapter we first give two analogies for an elementary understanding of the terms orchestration and choreography. We then go on with a more detailed consideration of these and other important terms and concepts associated with composition and examine the composition of Web Services as a way to master complexity. To clarify the different notions of the term composition appearing in literature we then attend to different aspects of composition, namely viewpoints, points of time, and approaches of Web Services composition.

3.1 Analogies

For a better understanding of the terms orchestration and choreography, there are often analogies presented in literature. In the following we want to provide two of these analogies which are regularly given. Usually, the analogy of dancers is used to explain the term choreography and the analogy of an orchestra is used to explain the term orchestration. As with any analogy, they are not fully coherent to the respective technical notions, but they can help for an elementary understanding. In the following we introduce both analogies and have a look at how the relative other concept may be integrated.

3.1.1 The Analogy of an Orchestra

The original meaning of the term *orchestration* stems from the field of music, as the name foreshadows. The Britannica Concise Encyclopedia defines orchestration as "the art of choosing which instruments to use for a given piece of music" [ans06]. In former centuries, the instrumentation of an orchestra was not well defined. Composers usually took all musicians available to them and arranged their instruments into a piece of music. During the 18th century, composers gradually began to deal with the systematic consideration of instruments to achieve a certain impression.

The idea behind the technical term *orchestration* can be highlighted with the analogy of an orchestra. The orchestra consists of one conductor and several musicians. The conductor knows the piece of music, the "plan", which he wants to be performed. Therefore he guides the individual instruments accordingly. The key point of orchestration is the fact that a central instance exists, namely the conductor, which is in full control of the progression of the piece of music.

3.1.2 The Analogy of Dancers

The term *choreography* is derived from the Greek words for "dance" and "write". The American Heritage Dictionary defines choreography as "the art of creating and arranging dances or ballets" ([amh04]). To get a notion of the term *choreography*, often an analogy of dancers is consulted. Thereby, a dancing couple is assumed, consisting of two dancers who together perform certain dance patterns. Both of them act on the instructions of the dance patterns which where designed and agreed on beforehand. While the design is usually not done by dance couples themselves, this can be seen as a kind of design-time action. At run-time both partners execute their respective part of the dance pattern. During the execution however there is no central controller who checks for compliance to the patterns. This is the key idea behind the term choreography: There is no central controller and the partners participate as peers. In this analogy there are only two participants given, as a dance couple consists normally of two people. The idea of a choreography however is not limited to two partners, usually there are more than two.

3.1.3 Extending the Analogies

Above we have introduced a respective analogy for the two terms choreography and orchestration. Now we want to see how it is possible to integrate the respective other concept with the analogies.

Orchestra and Choreography

Below we want to check the possibility of integrating the concept of choreography with the analogy of an orchestra. At first one could interpret the musical composition as a choreography and the composer as the builder of a choreography. Thereby one could at the same time get a differentiation between design-time, which would be the composition, and run-time, which would be the performance. We think however that this extension of the orchestra analogy is not fully appropriate, because the main idea of a choreography is a dezentralized participation of several equal partners. In this sense however, the analogy of an orchestra has only one participant, which is the conductor. Now one could say that the musicians are the equal participants. But this is opposed to the decentralized notion of a choreography, because the orchestra's analogy uses a centralized conductor.

In our opinion there is nevertheless a possibility for integration. The idea is as follows: Consider not one orchestra, but several of them. These are each conducted by their own conductor, but there is no additional master conductor, who exerts central control. Coherence is achieved only by the fact that all conductors adhere to a musical composition they agreed on beforehand. This could then be seen as a choreography.

Dancers and Orchestration

In the following we want to examine how the analogy of dancers can be brought together with the idea of orchestration. Thereby we will extend the decentralized approach of the dancers with a centralized part. We therefore isolate one dancer and consider only his steps. This single dancer can then be seen as a conductor of his own feet. He controls them to perform his part of the choreography. The movements of his feet are fully under his control and he can be seen as their central controller. Thus one would have integrated an explanation of orchestration into the analogy of dancers.

3.1.4 Deficiency of Analogies

Preceding we have introduced an analogy for both the terms orchestration and choreography, respectively. Further on we proposed an extension for both analogies to incorporate the respective other term. However, it is in the nature of analogies that they do not fully correspond to the topic they are intended to explain. For orchestration and choreography, they are a good starting point to understand the ideas and differences behind each term. An important point is not explicitly covered however by both the analogies. That is, a basic principle of Web Services is the exchange of messages to drive the overall process. For this, there is no explicit counterpart in the analogies. One might see the move of the conductor or some gestures or eye contact between the dancers as a sort of message exchange, but this remains somewhat vague.

3.2 Basic Terms

The composition of Web Services for achieving a certain goal is described by different yet similar terms, two of them have been motivated in the preceding section. Commonly found in publications about Web Services are the following terms: *collaboration*, *composition*, *conversation*, *coordination*, *orchestration* and *choreography*.

In the literature on Web Services one can observe some difficulty in discerning whether a term is used in its natural language meaning or whether it is used in a well-defined technical meaning. Furthermore different authors use these terms in different well-defined technical meanings. We give a short delineation of each term to provide a first understanding of the respective meanings. For the following terms we provide a short explanation to have a common understanding of them.

collaboration To collaborate is defined in [mw06] as “to work jointly with others” and “to cooperate with an agency or instrumentality with which one is not immediately connected.” The two important aspects mentioned herein are that a common goal is achieved by working jointly and loosely coupled. An example from the Web Services world would be the ordering from a web shop. This might be realized by using Web Services at both the shop’s and the

customer's side. The Web Service of the shop could offer information on prices and the Web Service of the customer could use this information to decide on the purchase. Shop and customer collaborate to reach the common goal, which is the conclusion of a deal.

conversation A key aspect of conversation is the exchange of messages between the communicating partners. They relate those messages and interact accordingly. In addition, a conversation usually lasts for several of such message exchanges. Thereby one partner may participate in different conversations simultaneously. This is for example the case if a bookshop maintains a conversation with the client and at the same time conducts conversations with several warehouses concerning the requests of the client.

coordination It is likely that component services of a composite service require to reach common knowledge on the outcome of a certain action or group of actions, or to agree on what action to perform next. The aim is to create a common context which may then be used by services to cooperate for effective results. The booking of a journey may for example consist of three separate activities, flight booking, hotel booking, and car booking. If one of these actions fails due to various reasons or does not respond within a given time, one might want to cancel the others also. A context for coordination could be used to reach consensus on the outcome of the actions, wether all were successful or one failed.

For the following terms we give a short overview of their meanings and consider the technical notions in the subsequent sections.

composition From composition results "a product of mixing or combining various elements", according to [mw06]. For a better understanding one can see the term composition in two senses. The part-of sense means that a Web Service encapsulates a combination of Web Services, the sequencing sense emphasizes that a Web Service defines the invocation order of other Web Services [Dus03]. Composition includes the specification of the participating services, in which order they are to be executed and how to handle potential exceptions. There are also two types of composition. With static composition, the concrete Web Services are chosen at design-time, whereas with dynamic composition, there is only a service type specified at design-time and the actual service is chosen at run-time. In our web shop example, composition means that the services at both sides are specified and the single steps necessary for a successful ordering are given.

orchestration The term orchestration denotes the composition of services as seen from the perspective of one participant. The participant considers only those actions he can observe and which are under his control. The web shop example could illustrate this as follows. The shop is only concerned with those interactions it directly participates in. If the customer has additional interactions going on with for example a parcel service, the shop would not be aware of them.

choreography With the description of a choreography the focus lies on a group of participants and its interaction model. A choreography enables the participants to clearly define the rules of collaboration and to agree on them beforehand. There is no centralization point for control or check of compliance. Each participant is responsible for the compliance of its part of the choreography.

Figure 3.1 illustrates the basic terms by means of an example. The whole graphic depicts the choreography in which a customer, shop, two warehouses, and a parcel service participate.

The customer starts a conversation by requesting an item from the shop. The shop starts two conversations, one for each warehouse. The conversation with warehouse 2 ends after the response, but the conversation with warehouse one is continued by ordering the good. The warehouse in turn has a conversation with the parcel service on the commission of the ordered good. The conversation between the customer and the shop is then ended by the confirmation of the shop. An example for an orchestration is is shown at the lifeline of the shop. All exchanged messages of the shop make up its conversation.

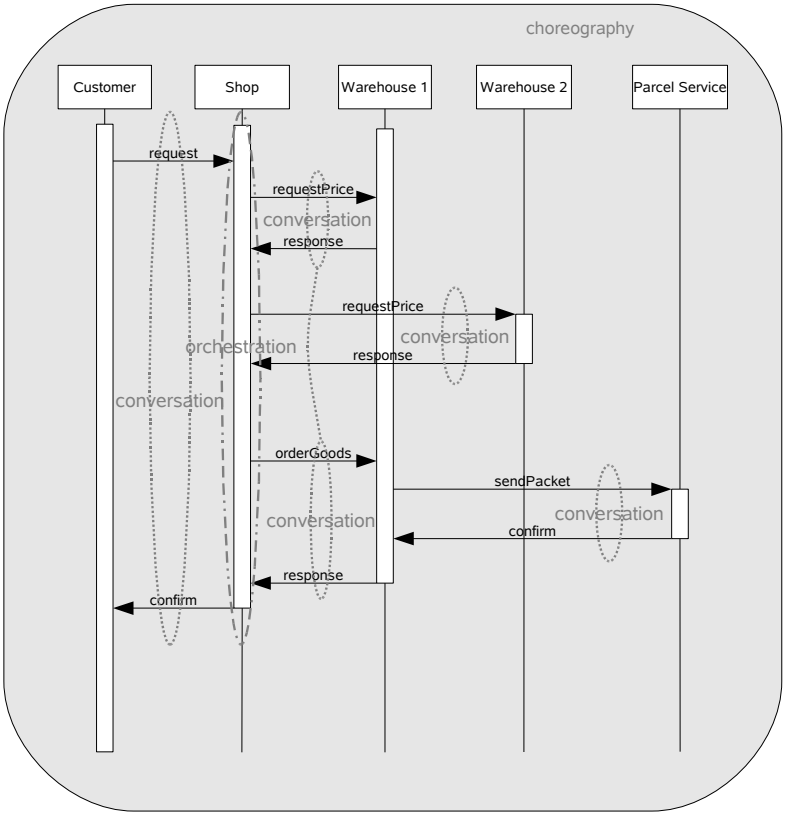

Figure 3.1: Basic Terms illustrated

3.3 Basics of Composition

In the second chapter we have introduced the basics of Web Services, how to define and invoke them. This facilitates simple interactions between a client and a Web Service, as with remote procedure calls. Now we look at requirements for enabling an effective composition of Web Services into a larger course of action and deal with composition as an iterative process.

3.3.1 Requirements for Composition

The use of conventional programming languages for assembling a composite Web Service is a widely used approach. However, programming languages focus on low level details of the implementation of Web Services. There, the developer has to manually assemble the head and

body of the messages by calling several API functions. To ease the modelling of actual business logic and to have abstractions from APIs or infrastructure, the usage of a Web Services composition middleware is desirable. Alonso et al. identify three main elements such a middleware should include [ACKM04]:

- a **composition model and language**, allowing the specification which services participate as component services, the order of their invocation, and the determination of possible parameters. The resulting specification is called *composition schema* [ACKM04]. The language is specifically designed for the composition of Web Services, and the composition schema is then a program written in this language.
- a **development environment**, which enables designers to specify a composition schema. Typically, this is achieved by providing a graphical user interface to the designers. They compose a new Web Service by modelling component services and control flow graphs on a canvas. The result is then a composition schema which can also be translated into a textual represention.
- a **run-time environment**, which takes care of the execution of the composition schema. This includes the invocation of the component services and the verification of the correct order of messages as defined in the composition schema. Such an environment is often called *composition engine* or *orchestration engine*.

3.3.2 Composition to Master Complexity

As Alonso et al. state, one of the most interesting properties of service composition is that it can be iterated [ACKM04]. This means that complex applications are built by progressively assembling components. During this process, components at lower levels of abstraction are aggregated into a larger course of action. This aggregation in turn can be offered as a new service, now residing at a higher level of abstraction. Alonso et al. term this process of aggregation *service composition* and the aggregated services are referred to as *component services*. The resulting higher level service is then called *composite service*. The component services which make up this composite service may in turn be composite services, thus the iteration. They may also be *basic services*, which means that their implementation does not consist of invocations of other Web Services but rather of some arbitrary interaction with the respective local system [ACKM04].

This possibility of iteration facilitates solving complex problems by decomposing them into smaller parts which are easier to cope with. Alonso et al. call this “a way to master complexity” [ACKM04]. Thereby the developers benefit from reuse possibilities of service composition and at the same time the user can seamlessly access complex services [MM04]. Here, the term composition is used in the meaning of a procedure. Below we will explain further notions of the term composition.

3.4 Aspects of Composition

The first section has given a short introduction of the main terms which are used in the area of Web Services composition. This section is dedicated to a better understanding of the terms composition, orchestration and choreography and the differences in their usage. Therefore we deal with three aspects of understanding the term composition. The first one is the point of view,

whether seen as a whole, or whether concentrating on a single part. The second aspect treats the point of time at which a composition can be considered. Lastly, we look at how a composition can be accomplished.

3.4.1 Viewpoints of Composition

When dealing with multiple participants and their interacting services, there are different possibilities for points of view. Following [BDO05], we describe three viewpoints to look at the composition of services: The global viewpoint regards the whole picture including all participants, their services, all exchanged messages, and their possible sequences. The local viewpoint pictures one participant with respect to the external observable behavior and the valid sequences of interactions. The internal viewpoint gives insight into internal implementations of the services of one participant, wether implemented by invoking other services resulting in composite services or by executing code written in arbitrary languages. In the following we have a look at those viewpoints and its characteristics, as well as at different technical terms provided in literature for the respective viewpoints.

Global Viewpoint

This viewpoint concerns multiple participants and their services where the interactions between these services are seen from a global perspective. There a collection of services is engaged in a collaboration in order to achieve a common goal. Global also means that all participants are considered equal from a neutral point of view, not of one participant. A specification of the interactions of services that takes up this global viewpoint is usually called *choreography*, but the authors of ebXML also call it "multiparty collaboration". As described in [KBR+05]

> A Choreography specifies the interactions between services across business entities [...] while leaving actual implementation decisions in the hands of each individual company.

So, a choreography captures the participating services and the interactions in which they engage as well as dependencies between these interactions, including control-flow, data-flow, message correlation, and transactional dependencies. In chapter 6 we will deal with message correlation in more detail since there is some ambiguity in its meaning and realisation. Internal actions, actions that do not result in an externally observable effect, are not described.

A choreography can be seen as a multiparty contract between the collaborating participants that describes the composition of the offered services from a global perspective. The rules of participation in this collaboration can be clearly defined and jointly agreed to. It is then the responsibility of each participant to ensure compliance with this choreography.

Havey identifies as a major design characteristic of choreography the fact that interacting participants cooperate "by exchanging messages with no dependency on a centralized controller" [Hav05]. If a central controller is assumed, this is then seen as a local viewpoint which is dealt with below. Therefore, a differentiation between centralized and decentralized control of actions both seen from a global viewpoint is not feasible.

The modelling of this and the following viewpoints can be achieved using different modelling notations. List and Korherr present among others the following notations: UML 2.0 activity

diagrams, Business Process Definition Metamodel (BPDM), Business Process Modelling Notation (BPMN), and Petri Nets [LK06]. In figure 3.2 we give an example of an UML activity diagram modelling a choreography.

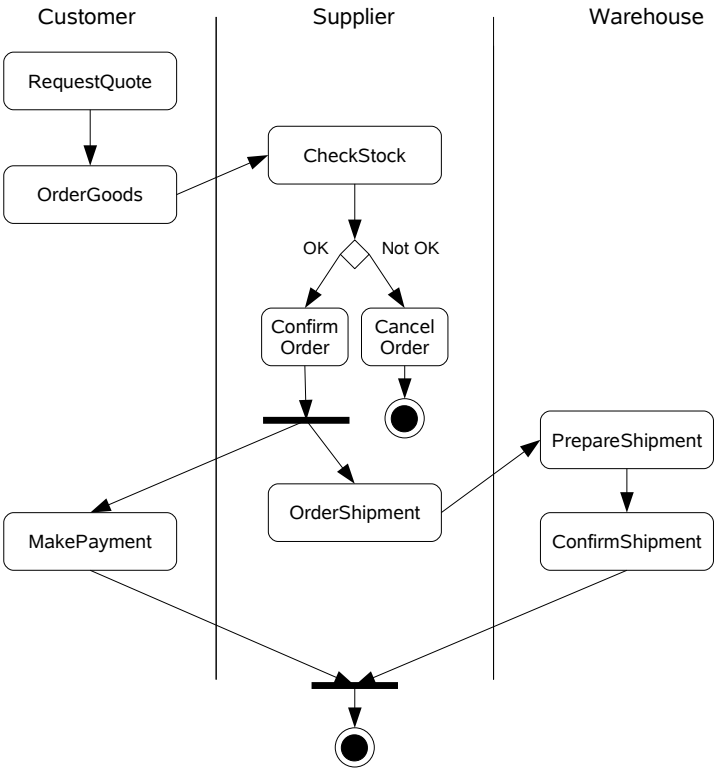

Figure 3.2: Choreography example - UML activity diagram

Local Viewpoint

The local viewpoint focuses on the services of one participant and the behavioral aspects of their interactions with other services. The dependencies of these interactions such as control flow, data flow, message correlation and transactions are described only from the viewpoint of this participant. Like with the global viewpoint there is no description of any internal logic, such as data transformations or the basis of decisions. This viewpoint is identified as *behavioral interface*, as in [BDO05]. The Business Process Execution Language, or BPEL for short, refers to this as an *abstract process*, which denominates the "mutually visible message exchange behaviour"[ACD+].

Again, modelling in the form of an ULM activity diagram is possible. This time, one only looks at the activities of one party. Figure 3.3 shows the externally visible activities of the "purchaseOrder" portion of the choreography of the supplier.

Internal Viewpoint

From the internal viewpoint both the communication actions and the internal actions are considered. The externally offered services can be implemented in two ways. On the one hand they

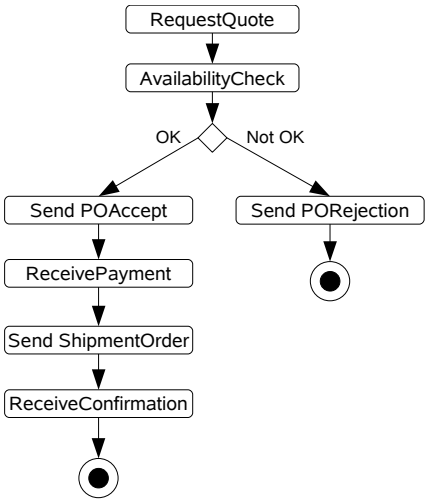

Figure 3.3: Local Viewpoint example - UML activity diagram

can be composed of other Web Services which results in a *composite service* [ACKM04]. On the other hand their functionality can be realized by using a general purpose programming language. The result is then called a *basic service* [ACKM04]. In either way, it results in a description almost without non-determinism. Wether it is realized as a composite or basic service is though irrelevant from the client´s point of view.

The term *orchestration* is widely used to describe a specification that takes up this internal viewpoint. The specification of the Business Process Execution Language (BPEL) however does not use the term orchestration throughout its documents. BPEL refers to this viewpoint as an *executable process* denominating the modelling of actual behavior of a participant in a business interaction [ACD+].

To model the internal viewpoint the UML activity diagram obtained from the local viewpoint is extended. Now, internal actions are added to the diagram to model the full workflow of the implemented service. Figure 3.4 examplarily shows the internal implementation of the activity "AvailabilityCheck".

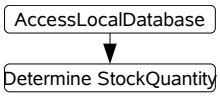

Figure 3.4: Internal Viewpoint example - UML activity diagram

Relationship between Viewpoints

Barros et al. describe the viewpoints presented above as partly overlapping [BDO05]. According to them this is useful in two ways. Firstly, the overlap can provide a basis for checking the consistency between descriptions that take up different viewpoints. Between global and local viewpoint the overlap can be used to find services with corresponding local definitions which will fit with the global definition. Between local and internal viewpoint an existing local definition can be used to check the internal implementation while programming.

Secondly, in the direction from global to internal viewpoint, it is possible to generate code skeletons for the respective underlying viewpoint. From global to local viewpoint, the projection of the global definition to one participant is considered. From local to internal viewpoint, one adds the internal logic that makes up the offered services.

A step further go van der Aalst and Weske as they describe a public-to-private approach to interorganizational workflows. It is based on the notion of inheritance and comprises three steps. First, a shared public workflow is specified. This is then patitioned according to the participants. Lastly, a private workflow for each participant is devised by refining its respective partitioning. The idea is that each private workflow is a subclass of its part of the public workflow under projection inheritance [vdAW01]. The shared public workflow could be identified as choreography, the partitioning as an behavioral interface or abstract process as in BPEL, and the private workflow as orchestration or executable process.

3.4.2 Point of Time

Additionally to the point of view it is also possible to have a look at the point of time at which a composition is considered. We identified from publications three points of time at which to look at what is then called composition, respectively. Firstly, there is composition as a process, then there is composition as the result of a process, and lastly there is composition as seen at run time.

Composition as Process

When considering composition as a process, we have to distinguish between two denotations of the term process. Firstly, it can be seen in the meaning of the (business) process that is performed by a given Web Service or group of Web Services. Secondly the term process can be understood as the development of composed services. In the following we use the term process with the second notion. If we look at composition as a process, we deal with building the specification of a composite service. This can be achieved manually or automatically. The languages BPEL and WS-CDL are intended for manual composition, whereas the language OWL-S aims at facilitating the automatic composition of services.

The overall course is as follows. The input to the process is a set of available services and the functional requirements of the target service. The output is then a specification of the composite service. De Giacomo and Mecella denote the process of combining different services into a larger course of action as *composition synthesis*. To facilitate this synthesis, they give transition systems as an abstraction layer with mappings to either BPEL or WS-CDL. The process of composition is also researched with the aspect of planning. There, a client's request is seen as a goal to achieve. The desired composite service is then a plan, i.e. a sequence of actions, that transforms the initial state into a state satisfying the goal. Further, the automatic generation of a composite service from properly modeled available services and client requests is considered. There the focus lies either on a sufficient description of the desired properties of the respective services, or on the automatic composition according to the client´s request [GM06].

Composition as Result

The term composition is also used to denote the result of the process of combining. With the two languages WS-CDL and BPEL, the result consists of a document which specifies all relevant things as participants, their services, all correct sequences of exchanged messages, possible exceptions and compensation handling. Alonso et al. call this result a *composition schema* and relate it to the source code of a program which is ready for execution. The part which specifies the invocation order of the component services is sometimes referred to as an *orchestration schema* [ACKM04]. The term orchestration however is also used for specifications that describe the internal viewpoint [BDO05], as we explained before. Though there is a connection between the terms orchestration and orchestration schema, as the orchestration schema describes an important part of the internal viewpoint's specification, the similar use of the term orchestration may be confusing.

Composition as Execution

When composing Web Services, eventually executing the result may also be denoted by the term composition. Thereby, at run time there is an execution engine which manages the particular *composition instances*. A composition instance is a composition schema which is executed by a run-time engine. This includes in the case of a composite service the invocation of other services and scheduling the different steps. In the case of a basic service, it consists of executing the code of the respective programming lanuage. This engine is often called *composition engine* [ACKM04].

De Giacomo and Mecella give a distinction between the design-time part and the run-time part of the composition of Web Services. During design-time, the local viewpoint is termed as composition, whereas the global point is termed as choreography. During run-time however, composition translates into the term orchestration and choreography becomes the term coordination [GM06]. Noticably, here coordination does not mean that there exists a kind of explicit coordination protocol which is exerted between the participants, rather it means that the choreography description is used at every participant to check the compliance of the incoming message and to ensure the compliance of the outgoing messages. The choreography is so to say a coordination on the paper. As for every point of time the particular term is shortened to composition, it is often difficult to know which notion is meant in different publications.

3.4.3 Composition Approaches

As we have pointed out above, the term composition denotes not only the result of composing Web Services into a new, larger, course of action. It is also used to term the process of composing. Different strategies for composition as a process can be identified here. Schahram and Schreiner identify among others the following two categories of composition approaches: Static vs. dynamic service composition and automated vs. manual service composition [DS05]. The languages BPEL, WS-CDL, and OWL-S, which are dealt with in greater detail in Chapter 5, fall into these categories. BPEL and WS-CDL are aimed at manual service composition, whereas OWL-S is aimed at automatic composition. As we focus in this thesis on composition as a result, any of these approaches could be used to gain this result. Following [DS05], we give a short introduction to the approaches below.

Static vs. Dynamic Service Composition

The differentiation between static and dynamic composition deals with the point of time at which a concrete Web Service is integrated into the specification of a composition. With static composition the concrete services are determined and integrated into the specification at design-time. Web Services are but an active area where new services are added, and existing services change or disappear. This implicates for the static composition approach that every change of an already integrated services has to be taken into account in the specification. With dynamic composition on the other hand, at design-time there is only a specification of the type of service given. The concrete service is then integrated at run-time. Thereby it is possible that the concrete service has to be discovered first or that is it already known at run-time. The languages BPEL and WS-CDL provide for a static composition approach.

Automated vs. Manual Service Composition

Alonso et al. describe a development environment for manually composing Web Services into a larger course of action. There a graphical user interface is provided where the designer can drag and drop Web Services and compose them in a graph like way into a control flow [ACKM04]. The goal of automated service composition is to automate the search, selection and composition of Web Services, whereas composition is meant as specification of the sequencing. In this area, an ontology-based composition approach is examined. Therefore it is necessary for the respective Web Service to have a standardized and computer-interpretable description of its properties and capabilities. As WSDL does not provide any semantic description for a Web Service, the Web Ontology Language for Web Services (OWL-S) is intended to provide this. The result of automatic composition can then be expressed using a composition language like BPEL or WS-CDL. Thus one can pertain compatibility with established (BPEL) or emerging (WS-CDL) standards for the description of composite Web Services.

3.5 Summary

In this chapter we have introduced the term composition and delineated its different notions with respect to the area of Web Services. Further terms important to the composition of Web Services have been examined. We firstly provided two analogies for a basic understanding of two important concepts of composition, orchestration and choreography. Then we delineated these and the additional terms more technically. Composition was subsequently presented as a way to master complexity in the process of Web Service development. Lastly we illustrated different notions of the term composition by means of three aspects of consideration. These were point of time, point of view, and approaches of composition.

CHAPTER

FOUR

Business Process Management

The modeling of business processes is the largest application area of Web Services Composition. Modeling is part of a larger concept, namely Business Process Management (BPM). According to the Business Modeling & Integration (BMI) Domain Task Force, which develops BPM specifications, Business Process Management aims at the mastering of complex business processes, including organisational measures and different system technologies [bmi06].

In this chapter we give an introduction to the idea of Business Process Management, its terminology and its historical development. We then illustrate the essential characteristics that form the basis of many busines process modeling languages. A comprehensive overview of the abstract constructs a business process modeling language should provide for describing a business process is given. The constructs include facilities for the description of control flow, data flow, and participants, as well as concepts for transactions, compensation, and exception handling.

4.1 Introduction

Business Process Management (BPM) is, roughly put, about the modeling of business processes and therein the flow of their activities. This is typically achieved using standardized graphical and XML representations. Software supporting the whole management process is called Business Process Management System. Havey equates Business Process Management with Business Process Modeling and considers the terms to be synonyms [Hav05]. However this may be confusing as the modeling of a business process is only a part of the whole Business Process Management. Figure 4.1 shows the main elements of Busines Process Management.

If Business Process Management techniques are applied, the Modeling phase is a typical starting point. It deals with the specification of business processes and includes analysis and actual modeling of the single business processes. Analysis is about elaborating business processes that are not already defined or writing down and optimizing existent processes [LR00]. Note that there is a later phase called Analysis which is however different from this step of analysis. Actual modeling is then done using graphical and XML representations as mentioned above. It focuses on the determination of participants and the sequencing of their interactions. A graphical interface

Figure 4.1: Elements of BPM [Bar]

allows for the arrangement of symbols representing participants and activities into graph-like interactions. This graphical represention is then translated to a textual description using XML syntax. This description is commonly realized by employing a special language which offers constructs to easily describe the composition of processes. Since we survey languages for the modeling of business processes, we focus on these languages in the next chapter. For completeness we also introduce the remaining phases of Business Process Management.

The next phase of Business Process Management is Automation. In this phase the processes are actually executed. As a part of a Business Process Management System, an execution engine is used to instantiate the previously defined processes. The engine loads the process definitions like a computer loads programs. It then runs through the single steps of the process, performing the specified activities, managing events and interoperating with external systems.

The phase of Monitoring deals with checking the instantiated processes. Monitoring records and displays the current state of a Business Process Management System. This includes the amount of currently running processes, their average life time, possibly occurring errors, stuck processes, but also the state of the management sytem itself, like CPU or memory usage. Most of this information is presented graphically for easy interpretation by users. It is however possible and in Business Process Management even desired to also automate this step by giving thresholds and rules for appropriate actions. If a process has a typical life time of one day and a certain process is already running for two days it may be addressed according to some rules either by users or automatically by the management system. Furthermore, important events in the life of a process are often recorded in a history for later examination.

This leads directly to the next phase, Analysis. During this phase the previously collected data is evaluated. Two types of analysis can be discerned. Firstly, there is the online analysis which takes place at the runtime of a process. It may include allocating additional CPU time or changing the maximum amount of concurrently running processes. Secondly, there is the analysis of a process when it is completed. The data of the monitoring phase is compared to assumptions that have been made when the process was designed [LR00].

The Improvement phase is conducted based on data of the previous Analysis phase. Improvement is possible on the one hand by adjusting parameters of the management system such as CPU usage per process, or on the other hand by changing the process descriptions themselves, for example due to reoccurring errors. This may be done manually by users, but one goal of Business Process Management is to automate the Improvement phase wherever possible.

The Workflow Management Coalition (WfMC) gives the following key benefits if using sophisticated business modeling techniques [WfM06]:

- **Improved efficiency:** automation of many business processes results in the elimination of many unnecessary steps
- **Better process control:** improved management of business processes achieved through standardizing working methods and the availability of audit trails
- **Improved customer service:** consistency in the processes leads to greater predictability in levels of response to customers
- **Flexibility:** software control over processes enables their re-design in line with changing business needs
- **Business process improvement:** focus on business processes leads to their streamlining and simplification

The Modeling, Automation, and Monitoring phase of Business Process Management and their support by computers originally stem from the workflow domain. Their relationship is dealt with in short in section 4.3.

4.2 Terminology

Several technical terms are used in the area of Business Process Management like business process, process definition, and workflow, to name only a few. In this section we give a delineation of the term process and have a look at the basic terminology of the business process domain.

4.2.1 Processes

The term *process* is derived from the Latin word for "to advance" or "to progress." In computer science, it is used to describe several concepts that have a notion of progression or advancement. In the business process domain, the term business process, which is often shortened to process, is used to describe the flow of work. An operating system process on the other hand denotes the concept of advancement on a level closer to hardware. Since business process management deals with the automatization of business processes with computers, it is important to mind the respective usage. Two of them, namely operating system processes and business processes, are dealt with below.

Operating System Processes

At the hardware level, the term process is used to describe a fundamental concept of operating systems. There, a process is a programm in execution, an "entity that can be assigned to and executed on a processor" [Sta04] and that needs certain resources such as cpu time, memory files, and I/O devices to accomplish its task. It encompasses all information that is necessary to manage processes, such as the program to be executed and data needed by the operation system. Typically, this description includes program instructions, data stored in main memory, allocation of registers, and administrative information. From this point of view, the term process is understood mainly as a description of hardware resources and their possible contents.

Business Processes

The second notion of the term process refers to a description on a higher level of abstraction in terms of hardware and operating systems. The Workflow Management Coalition (WfMC) defines a business process as [Wor]

> A set of one or more linked procedures or activities which collectively realise a business objective or policy goal, normally within the context of an organisational structure defining functional roles and relationships.

As Leymann and Roller point out, the loan process in a bank or claims processing in an insurance company are good examples for a business process. A business process may contain activities only from a single organizational unit, or it may include activities of several different organizations. The term process itself is used to refer to a formalized view of a business process, using for example graphs or Petri Nets for representation. Portions of a business process may be automated activities, others may be executed by humans. The definition of both manual and automated activities is referred to as a *process model* [LR00].

An example for an activity performed by humans would be a conversation with a customer regarding an insurance claim. The reminder of the claim processing could then be done by automatically executing the predefined steps of the business process claim processing. In this thesis, the term process denotes in particular the composition of heterogeneous and distributed Web Services into a new course of action.

4.2.2 Relationship of Terms

The Workflow Management Coalition (WfMC), which is concerned with providing a common architecture and terminology for the industry, defines in its workflow glossary relationships between basic terminology of the domain of workflows (see figure 4.2). Since Business Process Management is concerned with process-oriented applications [Hav05], a business process is at the top of the illustration. Firstly, a business process denotes what is intended to happen. Consider the example of an insurance claim. The intention is the acception or rejection of a claim. The single steps to reach this goal are specified in a process definition, which is a representation of what is intended to happen. A process definition is [Wor]

> The representation of a business process in a form which supports automated manipulation, such as modelling, or enactment by a workflow management system. The process definition consists of a network of activities and their relationships, criteria to indicate the start and termination of the process, and information about the individual activities, such as participants, associated IT applications and data, etc.

A process definition can be split up into sub-process definitions such as reviewing the application for completeness or checking of correctness, and is composed of activities that represent single logical steps. These could be manual activities, such as customer conversation, or automated activities, such as the check for actual membership of the claimant. If the design of a process is completed, its definition is loaded into a workflow management system, which controls automated aspects of the business process. That is, it manages process instances that represent what is actually happening.

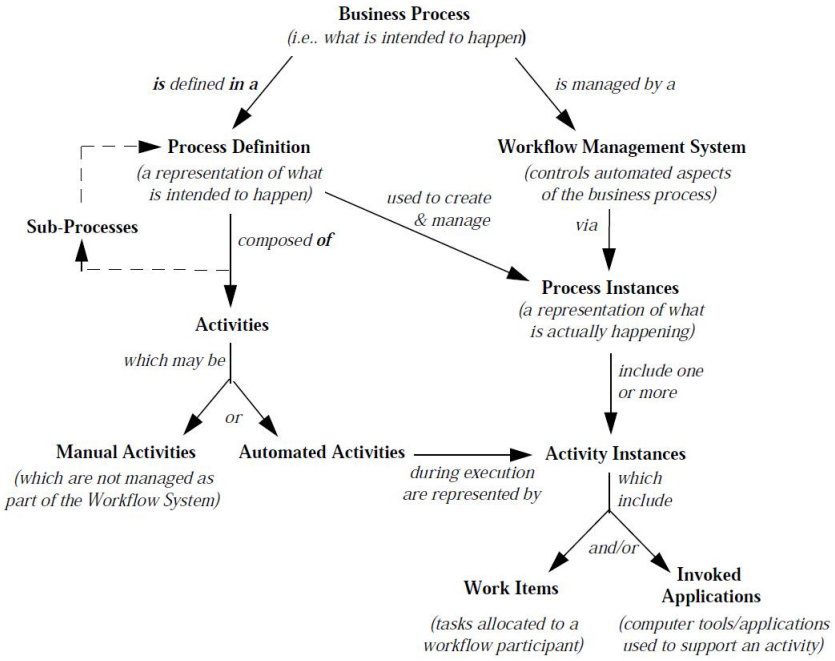

Figure 4.2: Relationship of Workflow Terminology [Wor]

Furthermore, Leymann and Roller discern between process defintion and workflow definition. They call the definitions *models* [LR00]. Figure 4.3 illustrates their differentiation. According to

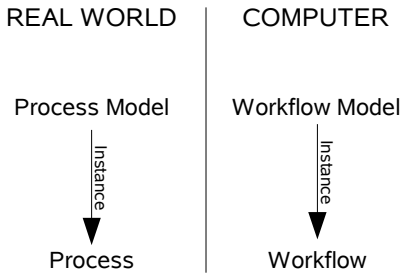

Figure 4.3: Process and Workflow [LR00]

them, a process model "describes the structure of a business process in the real world." This includes the definition of all possible sequences of actions. A process model may be the description of insurance claim processing. From this model, which can be seen as a template, are then instances created. These instances do the actual work, as described in the workflow glossary above. Leymann and Roller base their distinction on the execution by a computer. A process may consist of steps that are not run, and are not runnable, on a computer. Consider for example the conversation with a customer, which is still done by a human. A process model would also capture this part of the process, whereas a workflow model only desribes those part that are runnable on a computer. A process model may consist only of computer-executable steps and in this case workflow and process would be synonyms. However, it may also include description of real world activities that are not runnable on a computer. In this thesis we will continuously use the term process, even if it sometimes corresponds more to the notion of workflow as given above, in order to be compliant with specifications used in the next chapter.

4.3 Historical Development

Historically, Business Process Management evolved from workflow management. A workflow in the classical sense of a *flow of work* is defined in [OHE99]:

> The classical workflow paradigm is a river that carries the flow of work from port to port and along the way value gets added. Workflow defines the operations that must be visited along the way and what needs to be done when exception occur.

In the early days there were no workflows even specified. Routine work was done purely paper-based and there was no written specification of the chain of actions for a certain business goal, for example the processing of an insurance claim. The knowledge on the sequence of work was implicitly known by the employees and passed on between them. This led amongst others to work that was not done and work that was executed doubly. Consider the example of insurance claim processing. On the arrival of a new claim, the receiving clerk knows what to do with it and to whom to pass it along based on his implicit knowledge. If she is off-duty for some reasons, it is possible that there is nobody who has also that implicit knowledge.

Workflows

A first step towards workflows is the written definition of the sequence of actions. All work was still manually done and passed from employee to employee by normal mail, but it was clearly defined when who had to do what. With the advent of computers, workflows were automated. The descriptions of workflows were translated into a form executable by a computer. Additionally, the processed information was digitized making the process as paperless as possible [Hav05]. The level of automation however was low. The single steps of a workflow were now known to an application which assigned them to employees according to the description of the workflow. The application was capable of driving the workflow and starting necessary progams, but did not contribute to the execution of the single steps. The actual work was still performed manually by humans. They just had to care less on the proper order of the steps, because this was controlled by the workflow application. Figure 4.3 shows a typical scenario. A document is scanned and then processed according to a predefined flow of work. Consider again the insurance claim example. Now the receiving clerk digitizes the claim and from that moment the workflow application directs the different steps. It may also reroute assignments of work if a certain clerk is not available.

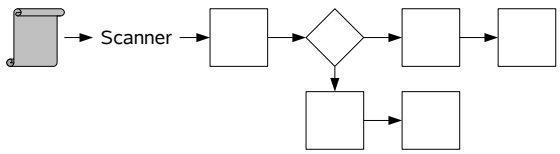

Figure 4.4: Workflow based on scanned document [Hav05]

The next step is the automation of the single process steps themselves [All01]. Parts of a workflow consist of steps that can be executed completely by computers, such as calculations or decisions based on computer-interpretable conditions. That is, parts of a workflow that were previously accomplished by humans are now handled by computers. The Workflow Management Coalition (WfMC) defines the term workflow in their glossary as

> The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of rules.

The workflow application has now to be capable of not only assigning work to employees, but also executing certain program logic. Yet, this logic is typically specified using a general purpose programming language, thus having the developer to focus on low-level details such as libraries and API calls rather than on the actual business logic. The insurance claim example may now look as follows. At certain steps of the insurance claim processing, until now a clerk made decisions based on a set of rules, such as membership of the claimant or whether the claimed service is included in the respective membership. Such steps are now subject to automation.

Workflow Management Systems

A further step is the use of a workflow management system. The WfMC defines it as

> A system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications.

This systems allow for the modeling, automation, and monitoring of workflows. As already described in section 4.1, modeling deals with the analysis and specification of business processes, automation with their execution and monitoring with the checking of the execution.

The last step in current development is the transition from purely document based workflows and workflows with single automated steps to distributed and event driven processes that communicate with each other. They offer their functionalities as services, that can be contacted by other processes. Consider again the example of an insurance claim. There the claim handling process may contact another process which is responsible for verifying the data provided by the claimant. The insurance claim process is in turn offered as a service and can be invoked by other processes interested in its functionality of handling insurance claims. The description of this mutual interaction of services is subsumedly treated by the term composition (see section 3). The concept introduced with the advent of services and communicating processes is called Business Process Management. It broadens the area of workflow management and integrates the automatization of the phases Analysis and Improvement (see figure 4.1).

4.4 Principles of Business Process Modeling

One step of Business Process Management is the modeling of a business process. Typically, this is done by using a graphical notation for easy depiction of the intended process and by translating this graphical representation into a textual description. Such a description of a business process typically includes the specification of all participants and the possible sequences of their activities. This could be achieved by using a general purpose programming language. A more suitable approach is to use a modeling language designed for the specification of business processes. There is however no clear boundary between programming and modeling languages. The available constructs are the ones which are also typically found in imperative programming languages:

control flow and data flow constructs and exception handling. Additional constructs are those for describing participants, and compensation. Below we give an overview of a graphical notation and then introduce the concepts necessary for a sufficient description of a business process.

4.4.1 Business Process Modeling Notation

Havey gives the Unified Modeling Language (UML) and the Business Process Modeling Language (BPMN) as the most suitable approaches for modeling business process behavior [Hav05]. In the following we give a short introduction to BPMN as we will use it to illustrate the modeling concepts below. The Business Process Management Initiative, which deals with standardization of BPM, develops the BPMN. Their goal is to provide a notation that can be understood by all business users, from the business analyst that drafts the process, to the technical developer that implements the process. The BPMN is aimed at being both a simple mechanism for creating business process models and a means of handling the complexity of business process. It provides a graphical notation to describe business processes in a standardized manner and combines readability, flexibility, and expandability. BPMN is able to handle public and private business processes and choreographies, as well as execption handling, transaction, and compensation [Whi06].

Core Elements

The four basic categories of elements are: Flow Objects, Connecting Objects, Swimlanes, and Artifacts. Flow objects are the main graphical elements to define the behavior of a Business Process. There are three Flow Objects: Events, Activities, and Gateways. There are three ways of connecting the Flow Objects to each other or other information. There are three Connecting Objects: Sequence Flow, Message Flow, and Association.

4.4.2 Control Flow

A description of the control flow of a process determines all possible sequences of tasks and consists of a set of constructs that may be combined into any arbitrary sequence. Van der Aalst et al. give 20 patterns that are to support the process designer in assembling activities. These patterns provide standard solutions to several reoccuring problems when dealing with the specification of business processes. They group their patterns into six categories: Basic control patterns, advanced branch and join patterns, structural patterns, multiple instances patterns, state-based patterns, and cancellation patterns. Below we will examine each pattern in more detail and give an example for each in BPMN, following [Whi04].

Basic Control Patterns

Basic control patterns are used to model fundamental process capabilities: sequencing, spawning, and joining of activities as well as parallel execution and branching based on conditions.

Pattern "Sequence" The Sequence pattern is defined as a segment of a process instance with an ordered series of activities. The next activity starts after the previous activity has ended. The

WfMC denotes this pattern as "sequential routing." Example: Shipping goods is started after payment is received.

BPMN uses the term "sequence flow" to refer to this pattern of a series of activities. Figure 4.5 shows the modeling of a sequence in BPMN. White proposes the use of a conceptual "token" to show the behavior of the patterns. This token travels fom activity to activity as given by the connecting arrows of the sequence. Each activity receiving a token is conceptually executed.

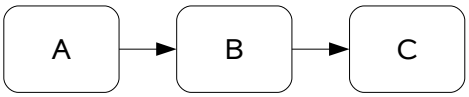

Figure 4.5: Sequence

Pattern "Parallel Split" The Parallel Split pattern allows activities to be performed simultaneously or in any order. The WfMC calls this an "AND-Split" as a part of their concept "parallel routing." Parallel routing is completed by an "AND-Join" at the end of the parallel activities. Example: A travel agency may request quotes from several airlines in parallel for determining the best price.

White describes three possibilities of BPMN to model a parallel split. The first possibility is considered to be used for uncontrolled flow which means that there is no gateway controlling dependencies or conditions (see figure 4.6). Secondly, the use of a gateway as in figure 4.7 is possible. At the gateway the arriving token is duplicated and sent down each outgoing arrow. There is also no check of any conditions. The third mechanism consists of the use of a subprocess (see figure 4.8). If the optional start element of a subprocess is missing, then any activity without incoming control flows will start. For each of those activities a token is generated.

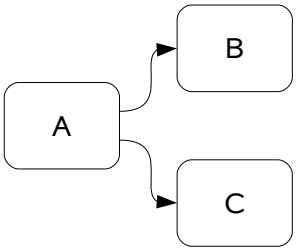

Figure 4.6: Parallel Split - Variant 1

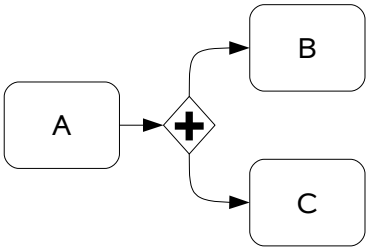

Figure 4.7: Parallel Split - Variant 2

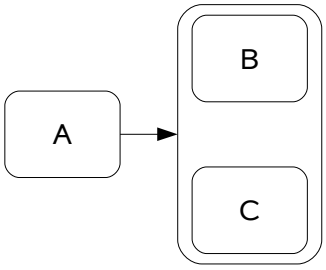

Figure 4.8: Parallel Split - Variant 3

Pattern "Synchronization" The Synchronization pattern is used to converge activities executing in parallel that were generated by a parallel split. The activity after the synchronization is executed not until each final activity of the incoming control flows has completed. The WfMC refers to this as an "AND-Join." Example: The beforementioned travel agency waits for all airlines to send their quotes before selecting the best offer.

The BPMN offers two mechanism to realize the synchronization pattern. Firstly, it is possible to use the Parallel Gateway, already known from the Parallel Split pattern. Figure 4.9 shows the use of the gateway, which conceptually waits for a token from every incoming control flow and then passes a single token to the next activity. The use of a gateway is necessary since a synchronization of flows is to be achieved. The first and second variants of the above Parallel Split pattern use the Parallel Gateway for synchronization. Secondly, the synchronization of splitted control flows is also achieved by the use of a subprocess. A subprocess ends when all of its activities end that do not have outgoing control flows (see figure 4.10). When any such activity and with it the subprocess has ended, a token is passed to the next activity.

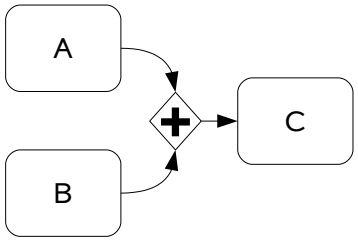

Figure 4.9: Synchronization - Variant 1

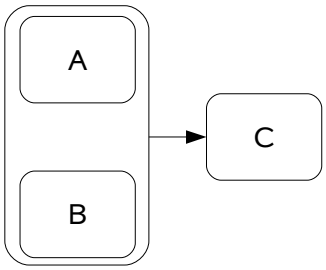

Figure 4.10: Synchronization - Variant 2

Pattern "Exclusive Choice" The Exclusive Choice pattern allows to split the control flow, so that one of two or more possible paths is chosen. The WfMC defines this as an "OR-Split." Example: The activity evaluate_claim is followed by either accept_claim or reject_claim.

The BPMN uses an Exclusive Data-based Gateway to model this pattern. Conceptually, for each incoming token there is exactly one token passed to one of the outgoing control flows. Each outgoing flow is provided with a Boolean expression, which is evaluated at the arrival of a token.

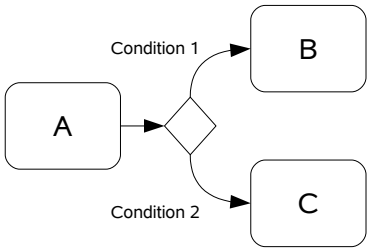

Figure 4.11: Exclusive Choice

Pattern "Simple Merge" The Simple Merge pattern is used to re-join previously splitted branches of an exclusive choice. This pattern differs from the Synchronization pattern in that there only one branch is actually executed. Therefore, no synchronization is required. The WfMC terms this

an "OR-Join." Example: Both activities accept_claim and reject_claim are followed by the activity contact_customer.

Since it is assumed that only one incoming control flow is executed, BPMN allows the use of uncontrolled flow notation, as in figure 4.12. As an alternative, an Exclusive Gateway may be used (see figure 4.13).

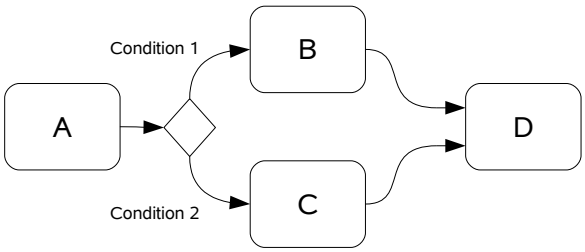

Figure 4.12: Simple Merge - Variant 1

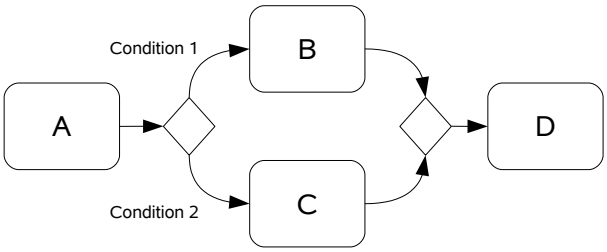

Figure 4.13: Simple Merge - Variant 2

Advanced Branching and Synchronization Patterns

These patterns are useful for coping with more complex situations. They focus on different solutions for advanced branching and synchronization requirements.

Pattern "Multiple Choice" The Multiple Choice pattern is similar to the Exclusive Choise pattern. However, it is allowed to choose from one to all of the alternative control flow paths based on a Boolean expression given for each path. It would also be possible to allow the selection of none of the paths, but that would mean that the process flow stops unexpectedly.

For the realization of the Multiple Choice pattern BPMN offers two constructs. Firstly there is the possibility to omit a central gateway object. Then one puts conditions on each outgoing control flow and marks them additionaly with small gateways at their beginning (see figure 4.14). The second possibility is the use of an "OR" Gateway, as shown in figure 4.15. Every outgoing control flow is provided with a Boolean expression, which is evaluated at run-time. An arriving token is sent to all paths which conditions evaluate to true.

Pattern "Multiple Merge" The Multiple Merge pattern allows for the merging of multiple paths, but without providing synchronization. That is, there is no control of the flow of tokens. If more

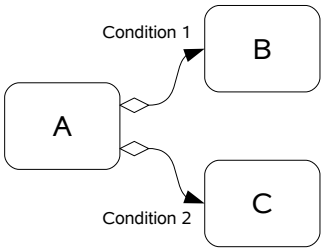

Figure 4.14: Multiple Choice - Variant 1

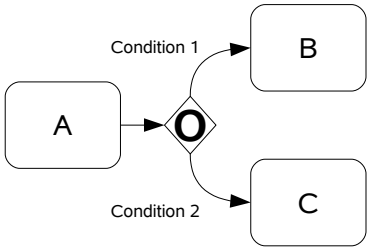

Figure 4.15: Multiple Choice - Variant 2

than one path is followed during execution (e.g. B and C) and the multiple merge location is reached by all of their tokens, the next following activity may be executed for every arriving token. White considers the behavior of this pattern as potentially confusing because a business process instance may be implicitly duplicated [Whi04].

BPMN does not use a gateway because the Multiple Merge pattern resembles an uncontrolled flow merge (see figure 4.16). Each incoming token would cause the creation of a process instance for the next following activity.

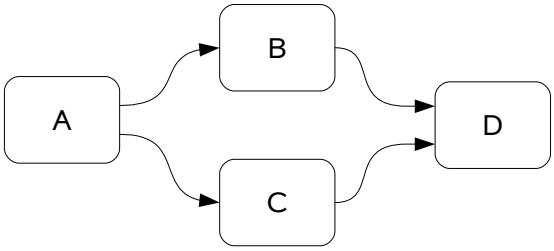

Figure 4.16: Multiple Merge

Pattern "Discriminator" The Discriminator pattern combines paths that were split beforehand. At that point of the control flow the discriminator listens on all incoming paths. When one token arrives, it is passed along to the next activity. The remaining tokens from the other paths are then silently ignored. When all tokens of all paths have arrived the discriminator is reset and ready to pass on another token.

An Exclusive Gateway is used in BPMN to realize the Discriminator pattern. The exclusiveness of the gateway automatically provides for the desired behavior. Figure 4.17 shows an example.

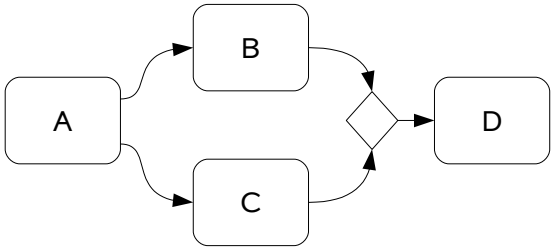

Figure 4.17: Discriminator

Pattern "N out of M Join" This pattern is similar to the preceding Discriminator pattern. The N out of M Join pattern merges M paths into one. Unlike the Discriminator pattern, it waits not for one but for N tokens to arrive. Then, one token is passed along to the next activity. Subsequently incoming tokens are ignored.

BPMN provides a Complex Gateway for handling that kind of patterns. The Complex Gateway can be added with an expression specifiying the number of tokens necessary for passing on a token. As in the Discriminator pattern, later tokens are ignored.

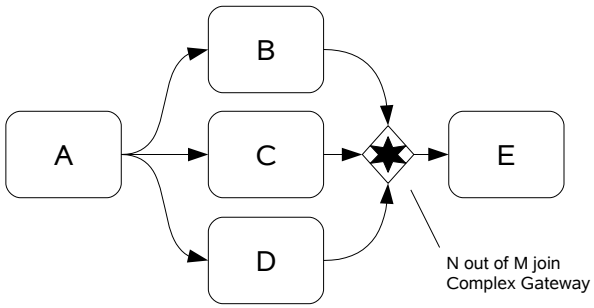

Figure 4.18: N out of M Join

Pattern "Synchronizing Merge" The Synchronizing Merge pattern is used to synchronize parallel paths. The idea is that not all parallel paths are actually executed and so the Synchronizing Merge pattern has to determine how many tokens it has to expect before passing on a token.

BPMN uses an Inclusive Gateway for both the creation and the merge of parallel control flows as shown in figure 4.19.

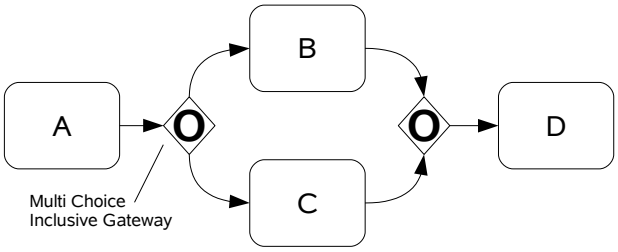

Figure 4.19: Synchronizing Merge

Structural Patterns

Van der Aalst et al. provide two structural patterns: arbitrary cycles and implicit termination. These patterns may make the design more readable, as WHILE and GOTO is simply realizable and there is no need to bring together all endpoints to a single endpoint.

Pattern "Arbitrary Cycles" The Arbitrary Cycle pattern allows the repetition of arbitrary sections of a process. It can be seen as a loop, but an unstructured one, allowing more than one entry or exit point. White denotes this pattern to be "important for the visualization of valid, but

complex, looping situations in a single diagram." It is however not meant to replace the basic construct of a loop.

BPMN uses gateways to create Arbitrary Cycle patterns. Thereby, gateways are connected to other activities by control flows (see figure 4.20).

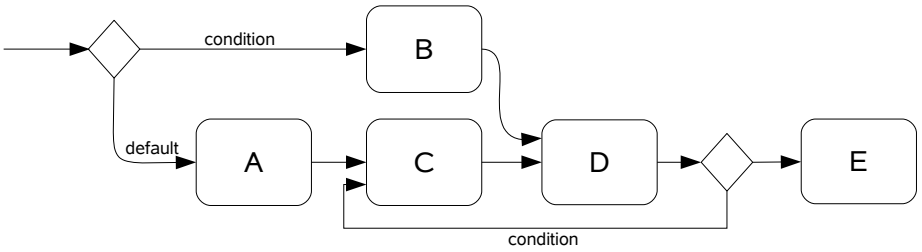

Figure 4.20: Arbitrary Cycle

Pattern "Implicit Termination" The pattern Implicit Termination is used if a specific path is required to end, but other paths should persist. A definition of a process is required in many languages to have exactly one exit point. This leads to complicated merge logics. The Implicit Termination pattern aims at relaxing this design rule.

BPMN offers an End Event to indicate that a particular path has endend. Except of the Terminate End Event, that terminates the entire process, all remaining types of End Events of BPMN can be used to model the Implicit Termination pattern. The variants differ in their results, including message, exception, transaction cancellation, compensation, link to another process, or a combination of them.

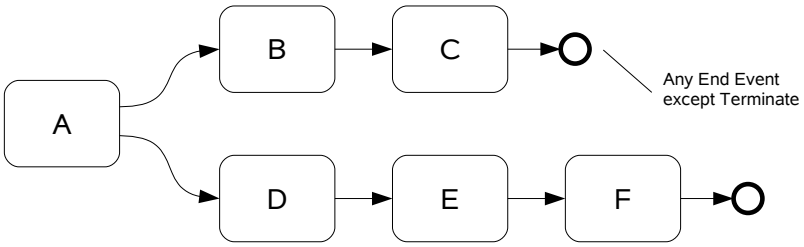

Figure 4.21: BPMN: Implicit Termination

Patterns Involving Multiple Instances

The following patterns deal with multiple instances (MI) of an activity that are run concurrently.Consider as an example of multiple instances the delivery of goods. If there are several goods ordered and they are delivered successively, an activity "deliverGoods" may be instantiated several times.

Pattern "MI With A Priori Designtime Knowledge" The notion of this pattern is that the modeler knows in advance how often the activity should be performed. He then specifies at design time the number of desired parallel running instances in the process definition. At run-time,

the process instances are joined before continuing with the next activity. That is, it is waited for all acitivities B to finish before the following activities starts (see figure 4.22).

To realize the pattern MI With A Priori Design Time Knowledge, BPMN uses three attributes of its activity construct. The LoopType attribute is set to Multi-Instance, the attribute MI_Instance-Generation is set to Parallel, and the MI_Condition attribute is set to the static number of desired process instances (see figure 4.22). The parallel dashed inside the activity denote the use of this pattern. BPMN does not provide a notation for a concrete number, so a comment could be used for this.

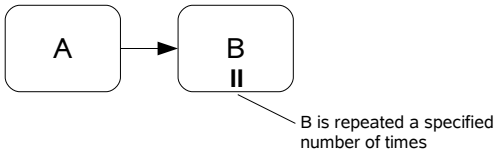

Figure 4.22: MI With Designtime Knowledge

Pattern "MI With A Priori Runtime Knowledge" The intent of this pattern is that the number of desired instances is not known until runtime, but there it is known before the creation of the first instances. Therefore, it cannot already be defined at designtime. Example: When ordering goods, it may be specified that every good is shipped seperately. The amount of goods is not known at design-time, but can be determined before shipping is started, provided that all goods are available.

The solution BPMN offers is similar to the previous pattern. But this time, the LoopCondition attribute of the respective activity is set to determine the number of instances at run-time. It is denoted by an arrowed circle at the bottom of the activity.

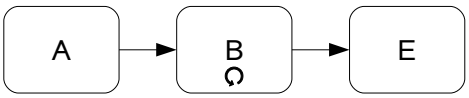

Figure 4.23: MI With Runtime Knowledge

Pattern "MI With No A Priori Knowledge" This pattern is used if there is no knowledge at runtime on the desired number of instances even when executing the first instance. The exact number is determined at some point during the execution of the already running instances and may also be redetermined. That is, also if instances are executed or some are already completed, new instances may be started. It is not possible to realize this pattern with a simple loop or multi-instance pattern.

Figure 4.24 shows the logic necessary for this pattern. Activity B is the actual target of which multiple instance are to be created. It is paired with activity C which determines the necessity of creating additional instances. If another instance is required, one follows both paths marked as "More copies of B needed". The lower one loops to the upstream gateway before the Parallel Split and so to another creation of an instance of B. At the same time Activity C again checks the need for further instances of B. The upper path "More copies of B needed" is necessary for this branch

of the parallel split to end in case of the lower one looping back. If no more copies of B are needed, the synchronization point waits for all instances of B to complete. Afterwards, activity D is started.

Figure 4.24 shows the realization of the pattern MI With No A Priori Knowledge using BPMN. A Parallel Gateway is used to start activities B and C in parallel, and Exclusive Decision Gateways are used for deciding on the necessity of new instances.

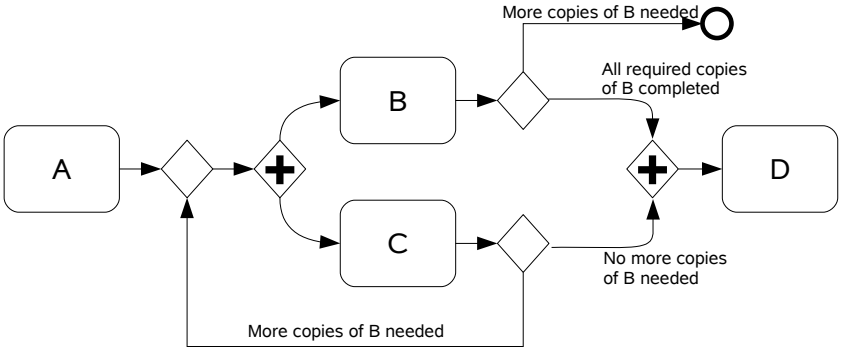

Figure 4.24: MI Without Prior Knowledge

Pattern "MI Without Synchronization" The pattern MI Without Synchronization differs from the previous patterns in that it allows to have no point of synchronization. This pattern allows to spawn off new threads of control. Each of these threads is then independent from the actual workflow. There is no waiting for them to synchronize. That is, the following activity is started right after the creation of the first of multiple instances, as shown in figure 4.25. Note the missing synchronization gateway.

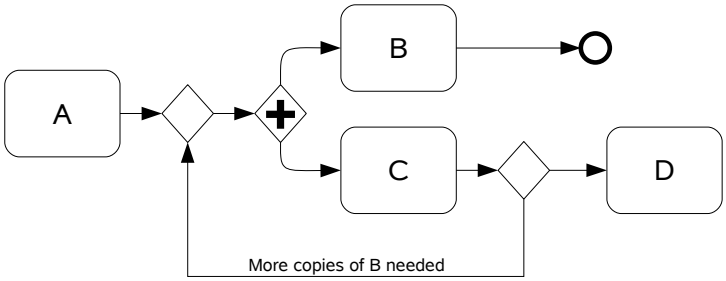

Figure 4.25: MI Without Synchronization

State-based Patterns

Van der Aalst et al. identify three state-based patterns: deferred choice, interleaved parallel routing, and milestone. Those patterns are aimed at processes that are mainly event-driven and offer functionality for waiting for events.

Pattern "Deferred Choice" This pattern is similar to the Exclusive Choice pattern, but the basis for the decision which path to take is different. The point of Deferred Choice in the process will

wait for an event to occur during process execution. Later events for the same or other paths are ignored.
BPMN realizes this pattern by using an Exclusive Event Based Gateway and Intermediate Events of type Message. An arriving token waits at the gateway until the reception of an event. The first incoming event determines which path is taken. Conceptually, the waiting token is then passed along this path. Figure 4.26 shows the pattern modeled in BPMN.

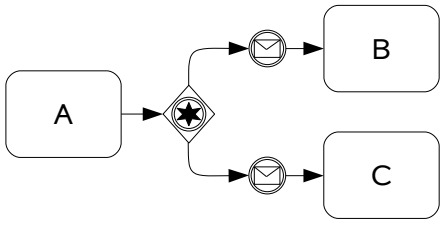

Figure 4.26: Deferred Choice

Pattern "Interleaved Parallel Routing" The trick with this pattern is that the "parallel" activities are in fact executed sequentially. Every activity is executed exactly once and one at a time but there is no order given at designtime. The order is not determined until runtime.
BPMN handles this pattern by using the concept of an Ad Hoc Process. Activities B and C are modeled inside a subprocess, marked with a "tilde", denoting the Interleaved Parallel Routing pattern. The ordering attribute of the subprocess has to be set to sequential.

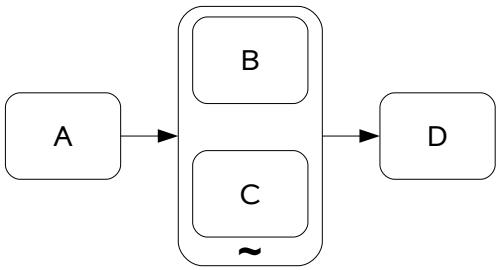

Figure 4.27: Interleaved Parallel Routing

Pattern "Milestone" The Milestone pattern is useful to test wether some part of a process is in a given state. It denotes an activity that is bounded by an enabling event and a disabling event. Before the enabling event or after the disabling event the activity cannot be executed. Havey gives an auction as an example. Bidding may take place at any time between the start and the end of the auction. Figure 4.28 gives an example of a realization using BPMN. Wohed et al. however note that this diagram is an incomplete realization, because it does not model the expiration of the milestone. They object that "an activity (activity D in the example) which potentially can be executed at a certain milestone (in the example, after B has completed and before E has started) is always executed." Wohed et al. propose a workaround to cope with this shortcoming, which is depicted in figure 4.29. Basically, they model activity D, which may be run at the milestone, in a separate process. Before every start of activity D a it is checked whether the milestone is reached. An Event Based Decision Gateway is used to realize the milestone. It is followed by two

branches. One models the expiration of the milestone and the proceeding of the workflow. The other one handles the incoming events and checks the state of the process. Wohed et al. mention two draw-backs of their solution. The milestone is re-initialized with every new loop and the milestone needs to be recalculated if relative time is used [WvdAD+05].

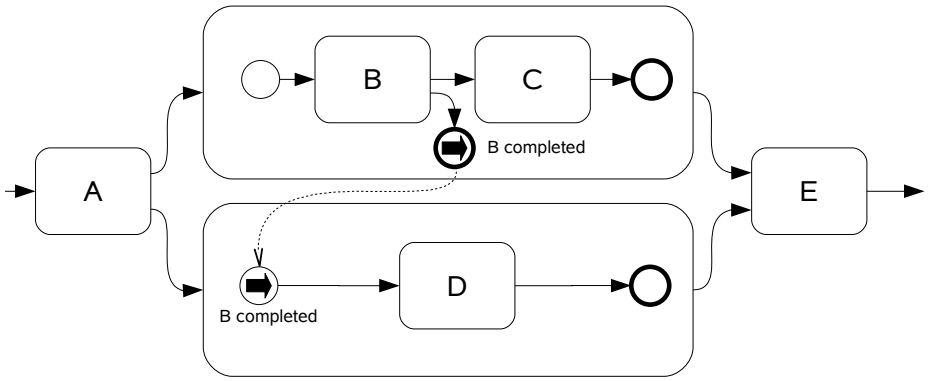

Figure 4.28: Milestone - incomplete solution

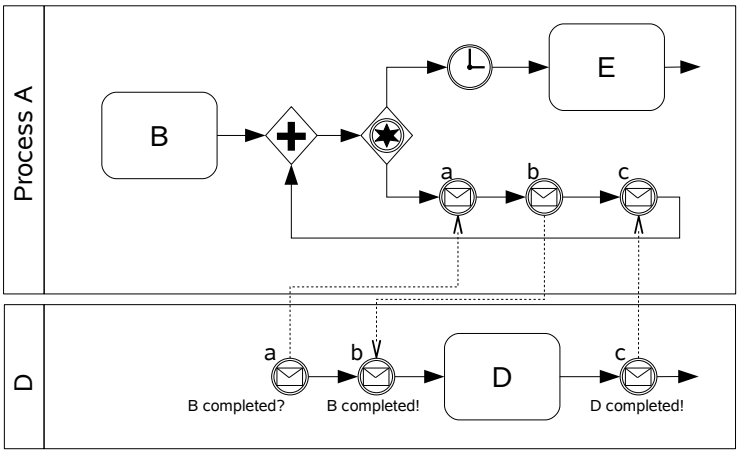

Figure 4.29: Milestone - workaround solution

Cancellation Patterns

It is desirable to have constructs for elegantly cancelling an activity or process at any point of its execution. Cancellation Patterns provide possibilities to cancel a single activity or the whole process. Consider the example of ordering goods. One may want to cancel the whole ordering, or perhaps only the reordering of a unavailable good while maintaining the remaining ordering.

Pattern "Cancel Activity" This pattern describes the ability to stop the execution of an activity. This is particularly useful for long-running or suspended processes. Figure 4.30 gives the modeling using BPMN. The long-running step needs to be canceled. Therefore the canceller B, listening at the same time for cancellation events, notifies the long-running activity C on receipt of a cancel event. Then the long-running activity stops and the following activity can be started. When

considering the ordering of goods, one of these goods may be unavailable at that time. After a certain time or the occurence of a other event, one may want to cancel the reordering. This can be modeled using the Cancel Activity pattern.

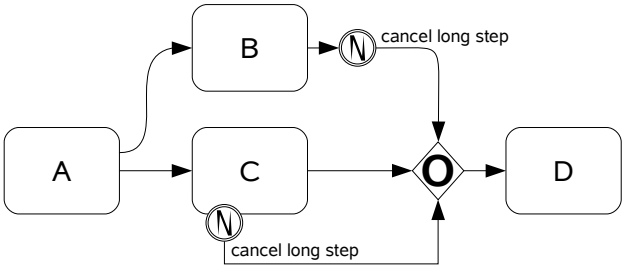

Figure 4.30: Cancel Activity

Pattern "Cancel Case" The Cancel Case pattern is similar to the previous pattern, but it is intended to be used for canceling a whole process or subprocess, not just one or a set of activities. Here again, activity B is used as a listener for events requesting cancellation (see figure 4.31).

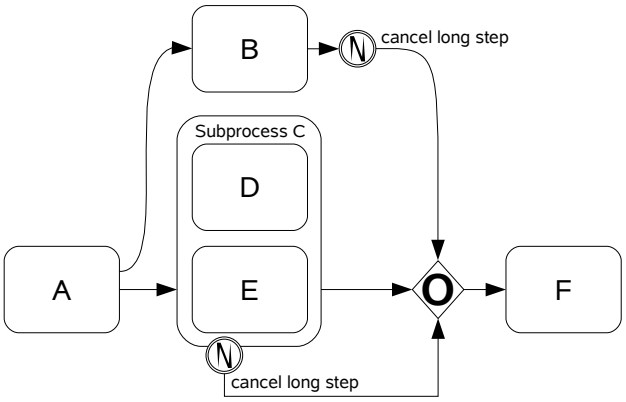

Figure 4.31: Cancel Case

4.4.3 Data Flow

The model of a business process also includes the specification of the flow of data. Nearly every activity needs data of some kind, either for making decisions or for further processing. That data may be numbers, strings, or whole documents that are passed between activities. Two types of data are discerned, *input data* and *output data*. Every activity conceptually has an Input Container and an Output Container. The Input Container stores input data that is necessary for the respective activity. The Output Container contains data that was produced during the execution of the respective activity [LR00]. Figure 4.32 shows a simple case with control flow and data flow being of the same structure. Sometimes however one might need to process data that is not provided by activities immediately connected by control flow. Then, data is not only exchanged with preceding or following activities, but also with other activities that are providing the necessary data. As shown in figure 4.33, the structure of control flow and data flow is different.

The Input Container of activity C receives data from the Output Container of activity A though not directly connected by control flow.

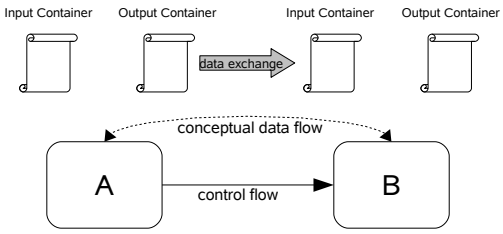

Figure 4.32: Data Flow between activities (simple case) [LR00]

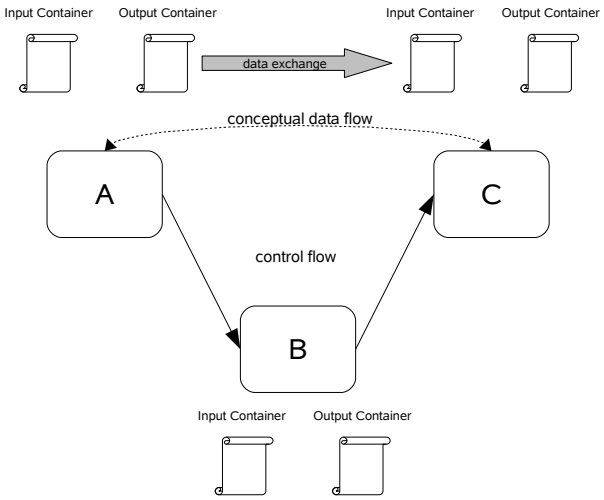

Figure 4.33: Data Flow between activities [LR00]

4.4.4 Participants

As business processes are also aimed to be used for the description of cross-enterprise business interactions, the specification of the participating partners is an important requirement. Participants are all entities that could be responsible for executing a task. Entities include persons for manual activities such as customer interactions as well as applications for automated execution of portions of the process. Thereby, a participant represents both a consumer of a service and a provider of a service. Two kinds of specification are possible. Firstly, one can describe a certain instance of a participant that shall be responsible for executing a certain task. However, if this specific instance is not or no more available, the process definition has to be adopted. Therefore, the second possibility is more reasonable. Instead of participant instances, one specifies the type of the participant and the necessary functionalities. Then it is possible to exchange actual instances if the type of the instances remains the same.

To be able to assign more than one behavior to a participant, the concept of roles is often used. Consider the example of a customer, a retail dealer, and a wholesale trader. The retail dealer represents one participant, but has two different roles. To the customer, he takes up the role of a

seller, but to the wholesaler, he is in the role of a buyer. These are two different behaviors of one participant, brought together by the use of roles.

4.4.5 Transactions

The concept of transaction is used to make applications robust and safe. A transaction is a sequence of actions which is combined logically to one unit of action. Actions within a transaction have to comply to the ACID properties [LR00]:

Atomicity The logical unit of a transaction is to be executed or not. That is, all contained actions are to be executed altogether or none at all. If an error occurs during any of the steps of a transaction, the state before the beginning of the transaction has to be restored. Any steps that are already completed within this transaction have to be reversed. This is called a rollback.

Consistency The state of a process is considered to be consistent, if it complies to certain conditions. A transaction may assume that a process is in a consistent state and has to quit with the process being in another consistent state. During the processing of a transaction there may be temporary inconsistencies.

Isolation This properties means that a transaction is executed as if there were no other transactions at the same time. Effects of other transactions executed at the same time are not to be noticed by a transaction. This ensures that transactions that are in fact executed in parallel have the same effects as if they were performed sequentially.

Durability When a transaction quit successfully, all changes made by this transaction have to be durable. That is, even if the system fails, the effects of a completed transaction are recoverable. Typically this is achieved by storing new consistent states on disk.

An example of a transaction is the transfer of money from one bank account to another. The transfer is performed in two steps: The first step is the withdrawal of the money from the sending account, and the second step is the deposit of the money on the receiving account. It is desirable that both operations are carried out successfully from the perspective of the bank (atomicity). The new state of the system is valid, because the amount of the money withdrawn and the money deposited is equal (consistency). Logically, there are no other operations on both the accounts possible for the time of the money transfer (isolation). All is executed as if there would be no other transactions. Subsequently the effects of the transactions are stored in order to preserve the changes (durability). As these actions are executed automatically and only of short duration, compliance with ACID properties is practicable. As in the previous example, activities may be distributed over different enterprises, in the example over different banks. However these activities are required to have transactional behavior. Leymann and Roller call this set of activities an *atomic sphere*.

Other business processes however may take a longer time to complete. Consider the booking of a journey including hotel, car, and airline reservation. Typically, the request for quotes is not done completely automatically. Therefore the model of an atomic sphere can not be applied. This kind of transaction is commonly called *long running business transaction*.

4.4.6 Compensation

Since business processes may also be executed in a distributed environment and may last for some time, the concept of transaction and its ACID properties may not be suitable. Locks and isolation may be unsustainable for the long periods of some business processes. For this kind of processes the term *long running business transaction* is used. Leymann and Roller denote the set of activities that participate in such long running transactions a *compensation sphere*. Activities within this sphere may have completed successfully, but at some later point it may be necessary to undo all activities, that is, the enclosing compensation sphere.

Typically, the completed activities are already made durable, so simple reversal is not possible as if it had never happend. Therefore, appropriate compensation activities are necessary. Leymann and Roller call this "repair work." Consider the example of a letter being written and sent to a customer. The activity is completed successfully, since the letter was actually written and the customer received it. Later on it is discovered that there have been wrong numbers in the letter. Because the sending of the letter cannot be undone, a compensation of the performed activity is necessary. The "repair work" may be realized by sending a second letter with the right numbers. The activities write letter and send letter are enclosed by a compensation sphere.

Activities put into the same compensation sphere are considered to be highly related. If one of these activities has to be repaired, all other activities have to be repaired too, even if they completed correctly [LR00]. For every compensation sphere there are activities specified that would repair the contained activities. There are two ways to trigger compensation. It may be achieved by sending an explicit compensation event, or by just canceling a compensation sphere or an activity within a compensation sphere. It is not possible to have a default compensation activity for all kinds of processes, because the "repair work" is highly dependent on the domain of a process.

4.4.7 Exception Handling

In the progression of a process there may occur exceptional situations. There are two ways of attending to these situations. The first one deals with an exceptional situation within the normal control flow. The designer has to model the handling using the normal control flow constructs. The second solution consists of a separate handling mechanism for an exceptional situation. This mechanism allows the designer to use specialized constructs that interrupt the normal control flow. The designer associates activities with these constructs that handle the occurred exception. After that the control flow is resumed at a predefined position. It also allows for the forwarding of an exception to an enclosing process to be handled there. This second solution is more supportive of the developer and denotes what is known from programming languages as Exception Handling.

For the second solution, consider the letter example again. The write activity has completed successfully, but during the send activity an exception occurred, maybe the address is unknown and the letter is returned. This can then be dealt with by exception handlers in different ways. It is possible to have a default exception handler that takes care of all occurring exceptions in a generic manner, for example by simply redoing the failed activity. Otherwise one can define specific exception handlers for certain exceptions that are known beforehand to possibly occur. At last, the exception may be forwarded to an enclosing process, if existent, for further processing.

4.5 Summary

This chapter has given an overview of the wide area of Business Process Management. We have concentrated on the modeling aspect of Business Process Management to prepare the subsequent examination of languages specially developed for that. Aspects of the modeling are control flow, data flow, participants, transactions, compensation, and exception handling. We have provided several control flow patterns for the solution of common modeling problems when dealing with business processes. dealing with business processes.

CHAPTER

FIVE

Selected Composition Languages

For the composition of Web Services several languages are under development today. The use of specialized composition languages promises advantages in simplicity of development, automation capabilities, and maintainance compared to the use of standard programming languages like C or Java. Web Services and their composition are expected to allow for "a seamless and almost completely automated infrastructure for electronic commerce" [ACKM04]. Three areas are of active research in conjunction with Web Services: orchestration of Web Services, choreography of Web Services, and the extension of Web Services with semantic descriptions. In this chapter we consider the following three languages which receive most attention in their respective area of application: Business Process Execution Language (BPEL), Web Services Choreography Description Language (WS-CDL), and Web Ontology Language for Services (OWL-S).

Several languages for the composition of Web Services as seen from an internal point of view are proposed: BPML[1], WSFL[2], BPSS[3], BPEL, to name only a few. The Business Process Execution Language has the strongest backing by leading vendors, including Microsoft, IBM, and SAP. It is emerging as the standard language for describing the orchestration of Web Services [ACKM04], thereby reconciling concepts of former languages.

In recent years it was realized by different vendors and researchers that there is also a need for choreography when dealing with business processes. It is not only important to be able to describe service collaborations from a single businesses point of view, but also to have coherent description of the interactions of multiple partners. An upcoming language aimed at facilitating the choreography of business processes is the Web Services Choreography Description Language (WS-CDL). This language offers constructs that allow for the modeling of interactions of business processes as seen from a global perspective. The resulting description can be seen as a contract between partners that they agreed upon.

Facilitating automatic finding and binding is also an active area of research in relation to Web Services. Therefore the language OWL-S is developed. The acronym stands for Web Ontology Language extended with features for modeling Services.

[1] Business Process Modeling Language
[2] Web Services Flow Language
[3] Businees Process Specification Schema

5.1 BPEL4WS

5.1.1 Introduction

Business Process Execution Language (BPEL4WS or BPEL) is an XML-based language for the specification and execution of business processes using Web Services. Its development was started in 2002 by IBM, Microsoft, SAP, Bea, and Siebel. The concepts of the language are based on two former languages, Web Services Flow Language (WSFL) of IBM and XLANG of Microsoft. WSFL was designed for the composition of Web Services and is based on the concept of directed graphs. In this concept activities are represented by nodes and connected by links denoting the control flow. XLANG originally served as a modeling language for business processes. It is based on the concept of block structuring as familiar from most current imperative programming languages for modeling the control flow between activities. BPEL now combines and extends the capabilities of both its predecessor languages.

A BPEL process description defines the control flow of multiple service interactions between participating partners which collaborate to achieve a business goal. It establishes a stateful interaction model and specifies the business logic that makes up the process being developed. BPEL also provides mechanisms for dealing with exceptions during the flow of work and for processing faults. The compensation of activities, that is taking back their effect, either because of exceptions or on request, is also supported.

Web Services specify their functionality using WSDL interface descriptions. The interaction model of WSDL however supports only stateless interactions, that is, single request-response interactions. Business processes are often long-running transactions that need to maintain a certain state and need to wait asynchronously for correlated messages. BPEL is a means for adding state and correlated asynchronous messaging to the stateless interaction model of WSDL.

Figure 5.1 shows the position of BPEL related to the Web Services stack. It comprises the following four layers:

Transport The transport layer handles the actual message exchange between the Web Services. It uses protocols like HTTP, SMTP, or FTP.

Messaging The encoding of the transmitted data is done at the messaging layer, using XML-based message formats like SOAP or XML-RPC.

Interface Description On this layer the interfaces of the participating Web Services are described using WSDL. It allows for stateless communication.

Orchestration The orchestration layer is the home of BPEL. The business logic and the control and data flow are determined on this layer.

BPEL provides two ways of describing a business process. Firstly it is possible to describe only the publicly visible part of a business process, the *business protocol* [ACD+].This facilitates the precise specification of the mutually visible message exchange behavior of each participant, but does not reveal their internal implementation. This corresponds to the local viewpoint described in section 3.4. Secondly, BPEL allows for a complete description of the internal logic of a business process, which leads to an executable process. This corresponds to the internal viewpoint of section 3.4.

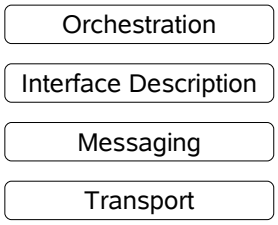

Figure 5.1: Web Services Stack

5.1.2 Structure of a BPEL Process

The BPEL description of a business process relies on three elements:

- WSDL descriptions of invoked partner services
- WSDL description of the process
- BPEL description of the process

The invoked Web Services of collaborating partners are described using the standardized interface description language WSDL. The BPEL process being developed is again a Web Service and as such its interface is also described by a WSDL document. This WSDL description can be generated automatically from the BPEL description. The specification of the process logic itself is stored in a BPEL document. In the following we deal with the content of this BPEL process description.

A BPEL process description follows a block oriented structure and consists of one single activity at the top level. This activity may be further structured and may contain also graph-like parts. The element *process* is the top level element. Its attributes define some global values. The attribute "queryLanguage" specifies the XML query language that is used for the selection of nodes in the XML messages exchanged between the process and the partner Web Services. The attribute "expressionLanguage" declares the expression language that is used for the description of conditions controlling control flow (e.g. the condition of a while-loop). Both attributes have XPath 1.0 as default values, denoted by the URI of the specification: http://www.w3.org/TR/1999/REC-xpath-19991116. The attribute "enableInstanceCompensation" specifies whether an instance of that process as a whole can be compensated. The default value is "no." The attribute "abstractProcess" determines wether it is an abstract process or an executable one. The default process is executable, so the value is "no." In the following we give an overview of the basic structure to introduce the parts of a BPEL process description.

```
1 <process name="ncname" targetNamespace="uri"
2             queryLanguage="anyURI"
3             expressionLanguage="anyURI"
4             enableInstanceCompensation="yes|no"
5             abstractProcess="yes|no"
6             xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process
                  /">
7     <partnerLinks>
8         <partnerLink name="ncname" partnerLinkType="qname" myRole="ncname
              " partnerRole="ncname">
9         </partnerLink>
10    </partnerLinks>
```

```
11      <partners>
12          <partner name="ncname">
13              <partnerLink name="ncname"/>
14          </partner>
15      </partners>
16      <variables>
17          <variable name="ncname" messageType="qname" type="qname" element=
                "qname"/>
18      </variables>
19      <correlationSets>
20          <correlationSet name="ncname" properties="qname-list"/>
21      </correlationSets>
22      <faultHandlers>
23          <catch faultName="qname" faultVariable="ncname">
24              <!-- some activity -->
25          </catch>
26          <catchAll>
27          <!-- some activity -->
28          </catchAll>
29      </faultHandlers>
30      <compensationHandler>
31          <!-- some activity -->
32      </compensationHandler>
33      <eventHandlers>
34          <onMessage partnerLink="ncname" portType="qname" operation="
                ncname" variable="ncname">
35          <correlations>
36              <correlation set="ncname" initiate="yes|no">
37          <correlations>
38              <!-- some activity -->
39          </onMessage>
40          <onAlarm for="duration-expr" until="deadline-expr">
41              <!-- some activity -->
42          </onAlarm>
43      </eventHandlers>
44      <!-- some activity -->
45  </process>
```

Listing 5.1: Basic Structure of a BPEL Process

The overview given in listing 5.1 does not contain any activities yet. They may be any of the following: *receive, reply, invoke, assign, throw, terminate, wait, empty, sequence, switch, while, pick, flow, scope, compensate*. We will explain these activities in a subsequent section.

Interactions

The main goal of BPEL is the composition of Web Services into a new larger course of action, which is in turn offered as a Web Service. As most of the composed services are offered by different busines entities BPEL needs the ability to model relationships between partners. In the following we deal with the constructs BPEL offers to model the relationships between partners and their interacting services. BPEL realizes this with the constructs *partnerLinkType*, *partnerLinks*, and *partner*.

Partner Link Type Partner Link Types describe a pair of message exchanging Web Services by defining the roles they play during their conversation. Together with the role, exactly one WSDL port type is specified. Web Services that want to participate in such a link for example as a Buyer have to implement the respective port type. Listing 5.2 shows a partner link type named BuyerSellerLink, and the participating roles Buyer and Seller have to comply to the respective WSDL port types.

```
1 <partnerLinkType name="BuyerSellerLink" xmlns="http://schemas.xmlsoap.org
      /ws/2003/05/partner-link/">
2     <role name="Buyer">
3         <portType name="buy:BuyerPortType" />
4     </role>
5     <role name="Seller">
6         <portType name="sell:SellerPortType" />
7     </role>
8 </partnerLinkType>
```

Listing 5.2: Partner Link Type

Partner Links Partner Links denote business relationships of a BPEL process with other Web Services. One partner link defines exactly one relationship between a BPEL process and a Web Service. Listing 5.3 shows an example of such a link. The type of the link is BuyerSellerLink, denoting the necessary types of the services and hence their WSDL ports. The BPEL process is in the role of a buyer, and the contacted services plays the role of a seller.

```
1 <partnerLinks>
2     <partnerLink name="wharehouseLink" partnerLinkType="BuyerSellerLink"
          myRole="Buyer" partnerRole="Seller" />
3 </partnerLinks>
```

Listing 5.3: Partner Links

Partner Whereas a partner link defines the relationship between two communicating processes, a partner element is used to represent business partners in general. The definition of a partner is optional and gives constraints on the functionalities of a business partner. The partner "SellerShipper" of listing 5.4 has to be capable of providing Seller and Shipper functionalities, denoted by the respective partner links.

```
1 <partner name="SellerShipper" xmlns="http://schemas.xmlsoap.org/ws
      /2003/05/partner-link/">
2     <partnerLink name="Seller" />
3     <partnerLink name="Shipper" />
4 </partner>
```

Listing 5.4: Partner

Scopes

The above concepts of partner link type, partner links, and partners are only globally definable. In contrast, Variables, Correlation Sets, Fault Handlers, Compensation Handlers, and Event Handlers

are also locally definable by using the concept of Scopes. BPEL offers Scopes to have a local validity block. The top level process element is the outermost scope and may contain further, also nested, scopes. There has to be at least one activity inside each scope. Listing 5.5 shows the possible contents of a scope.

```
<scope>
    <variables> ... </variables>
    <correlationSets> ... </correlationSets>
    <faultHandlers> ... </faultHandlers>
    <compensationHandler> ... </compensationHandler>
    <eventHandlers> ... </eventHandlers>
     ... some activity ...
</scope>
```

Listing 5.5: Possible Contents of a Scope

Variables Variables are used as input and output containers for messages of the invoke, reply, and receive activities, as well as to store messages that constitute the state of a business process. Variables are defined within a scope and belong to that scope, so local variable definitions are possible. The assignment of data to a variable or copying data from one variable to another is a common task which is facilitated by the assing construct.

Correlation Set A correlation set defines which parts of an exchanged message are used to correlate this message to previously sent or received messages. This is dealt with in more detail in the subsequent section on correlation.

Fault Handler Fault Handlers deal with occurring faults by defining activities for faults that are known beforehand. BPEL uses the concept of a catch construct which is already familiar from imperative programming languages. If an activity fails, the thrown fault is handed over to the fault handler of the enclosing scope. In case there is no fault handler for that specific fault, the fault is passed along up the scope hierarchy until the top level element, process, is reached. It is also possible to define a "catch All " fault handler that takes care of faults in a generic way (see listing 5.6).

```
<scope>
    <faultHandlers>
        <catch faultName="FaultName1">
            <!-- catch-logic-A -->
        </catch>
        <catch faultName="FaultName2">
            <!-- catch-logic-B -->
        </catch>
        <catchAll>
            <!-- generic handling -->
        </catchAll>
    </faultHandlers>
    <!-- some activities -->
</scope>
```

Listing 5.6: Fault Handler

Compensation Handler As described in 4.4.6 it may be desirable to undo previously completed work. Compensation Handlers specify activities that are assigned to a scope in order to compensate the enclosed activities (see listing 5.7). Compensation handlers are triggered by the activity *compensate*. The compensation of a scope is passed along to all enclosed scopes of the compensated scope.

```
<scope>
    <compensationHandler>
        <!-- activity compensating the normal effects of that scope -->
    </compensationHandler>
    <!-- normal activities -->
</scope
```

Listing 5.7: Compensation Handler

Event Handler Every scope can be assigned one event handler, as illustrated in listing 5.8. This handler is capable of dealing with two kinds of events. Firstly, incoming messages can be processed by the activity *onMessage*. Secondly, it is possible to react to alarms (*onAlarm*), which can be triggered by duration or deadline.

```
<eventHandlers>
    <onMessage partnerLink="Buyer" operation="cancelOrder">
        <terminate />
    </onMessage>
</eventHandlers>
```

Listing 5.8: Event Handler

Activities

BPEL discerns between *basic* activities and *structured* activities. The following basic activities allow for the modeling of communication with other services:

invoke The invoke activity facilitates the invocation of partner Web Services. It is either a asynchronous one-way operation, that is, send a message to a Web Service, or a synchronous two-way operation, that is, send a message and wait for an answer. Listing 5.9 shows the invocation of the "buy" operation of the service defined in the partnerLink "wharehouseLink."

```
<invoke
        partnerLink="wharehouseLink"
        portType="SellerPT"
        operation="buy"
        inputVariable="itemid"
        outputVariable="response"
</invoke>
```

Listing 5.9: invoke Activity

receive As a BPEL process contacts other Web Services to make use of their functionality, it may also be contacted itself. To model this behavior, BPEL offers the receive construct. Thereby

the partner link that the process expects to receive from and the port type and operation that the process expects the partner to invoke are specified (see Listing 5.10). The receive activity is also important for the lifecycle of a process, as the attribute "createInstance" indicates. It allows the specification whether a new instance of that process should be created on the receipt of a message. This is the only way in BPEL to cause the creation of a new process instance.

```
1 <receive
2         partnerLink="wharehouseLink"
3         portType="SellerPT"
4         operation="getPrice"
5         variable="id"
6         createInstance="yes"
7 </receive>
```

Listing 5.10: receive Activity

reply The reply construct is used to answer synchronously received messages. A variable may contain possible data to be sent back. Note that asynchronously received messages are replied using an invoke activity, because asynchronous messaging conceptually is a one-way communication.

```
1 <reply
2         partnerLink="wharehouseLink"
3         portType="SellerPT"
4         operation="buy"
5         variable="id"
6 </reply>
```

Listing 5.11: reply Activity

Synchronous communication is realized by defining a receive element and a subsequent reply element, with possible processing of received data between them. Asynchronous communication is modeled by specifying a receive element and at a later step a corresponding invoke element. That means that a partner has to provide a callback Web Service for the realization of asynchronous communication.

Further basic activities are *throw* which is used to signal a fault, *wait* which serves for waiting for a certain time or until a deadline, *terminate* which terminates a process instance, *assign* with which values can be assigned to variables, and *empty* which just does nothing, for example to catch and suppress a fault.

The main goal of BPEL is the orchestration of Web Services. In order to realize this, BPEL offers structured activities which group a set of other structured or basic activities. The following structured activities are specifiable:

Sequence The sequence activity contains a set of structured or basic activities which are executed sequentially in lexical order.

Switch The swich construct is similar to those of imperative programming languages. It consists of activities, each associated with a condition. In lexical order all conditions are evaluated until the first condition holds true. Its enclosed activity is then executed, all following activities are skipped. It is possible to specify an *otherwise* case which is taken if all conditions evaluate to false.

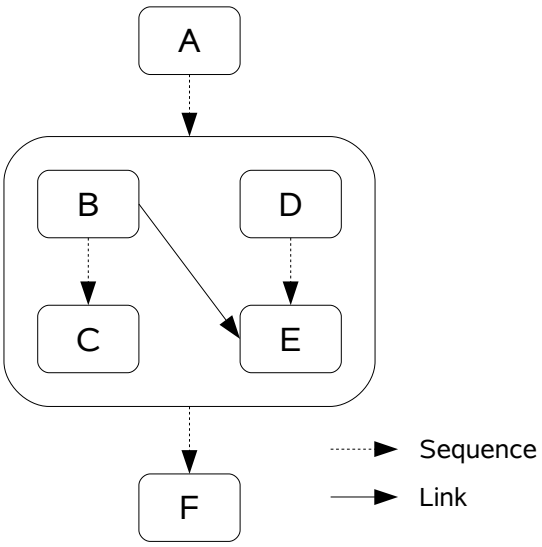

Figure 5.2: Flow Link graphical Example

```
activity_A
<flow>
    <links>
        <link name="BtoE" />
    </links>
    <sequence>
        activity_B
            <source linkName="BtoE
                " />
        activity_C
    </sequence>
    <sequence>
        activity_D
        activity_E
            <target linkName="BtoE
                " />
    </sequence>
</flow>
activity_F
```

Listing 5.13: Flow Link

While The while activity contains exactly one basic or structured activity which is performed repeatedly. It is executed until the given condition no longer holds true.

Pick The pick activity is similar to the switch activity, but the activity to be executed is selected based on incoming events. For every activity enclosed by pick an event is specified, either the arrival of a message (*onMessage*), or the expiration of a time alarm (*onAlarm*). Concurrent events are handled implementation-specific.

Flow The flow construct groups activities that are to be started in parallel. It is introduced to provide concurrency and synchronization. Every activity directly nested within it is started when the flow activity is started (see Listing 5.12).

```
<flow>
    <invoke partnerLink="..." />
    <invoke partnerLink="..." />
</flow>
```

Listing 5.12: Flow Activity

Furthermore the flow activity allows for the expression of synchronization dependencies between activities. This facilitates the definition of graph-like structures. *Links* are used to connect one and only one source activity to one and only one target activity. If the source activity has completed its execution the target activity can be started. Multiple incoming and outgoing links are allowed for one activity and the links can be asscociated with a transition condition. The definition of cycles is not permitted, this functionality is modeled using the while construct. Figure 5.1.2 shows an example of a link, listing 5.13 provides the description. The sequences B-C and D-E are executable concurrently, but activity E is not to be executed until activity B has not finished.

Correlation

If a BPEL process description specifies an executable process, it is intended to be executed by an execution engine. This engine is capable of interpreting BPEL code and creates process instances of the respective process description. However, messages specified to be sent between the process and collaborating Web Services are addressed only to the respective process description, not to the process instances.

It is very likely that there is more then one instance created of a business process description. Consider the example of an online book store. Several customers may contact the store at the same time, putting books into their respective shopping carts. For every customer, a separate process instance is created. The execution engine now needs a means to assign incoming messages to the proper process instances in order not to mix up the orderings.

For this BPEL offers the concept of *correlation sets*. It allows the specification of correlated groups of message exchanges whithin a process instance. The name of the set as specified in listing 5.14 indicates the correlation set which is expected to be contained in the messages sent and received.

```
1 <invoke partnerLink="Buyer" portType="BuyerPT" operation="addToBasket">
2     <correlations>
3         <correlation set="BookStorePurchase" pattern="out" />
4     </correlations>
5 </invoke>
```

Listing 5.14: Correlation Set

```
1 <correlationSets>
2     <correlationSet name="BookStorePurchase" properties="customerID" />
3 </correlationSets>
```

Listing 5.15: Definition of a Correlation Set

Listing 5.15 shows the definition of a correlation set. The element *correlationSets* is a direct child of the top-level element *process*. The properties are aliases for XPath 1.0 query strings that define paths within a message to the respective value. These values are considered key values for this communication relationship, identifying each message uniquely. By means of them the execution engine is able to correlate incoming and outgoing messages accordingly.

5.2 WS-CDL

5.2.1 Introduction

WS-CDL, the Web Services Choreography Description Language, is a language for describing the choreography of Web Services. As introduced in chapter 3 a choreography specifies the ordering of peer-to-peer interactions between multiple participating services without a central point of control. WS-CDL is specified purely as a descriptive language and is not aimed at providing input for any kind of execution engine [KBR+05]. It is developed by the W3C Choreography working group [cho].

The authors of WS-CDL use the phrase “common collaborative behavior” to characterize the content of a WS-CDL choreography description [KBR+05]. A WS-CDL specification can be seen a

contract between partners which collaborate to achieve a common business goal. To overcome weaknesses of a formulation in a natural language (e.g. ambiguity), WS-CDL offers constructs for a precise description of partners and their interactions, as seen from a global viewpoint. Each participant is then responsible for conforming to this contract by developing or adjusting its services accordingly. WS-CDL promises improvement in interoperability, easy determination of conformance, and distinctness in the definition of collaborations.

The position of WS-CDL in the Web Services stack can be seen in two different ways. Firstly, it is associated to the same layer as BPEL (see figure 5.3). This is reasonable if one looks at what is choreographed. WS-CDL composes Web Services which are described using WSDL as BPEL does. Secondly, WS-CDL can be seen as an additional layer above the orchestration layer. This is also appropriate because choreography arranges previously orchestrated services and may so be seen as a new layer (see figure 5.3).

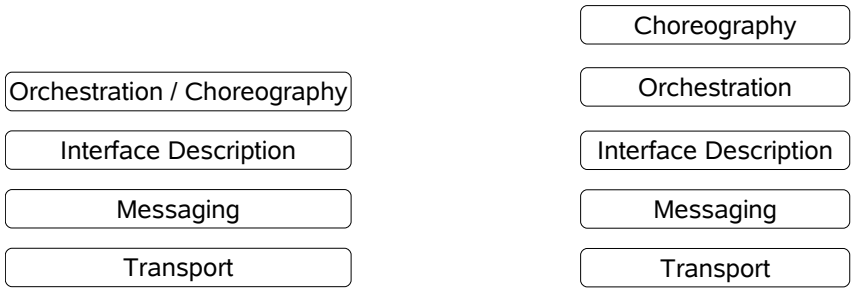

Figure 5.3: Web Services Stack

5.2.2 Structure

In this section we consider the structure of a WS-CDL document. A choreography description of a business collaboration consists of:

- a WS-CDL document, describing participants and interactions
- optional interface descriptions of the participating services, using for example WSDL

WS-CDL does not prescribe a certain interface description language like WSDL, it is also possible to use for example JAVA interface descriptions. Also it is not required for interface descriptions to be existent. To get an overview of the structure of a WS-CDL document, figure 5.4 shows a conceptual depiction. The description of a WS-CDL choreography is contained in a package which has several attributes characterizing the choreography. Name, author, and version are descriptive attributes useful for the developer of a choreography. *Information types* allow for the definition of types for variables used in the choreography. For example the type of a whole message can be denoted with an information type. *Token* and *token locator* are mechanisms that allow for the correlation of channel communication. A token locator defines a query that is used to extract a token from a message. A token thereby denotes a particular part of exchanged messages.

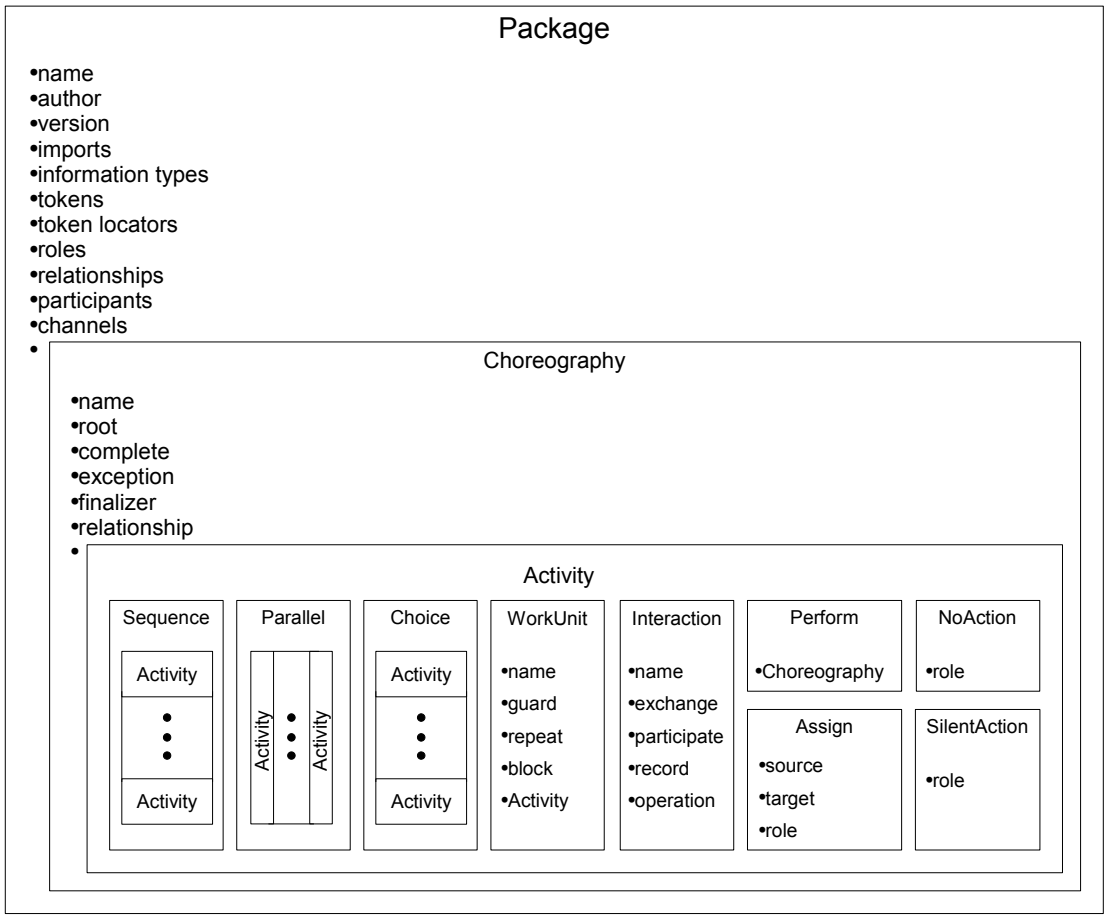

Figure 5.4: WS-CDL Structure Overview [BDO05]

Interactions

WS-CDL provides the ability to model peer-to-peer interactions between participants as seen from a global point of view. Each participant represents a requester of a service and at the same time is a provider of another service.This leads to multi-participant service dependencies. Within a WS-CDL document, the constructs *RoleType*, *RelationshipType*, *ParticipantType*, and *ChannelType* are used to model collaborations between participants.

RoleType A roleType denotes one of several potential observable behaviors of a participant. As in Listing 5.16 a participant can for example exhibit the behavior of a buyer, which interface is described in BuyerBehaviorInterface. This interface description may be given using WSDL, but WS-CDL does not prescribe that language.

```
1 <roleType name="BuyerRole">
2     <description type="documentation">
3         Role for Buyer
4     </description>
5     <behavior name="BuyerBehavior" interface="BuyerBehaviorInterface">
6         <description type="documentation">
7             Behavior for Buyer Role
```

```
8           </description>
9       </behavior>
10  </roleType>
```

Listing 5.16: roleType

RelationshipType A relationshipType describes exactly two roles that interact within a choreography. The example listing 5.17 shows two roles which are connected by the Buyer2Seller relationship, indicating their intention to exchange messages.

```
1  <relationshipType name="Buyer2Seller">
2      <description type="documentation">
3          Buyer Seller Relationship
4      </description>
5      <roleType typeRef="BuyerRole" />
6      <roleType typeRef="SellerRole" />
7  </relationshipType>
```

Listing 5.17: relationshipType

ParticipantType A participantType associates one or more roles to a particular entity. This is for example used for sharing information between different roles of one participant. This is similar to multi-threaded programming, where different threads of a system process share and access the same memory space.

```
1  <participantType name="Buyer">
2      <description type="documentation">
3          Buyer Participant
4      </description>
5      <roleType typeRef="BuyerRole" />
6  </participantType>
```

Listing 5.18: participantType

ChannelType A channelType represents the principle mechanism for the realization of interactions. It specifies where and how information is exchanged. Example 5.19 shows a channel located at the role of the Seller. The Seller provides a service which is accessible through the Buyer2SellerChannel and which is intended to be used by a Buyer. The address of the service is given by the reference element. The identity element allows for the unique identification of each channel instance.

```
1  <channelType name="Buyer2SellerChannel">
2      <description type="documentation">
3          Buyer to Seller Channel Type
4      </description>
5      <roleType typeRef="SellerRole"/>
6      <reference>
7          <token name="URI"/>
8      </reference>
9      <identity type="primary">
10         <token name="id"/>
```

```
11      </identity>
12 </channelType>
```

Listing 5.19: channelType

Activities

The activities which are possible inside a choreography container are parted in three types: *basic* activities, *ordering structures*, and *WorkUnit* activities. Basic activities describe the lowest-level activities inside a choreography and are either:

Interaction Interacting activities form the building block of a choreography. This is captured by the interaction activity which results in information exchange between participants. Possible actions are *request* and *respond*.

Perform The perform activity allows for the embedding of a complete, seperately defined choreography. It facilitates the reuse and combination of existing choreographies into new ones. There exists a mechanism for binding the caller's variables to the free variables in the performed choreography.

Assign The assign activity is used to create a new value for a variable or to assign the value of one variable to another variable, but only within one roleType.

noAction Performing nothing is noted with the noAction activity. An activity may syntactically be necessary, but no action is intended to be performed.

silentAction The silentAction activity indicates that some activity is performed and that the global observable behavior will be influenced, but the actual implementation is not revealed.

The *ordering structures* allow the specification of the control flow

Sequence The sequence ordering structure enables its enclosed activities sequentially, in lexical order.

Parallel The parallel activity encloses one or more activities that are performed in parallel. It completes successfully when all contained activities complete successfully.

Choice The choice ordering structure contains one or more activities. Only one of them is performed when the choice activity is enabled. The choice depends on two types of guard conditions: *Data-driven* choices are based on a condition that variables are evaluated immediately. *Event-driven* choices cause the performance of a choreography to hold until one of a set of specified events occurs, or to wait for certain variables to be populated.

The *workUnit* activity groups one or more activities into a single unit and specifies guard and repeat expressions. The guard condition determines whether a workUnit is performed at all, whereas the repeat condition decides whether the enclosed activities are repeated upon being once completed. If the guard condition is evaluated to false, the whole workUnit and so all enclosed activities are skipped. A special block attribute specifies whether the guard and repeat conditions are to wait for variables to become populated for evaluation.

5.3 OWL-S

5.3.1 Introduction

OWL-S is the acronym for Web Ontology Language for Web Services which aims at connecting the Web Services world with the Semantic Web. The sections above introduced BPEL and WS-CDL as languages for the composition of Web Services. Based on WSDL interface descriptions those languages specify participants and the control and data flow of their interactions. The resulting description may then be the input of an execution engine, running through the single steps of the newly created service. Though the steps of a composition are executed by a computer, they are not semantically interpretable for it. There is no possibility in those languages to semantically describe the functionalities of the created services.

OWL-S aims at making Web Services computer-interpretable, providing sufficient information for automated reasoning. Thereby it is used for the description of provided services as well as of services sought for by some entity. OWL-S is expected to facilitate the automated discovery, incovation, and composition of Web Services. For this OWL-S provides means to semantically describe the prerequisites of a Web Service, the single steps it is composed of, and the way it is to be contacted.

5.3.2 Structure

OWL-S offers a standardized ontology for the description of Web Services. The structuring of the ontology given below follows three questions [Mar06]:

- *What does the service require and provide?* This question is answered by the *Service Profile*.
- *How does it work?* This question is answered by the *Service Model*.
- *How is it used?* This question is answered by the *Service Grounding*.

According to these questions, OWL-S provides semantic information on a Web Service in three parts which are shown in figure 5.5. Below we give a description of each part.

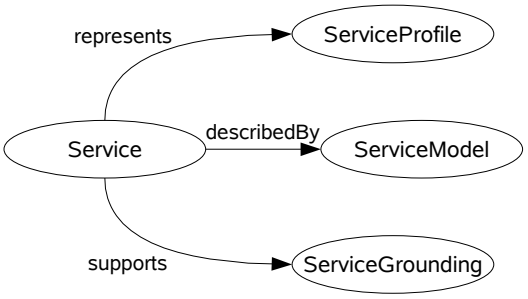

Figure 5.5: Top Level of OWL-S Ontology [Mar06]

Service Profile

A Service Profile is used to represent the capabilities that a service provides or that a service-seeking entity requests. It tells "what the service does." Three types of information are given in an OWL-S Service Profile: provider information, functionality information, and additional characteristics. Information on the provider of a service includes contact information referring to technical or business representatives of the providing entity. Information on the functionality of a provided service is an essential component of an OWL-S Service Profile. It is represented by two aspects: the information transformation (given as input and output), and the change in state (given as preconditions and effects). Additional characteristics may include attributes as quality or category of a service.

Service Model

A Service Model tells "how a service works." It gives further information on the functionality of a service. This includes, in case of a composite service, the single steps that are carried out during the execution of a service. This description may be used by potential requesters of the service for an in-depth analysis or for the composition with other services. OWL-S models a service as a process, so a process is at the top of the process ontology (see figure 5.6). The following paragraphs deal with the constructs that OWL-S offers for the semantic description of a process.

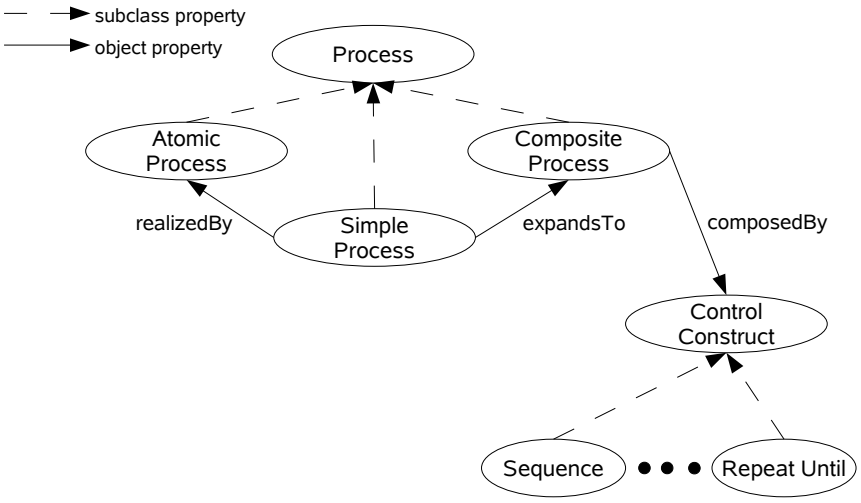

Figure 5.6: Top Level of Process Ontology [Mar06]

Atomic Process The principle of an Atomic Process is that is has no sub-processes and executes in a single step from the viewpoint of a service requester. The requester sends an input message, awaits the execution of the service, and receives the output message, without gaining any insight into the execution of the service. A grounding must be provided for each atomic process, as they are directly invocable. Listing 5.20 shows an example of an atomic process.

```
1 <process:AtomicProcess rdf:ID="PutInCart">
2     <rdfs:comment>
3         A very simple version of the put-in-cart process
4     </rdfs:comment>
```

```
5     <process:hasInput>
6         <process:Input rdf:ID="PutInCartItemID">
7             <!-- specification of the input -->
8         </process:Input>
9     </process:hasInput>
10 </process:AtomicProcess>
```

Listing 5.20: Atomic Process

Composite Process A Composite Process consists of other (composite or non-composite) processes. The way of composition is specified by constructs which are described below. A composite process can be represented by a simple process (see below), thus hiding the composition information. This is expressed by the property *expandsTo* in case of the simple process, and *collapsesTo* in case of the composite process.

Sequence The Sequence construct specifies a list of processes which are to be executed in the given lexical order. The enclosed components may be processes or control constructs.

Split Processes or control constructs enclosed by a Split construct are to be executed concurrently. OWL-S does not make any specification on waiting or synchronisation.

Split+Join The Split+Join construct offers synchronization capability through a barrier notion. A portion or all processes previously started by a split may be synchronized, that is, wait for the completion of their activities.

Unordered The Unordered construct requires the completion of all enclosed processes, thus providing synchronization of them. The processes may be executed in any order or concurrently.

Choice The Choice construct controls both process and execution. Its properties *chosen* and *chooseFrom* specify for example: Choose processes from *chooseFrom* and execute the *chosen* processes in sequence or parallel, or choose m (chosen) from n (choseFrom).

If-Then-Else The semantic of the If-Then-Else construct is equivalent to that of imperative programming languages. In the example provided on the OWL-S website the condition for the decision is expressed using SWRL[4] [MAB+].

Iterate Repeated execution of processes is done through the Iterate construct. The specific execution is described by attributes *whileCondition* or *untilCondition*.

Repeat-Until and Repeat-While These constructs are subclasses of the Iterate construct, with the respective attribute already specified.

Simple Process A Simple Process denotes also the single-step execution. Unlike an atomic process, it is not invocable and is not associated with a grounding. A simple process can be used as an abstraction in two ways: it may be used as a view on an atomic process, or it may represent a composite process by hiding the composite structure. The properties *realizedBy* (for an atomic process) and *expandsTo* (for a composite process) describe the respective case.

[4]Semantic Web Rule Language

Service Grounding

A Service Grounding answers the question how a service is to be used. That is, it gives concrete information on how an entity can access a service. This typically includes the specification of a communication protocol, message formats, and other details that are necessary for contacting a service. An OWL-S grounding has the central function of showing how to realize the abstract inputs and outputs of an atomic process as concrete messages. Therefore, OWL-S offers a mapping between a grounding and a WSDL interface description. Other interface description languages may also be used. Both OWL-S grounding and WSDL are complementary as they are not completely covering the same concepts. Figure 5.7 shows the overlap and difference of the languages. WSDL has no means to express the semantics of its services, and OWL-S does not allow for the expression of any concrete binding information.

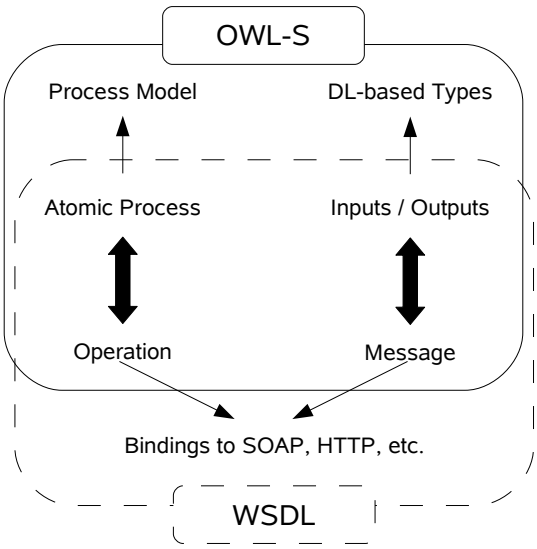

Figure 5.7: Overlap between OWL-S and WSDL [Mar06]

5.4 Summary

In this chapter we have introduced prominent representatives of different areas of Web Service composition. We first considered BPEL and its constructs for modeling business processes from a single businesses point of view. We then continued with WS-CDL, an upcoming language for describing collaborations of Web Services from a global point of view. Lastly we dealt with OWL-S, a language which is aimed at providing a basis for extensive automation of Web Service discovery, invocation and composition.

CHAPTER

SIX

Comparison of the Selected Languages

In this chapter we examine the languages BPEL, WS-CDL, and OWL-S. We start with the goals of these languages and then deal with the means they provide for the modeling of a business process. Thereby we firstly have a look at the basic ordering structures and then go on with more advanced ordering patterns. Lastly we address the problem of message correlation, that is, correlating messages to process instances and also other messages.

6.1 Goals of the Languages

BPEL

The main goal of BPEL is the ability to express a business process in a standardized way and by using a commonly accepted language defined for that. Two categories for a standardized description can be identified here:

- **Application Integration:** A long running business transaction consists of several single interactions between business entities. Consider a customer that wants to buy a book from an online bookshop. The WSDL description of the bookshop's Web Service does not (and cannot) provide information on the correct invocation order of the operations of that Web Service. The customer needs to know for example that the login operation has to be performed before the addItemsToCart operation can be invoked. BPEL allows for the specification of this ordering of operations. For this, a process can be described as an *abstract process* in BPEL, revealing the ordering of the operations but hiding actual implementation details. This abstract process definition enables an entity wanting to collaborate with this Web Service to conform to the ordering and integrate the Web Service into its own business processes. This abstract process description corresponds to the *local viewpoint* of section 3.4.1.
- **Execution:** The ultimate goal of formalizing business processes is to execute their steps. For this, the abstract process description is augmented with implementation details for each operation. BPEL not only allows for a specification of the correct order of operations to be invoked, but also for a precise description of the implementation of these operations. The

aim is to create a detailed description so that it can be executed directly by an execution engine capable of interpreting the description. This viewpoint is referred to in section 3.4.1 as *internal viewpoint*.

WS-CDL

WS-CDL is aimed at the precise description of collaborating participants and their interactions in order to achieve a higher goal. A WS-CDL document is meant to be a global contract between the participants specifying which participants interact with which and what the message exchanges will be that they engage in. 2 categories of specification can be discerned:

- **Business Relationships:** If participants collaborate to achieve a higher goal the business relationships between them need to be defined. Consider the example of a customer, bookshop, and parcel service. The customer orders books from the bookshop, the bookshop commissions shipment from the parcel service, and the parcel service delivers the ordered books to the customer. So there will be the following business relationships: customer/shop, shop/parcel service, and parcel service/customer. The WSDL specification aims at describing these business relationships. These relationships are described from a global viewpoint without a bias towards any of the participants.
- **Collaboration:** Additionally to the specification of the relationships between participants of a business process, the rules of collaboration are defined. These rules represent the commonly observable message exchange behavior of the respective participants. For every relationship there are the possible observable orderings of the exchanged messages described. This is similar to the abstract process definition of BPEL as there is also only the publicly observable behavior defined. But in contrast to the local viewpoint of BPEL, WS-CDL tatkes up the *global viewpoint* as described in section 3.4.1

OWL-S

The goal of the language OWL-S is somewhat different than that of WS-CDL and BPEL. Whereas these languages provide means to specify the single steps of a composition and the participating entities, the aim of OWL-S is to make Web Services computer-interpretable. That is, adding semantic information to the functionality, the description of the interface, and the single steps of a process.

OWL-S is expected to enable the following tasks:

- **Automatic Web Service discovery:** Automatic Web Service discovery denotes the automatic location of Web Services. Consider the example of a customer that wants to buy an airline ticket between two cities and wants to pay with a particular credit card. Currently, the search for an appropriate Web Service that satisfies the constraints of the customer has still to be done by humans. OWL-S aims at specifying properties and capabilities of Web Services in a computer-interpretable way to facilitate automatic discovery of services guided by given constraints.
- **Automatic Web Service invocation:** The invocation of a service should be as automated as the discovery. Until now the invocation of the airline ticket selling service has still to be

programmed by humans, because they can easily adapt to prerequisites like input format of a given service. OWL-S aims at facilitating the automatic determination of such prerequisites, in order to be able to adapt to and realize the service invocation automatically.

- **Automatic Web Service composition and interoperation:** Automatic Web Service composition denotes the automated achievement of a high-level goal. The customer of the previous example may want to fly from Frankfurt to New York and may want to pay with a certain credit card. For each of these sub-goals, an adequate Web Service should then be automatically discovered, invoked, and the results composed to reach the overall goal.

Summary

The three languages have complementary goals. As WS-CDL allows for the description of participants and commonly observable behavior from a global point of view without a central controller, it facilitates the definition of a choreography (see section 3.2). BPEL provides for the definition of a process and its interactions from the viewpoint of a single participant, thereby following the approach of orchestration. As described in section 3.4.1, both languages can complement one another for a complete description of a business collaboration in order to achieve a higher goal.

Firstly, it is possible to start with the elaboration of a WS-CDL description of a business collaboration. Once this is specified one can generate templates of the local viewpoints of the process for each participant from the global description, expressed using BPEL. Then these templates, or abstract process definitions, are added with the actual logic of the respective process. This results in an executable process description in BPEL for every participant.

Secondly, it is possible to start with already given BPEL abstract process definitions and use WS-CDL to combine them into new business collaborations. Thereby one merge the single allowed orderings of message exchanges into a commonly observable message exchange behavior.

OWL-S supports the first approach by providing semantic descriptions of the Web Services to be combined. This facilitates easy discovery of Web Services that needed to realize a particular choreography. It may for example help in finding all Web Services that provide book selling capabilities. The second approach is supported by OWL-S in the semantic description of the single steps of a process. This allows for an automation of the creation of business collaborations.

6.2 Ordering Structures

As the single steps of a business process are arranged like the single instructions in imperative programming languages, languages for the description of a business process have to provide similar ordering structures. In the following we introduce the basic ordering structures offered by the languages BPEL, WS-CDL, and OWL-S.

Sequence In imperative programming languages there normally is no explicit ordering structure to express the sequential execution of steps. This is denoted implicitly by the lexical ordering of the instructions. However, in most business process description languages there exists a special construct for specifying sequential execution. In all 3 languages BPEL, WS-CDL, and OWL-S it has the name *sequence* and the same semantics. That is, if the sequence construct gets enabled,

then all enclosed steps are executed in lexical order. The construct is regarded as being completed when the last step is completed.

Parallel In imperative programming languages the technique of parallel execution is relatively hard to grasp, since concurrency is realized only implicitly. Business process description languages however typically offer a construct which explicitly denotes that its enclosed activities are to be performed in parallel.

The language BPEL provides therefore the *flow* construct. Its semantics is as follows: if the flow activity gets enabled, then all contained activities are executed concurrently. A flow is considered completed when all activities within it have completed. Within the flow construct it is possible to model dependencies and synchronization (graph-like parts) of a business process by using *links*. The link construct is explained in section 5.1.2.

WS-CDL provides the *parallel* construct for modeling concurrent execution of activities. The semantics is the same as in BPEL. That is, on enablement all contained activities are performed in parallel, and the activity is completed when all activities have completed.

OWL-S offers two possibilities for modeling concurrent behavior. The *Split+Join* construct has the same semantics as described above, that is, it is not completed until all activities have completed. Additionally, OWL-S provides the construct *Split* with different semantics. This construct is considered completed when all activities just have been started.

Choice The choice construct is also known from imperative programming languages. It tests several cases in lexical order and takes the first one whose condition evaluates to true. BPEL offers for the modeling of that the construct *Switch*. This activity consists of one or more case elements and may be followed by an otherwise element. The case elements, which have an attribute *condition*, are considered in lexical order. The first element whose condition evaluates to true is executed, the remaining element are ignored. If no condition holds true, the otherwise element comes into account. If it is missing, it is assumed to be there with an enclosed empty activity.

For the modeling of a choice WS-CDL provides the *Choice* construct. Within this element there are workunits with guard conditions specified. The first workunit whose condition holds true is performed. If there are no guard conditions given it is assumed that the decision on which workunit to take is non-observable [KBR+05].

OWL-S also provides a choice construct to choose between several possible activities. To realize the aforementioned semantics, it is necessary in OWL-S to specify to choose “1 from n”, as it is also possible to choose “m from n”. Note that in OWL-S there is no lexical ordering of the elements within the choice construct, as they are seen as a set. So one cannot rely on lexical ordering if more than one condition hold true. This can be avoided by specifying disjunct conditions.

While The while construct of imperative programming languages allows for the repeated execution of particular activities. As the same is often necessary in business process descriptions, business process description languages offer a respective construct. BPEL provides the *While* construct for this. When the activity is enabled, the enclosed activity is executed repeatedly until a condition given as an attribute no longer holds true.

In WS-CDL this behavior is realizable by the use of a workunit which has attributes *repeat* and *guard*. If this attribute are specified, the workunit is repeated until the condition given by the attribute no longer evaluates to true. Note that it is necessary to specify also the guard condition, otherwise the workunit is executed at least once.

OWL-S offers the *Iterate* construct to model repeated activity. The number of repetition is specified using the attributes *whileCondition* or *untilCondition*.

6.3 Control Flow

The specification of the control flow of a business process is a major part in business process modeling. In section 4.4.2 we introduced workflow patterns that facilitate the modeling of standard situations in business processes. In this section we deal with the realization of the workflow patterns in the languages BPEL, WS-CDL, and OWL-S. We follow Wohed et al. in the description of BPEL patterns that are not directly supported in BPEL [WvdADtH02] and give our own proposals for WS-CDL and OWL-S where applicable. We depart from actual syntax in some listings for simplicity.

.

Pattern "Sequence" The sequence pattern denotes the execution of activities given in their lexical order. BPEL provides two possibilities for the realization of this pattern. On the one hand one can use the sequence construct which is inherited from XLANG (see listing 6.1), on the other hand one can use the link construct inherited from WSFL (see listing 6.2). If two activities are connected by a link, the target activity is enabled after the completion of the source activity.

```
<sequence>
    activity_1
    activity_2
</sequence>
```

Listing 6.1: "Sequence" in BPEL Sequence

```
<flow>
    <links>
        <link name="aLink" />
    </links>
        activity_1
            <source linkName="
                aLink" />
        activity_2
            <target linkName="
                aLink" />
</flow>
```

Listing 6.2: "Sequence" in BPEL

WS-CDL offers the same sequence notation as BPEL (see listing 6.3).

```
<sequence>
    activity_1
    activity_2
</sequence>
```

Listing 6.3: "Sequence" in WS-CDL

OWL-S provides a class *Sequence* which allows for the notation of a list of control constructs to be done in order.

Pattern "Parallel Split" and "Synchronization" The Parallel Split pattern allows for performing activities simultaneously or in any order. Synchronization provides a mechanism that waits for all activities to complete their actions before the next activity is enabled.

BPEL offers two ways to realize this pattern. The first solution uses a sequence to denote the activity that is to follow the synchronized activities, the second solution provides synchronization by the use of links. In listing 6.4 activity_3 is executed after the flow activity completed because of the enclosing sequence construct. In listing 6.5 activity_3 is the target of two links coming from activity_1 and activity_2. Therefore, activity_3 is not performed until both link sources are completed.

```
<sequence>
    <flow>
        activity_1
        activity_2
    </flow>
    activity_3
</sequence>
```

Listing 6.4: "Parallel" in BPEL

```
<flow>
    <links>
        <link name="L1 "/>
    <link name="L2 " />
    </links>
    activity_1
        <source linkName="L1" />
    activity_2
        <source linkName="L2" />
    activity_3
        joinCondition="L1 AND L2
            "
        <target linkName="L1" />
        <target linkName="L2" />
</flow>
```

Listing 6.5: "Parallel" in BPEL

The solution in WS-CDL is the same as the first one of BPEL. One can realize this pattern by using a parallel construct and a sequence construct that ensures synchronization (see listing 6.6) .

```
<sequence>
    <parallel>
        activity_1
        activity_2
    </parallel>
    activity_3
</sequence>
```

Listing 6.6: "Parallel Split" and "Synchronization" in WS-CDL

For the realization of the Parallel Split and Synchronization pattern OWL-S provides the classes *Split* and *Split+Join*. The class Split describes the concurrent execution of its components, whereas the class Split+Join additionaly provides synchronization of the enclosed components.

Pattern "Exclusive Choice" and "Simple Merge" The Exclusive Choice pattern provides a means to chose one of several possible branches within a process. The Simple Merge pattern

denotes a point in a process at which previously split branches merge. It is assumed that only one branch is actually active.

BPEL again has two possible solutions, relying either on the switch or on the flow construct. The switch construct encloses case elements associated with conditions. The first case branch which condition evaluates to true is taken when considered in lexical order. This realizes the Exclusive Choice. Simple Merge is modeled by the fact that activity_3 is executed after the switch construct (see listing 6.7). The other solution again makes use of the flow and link constructs as shown in listing 6.8. An empty activity in the beginning is the source of links to each branch activity. Every link is associated with a condition. That link is then followed that has a condition that holds true. The activityC in the end represents the Simple Merge pattern.

```
<switch>
    <case condition="C1">
        activity_1
    </case>
    <case condition="C2">
        activity_2
    </case>
</switch>
activity_3
```

Listing 6.7: "Exclusive Choice" and "Simple Merge" in BPEL

```
<flow>
    <links>
        <link name="L1" />
        <link name="L2" />
        <link name="L1s" />
        <link name="L2s" />
    </links>
    <empty>
        <source linkName="L1"
            transitionCondition=
            "C1" />
        <source linkName="L2"
            transitionCondition=
            "C2" />
    </empty>
    activityA1
        <target linkName="L1">
        <source linkName="L1s">
    activityA2
        <target linkName="L2">
        <source linkName="L2s">
    activityC
        joinCondition="L1s OR
            L2s"
        <target linkName="L1s">
        <target linkName="L2s">
</flow>
```

Listing 6.8: "Exclusive Choice" and "Simple Merge" in BPEL

WS-CDL provides the choice construct to realize the Exclusive Choice pattern. Synchronization is achieved as in the BPEL example by an activity following the choice construct. The choice is based on conditions given in enclosed workunits. If there are no workunits with associated conditions given then it is assumed that the decision is non-observable [KBR+05].

```
<choice>
    activity_1
    activity_2
</choice>
```

```
5 activity_3
```

Listing 6.9: "Exclusive Choice" and "Simple Merge" in WS-CDL

The class *Choice* of OWL-S allows for the description of the Exclusive Choice pattern. A single control construct of the enclosed components is chosen for execution. The Simple Merge pattern is realized by a following activity as in BPEL and WS-CDL.

Pattern "Multiple Choice" and "Synchronizing Merge" The Multiple Choice pattern describes a point in the process where not only one but several branches can be taken and executed in parallel. The Synchronizing Merge pattern then recombines the previously split process. The realization in BPEL is done using flow and link constructs. If the conditions associated with the links are not disjunct, more than one link can be followed (see listing 6.8). Synchronization is achieved by the OR joinCondition. The use of switch is not possible because there only the first case whose condition holds true is taken.

The Multiple Choice pattern can be achieved in WS-CDL using the parallel construct (see listing 6.10). Thereby the enclosed workunits are given non-disjunct conditions. So, more than one workunit is performed. Synchronization is given because the parallel activity is not completed until the last enclosed activity is completed.

```
1 <parallel>
2     <workunit guard="cond_1">
3         activity_1
4     </workunit>
5     <workunit guard="cond_2">
6         activity_2
7     </workunit>
8 </parallel>
9 activity_3
```

Listing 6.10: "Multiple Choice" and "Synchronizing Merge" in WS-CDL

The Multiple Choice and Synchronizing Merge patterns can be described in OWL-S using the *Split+Join* and the *If-Then-Else* classes. Every branch inside the Split+Join class is enclosed within a If-Then-Else class. Initially all branches are enabled, but only those are executed whose condition evaluate to true. The Join part of Split+Join realizes the Synchronizing Merge.

Pattern "Multiple Merge" The Multiple Merge pattern allows for the merging of multiple concurrently executed branches of a process. This pattern requires the repeated execution of activities within one process. BPEL, WS-CDL, and OWL-S do not provide a solution to this pattern, because they do not support the repeated instantiation of an activity within one process instance.

Pattern "Discriminator" The Discriminator pattern models a point in the process that listens on multiple incoming branches and executes the following activity when the first branch completes. The remaining branches are ignored until all branches completed, then it is started again. BPEL does not provide an activity to model this pattern. Using a link construct is also not possible, because the joinCondition waits for all incoming links and is then evaluated. The same situation is given in WS-CDL. The blocking attribute of a workUnit cannot be used, because if it is set to

false, there is no waiting at all for any incoming branch. OWL-S does not either provide support for the modeling of this pattern.

Pattern "Arbitrary Cycles" The Arbitrary Cycle patterns models control flow paths in a process in a "GoTo"-like way. It is possible to create loops, jumps, or shortcuts in any way. Since BPEL, WS-CDL, and OWL-S do not provide goto-functionality, there is no possibility of modeling this patterns in these languages.

Pattern "Implicit Termination" The Implicit Termination pattern is used to denote the ending of a specific path, without having to merge it with an exit point. A modeling in BPEL of this pattern is possible by the use of flow and link constructs. An activity that is not a source of a link can end without a particular termination activity. In WS-CDL this pattern can be realized using a sub-choreography and a non-blocking perfrom activity for the sub-choreography. There is then no waiting for the sub-choreography to end. The description of this pattern in OWL-S is possible by using the *Split* class, since this component completes as soon as all enclosed components have been started.

Pattern "MI without Synchronization" This pattern allows to spawn off new threads of control. In BPEL this is realizable using the receive activity with its attribute createInstance set to "yes" (see listing 6.11). This leads to the creation of new instances on each arrival of a message. A corresponding invoke activity has to determine whether there are new instances necessary.

```
<receive createInstance="yes">
    <!-- activities -->
</receive>
```

Listing 6.11: "MI without Synchronization" in BPEL

In WS-CDL the pattern MI without Synchronization can be realized by performing a sub-choreography. The activity to be performed without synchronization is put into a sub-choreography. This is then performed using the *perform* activity with its attribute *block* set to "false". This means that the sub-choreography is enabled and the enclosing perform activity is completed immediately after that. An enclosing workunit with a repeat condition may determine the number of instances (see listing 6.12)

```
<choreography name="act_1">
    activity_1
</choreography>

<choreography>
    <!-- preceeding activities -->
        <workunit repeat="moreInstancesNeeded">
            <perform choreographyName="act_1" block="false" />
        </workunit>
    <!-- subsequent activities -->
</choreography>
```

Listing 6.12: "MI without Synchronization" in WS-CDL

OWL-S allows for the description of the MI without Synchronization pattern by using the classes *Repeat-Until* and *Split*. The class Repeat-Until is responsible for looping as many times as

new instances are necessary, whereas the class Split is used to start a new activity without waiting for it to complete.

Pattern "MI with Synchronization" This pattern models a point in a business process at which a certain number of instances of a particular activity are created. Later, they are synchronized before the rest of the process is executed. 3 types are differentiated: The number is known at design-time, the number is known at run-time before the first instantiation takes place, and the number is unknown in advance during run-time. The solution for the first type is the same for BPEL, WS-CDL, and OWL-S. As the number is known at design-time, the respective activities can be duplicated according to the number and then be put into a flow construct (BPEL) or parallel construct (WS-CDL). The other two types are not directly supported in BPEL, WS-CDL, or OWL-S. For BPEL, Wohed et al propose a solution that uses a loop and a counter to realize the pattern (see listing 6.13). A pick activity inside a while activity waits for the arrival of three different messages: one message causes a new instance to be invoked (StartNewActivity), one message notifies on the completion of a previously invoked instance (ActivityFinished), and one message indicates that no new instances are needed.

```
1  moreInstances:=True
2  i:=0
3  <while moreInstances OR i>0>
4      <pick>
5          <onMessage StartNewActivity>
6              invoke activity_1
7              i:=i+1
8          </onMessage>
9          <onMessage ActivityFinished>
10             i:=i-1
11         </onMessage>
12         <onMessage NoMoreInstances>
13             moreInstances:=False
14         </onMessage>
15     </pick>
16 </while>
```

Listing 6.13: "MI with Synchronization" in BPEL

A realization of this pattern in WS-CDL is possible using a sub-choreography, the perform activity, and a synchronizing workunit. As shown in listing 6.14, a sub-choreography is used to model the activity which is to be instantiated multiple times. This sub-choreography is then performed by the main choreography. The workunit containing the perform activity is repeated based on the condition whether new instances are needed. The following workunit is set to be blocking, thus providing synchronization.

```
1  <choreography name="act_1">
2      activity_1
3  </choreography>
4
5  <choreography>
6      <sequence>
7          <workunit repeat="moreInstancesNeeded">
8              <perform choreographyName="act_1" block="false" />
9          </workunit>
```

```
10          <workunit guard="NoMoreInstancesNeeded" block="true" />
11      </sequence>
12      <!-- subsequent activities -->
13  </choreography>
```

Listing 6.14: "MI with Synchronization" in WS-CDL

In OWL-S a solution as the one for MI with Synchronization is not possible as we see it. OWL-S does not offer a Join construct that allows to wait for the created instances within the Split to synchronize.

Pattern "Deferred Choice" The Deferred Choice pattern is similar to the Exclusive Choice pattern, but the decision is based on incoming events. A realization in BPEL is possible by using the pick construct. This construct waits for the receipt of one of a number of messages and continues the execution according to the received message (see listing 6.15). It is not possible to have an explicit solution in WS-CDL nor in OWL-S for this pattern since there is no construct that waits for the arrival of messages.

```
1 <pick>
2     <onMessage operation="op_1">
3         activity_1
4     </onMessage>
5     <onMessage operation="op_2">
6         activity_2
7     </onMessage>
8 </pick>
```

Listing 6.15: "Deferred Choice" in BPEL

Pattern "Interleaved Parallel Routing" Every activity enclosed by a Interleaved Parallel Routing pattern is executed exactly once, but there is no order given. In BPEL there are two possible solutions for this pattern. Wohed et al propose to use the concept of serializable scopes (see listing 6.16). A serializable scope is a scope that has its attribute containerAccessSerializable set to yes. This ensures concurrency control on shared containers. For each activity that is to be interleaved, a different container is used. Each of those containers writes to a single shared container (container C). The fact that they are writing to the same shared container ensures that they perform sequentially, but the order is not given. Secondly it is possible to enumerate all possible interleaved patterns of the activities. The major drawback is however that this encreases exponentially with the number of activities (see listing 6.17). This solution is also the only possible way of realization in WS-CDL of this pattern.

```
<flow>
    <scope name=S1
        containerAccessSerializable:
        ="yes">
        <sequence>
            write to container C
            activity_1
            write to container C
        </sequence>
    </scope>
    <scope name=S2
        containerAccessSerializable:
        ="yes">
        <sequence>
            write to container C
            activity_2
            write to container C
        </sequence>
    </scope>
</flow>
```

Listing 6.16: "Interleaved Parallel Routing" in BPEL

```
<pick>
    <onMessage m1>
        <sequence>
            activity_1
            activity_2
        </sequence>
    </onMessage>
    <onMessage m2>
        <sequence>
            activity_2
            activity_1
        </sequence>
    </onMessage>
</pick>
```

Listing 6.17: "Interleaved Parallel Routing" in BPEL

OWL-S provides the class *Any-Order* for describing the Interleaved Parallel Routing pattern. Every enclosed control construct is to be executed exactly once but not concurrently.

Pattern "Milestone" The Milestone pattern denotes an activity whose enabling depends on a certain state. As there is no direct support of this pattern in BPEL, Wohed et al [WvdADtH02] propose a workaround as shown in listing 6.18. As long as activity_1 is not executed, the while loop ensures that activity_2 can be executed repeatedly. As WS-CDL and OWL-S do not offer a construct to realize a deferred choice, it is not possible to model this in these languages.

```
1_completed:="false"
<while 1_completed="false">
    <pick>
        <onMessage m2> activity_2
        </onMessage>
        <onMessage m1>
            <sequence> 1_completed:="true" </sequence>
        </onMessage>
    </pick>
</while>
activity_1
```

Listing 6.18: "Milestone" in BPEL

Pattern "Cancel Activity" The Cancel Activity Pattern allows for the cancellation of single activities within the execution of a process.

The modeling of this pattern in BPEL can be achieved by using its fault and compensation handler constructs. Termination of an activity is based on the termination of a scope. A fault is used to cancel the activities of the respective scope, and a compensation handler compensates begun or completed work within this scope.

In WS-CDL this pattern can be modeled in a similar way. Termination is carried out based on workUnits and the use of exception handling. The finalize activity of WS-CDL cannot be used because it is enabled only in case of a successful completion of the respective workUnit. OWL-S does not provide a class for modeling the Cancel Activity pattern.

Pattern "Cancel Case" The Cancel Case pattern provides a possibility to end a whole business process. BPEL offers the terminate construct to realize the Cancel Case pattern. All activities of the enclosing process are ended without fault or compensation handling. In WS-CDL it is possible to achieve this behavior by using an exception that is caught at the outermost nesting level. An exception that is not caught is propagated outwards until the outermost scope is reached, which is then the whole choreography. Again, OWL-S does not provide a class for modeling the Cancel Case pattern.

Summary

BPEL, WS-CDL, and OWL-S provide the basic constructs that are necessary for modeling all basic business processes. That is, they offers constructs for the specification of sequential, concurrent, and repeated execution of activities as well as for branching based on conditions. However if it comes to specifying complex business processes where particular patterns like the aforementioned are needed, there is some lack of support.

Neither of the languages supports merging of multiple branches that would lead to multiple instantiations of the merging activity and all following activities. Conceptually, as described in section 4.4.2, there would be multiple "tokens" arriving at the merging activity, all of them enabling this activity. All of them would also be passed to the following activities.

The pattern Arbitrary Cycles is not provided by any of the languages. One reason is probably that the definition of an arbitrary cycle with go-to between different scopes would make compensation handling very difficulty, even unrealizable.

The support of a Milestone pattern would be helpful just in WS-CDL. Scenarios are conceivable that require the ability to make certain that all participants have reached a particular state.

The lack of support for Cancel Case and Cancel Activity in OWL-S prevents the specification of a premature cancellation of a whole process or parts of it. An service seeking agent may base its decision on the suitability of a Web Service on the capability to cancel a certain activity or the whole process.

6.4 Data Flow

The specification of the flow of data is also a part of the modeling of a business process. As we described in section 4.4.3, the flow of data may follow the same structure as the flow of control, or it may depart from it. Furthermore one has to discern between data belonging to activities

of a single role or of different roles. In BPEL the flow of data is realized through the exchanged messages. The activities invoke, receive, and reply have atttributes *inputVariable* and *outputVariable*, specifying the data to be sent and the data to be received. In case of an asynchronous invocation, only the input variable for the invocated operation is given as there is no synchronous response expected. The specification of a data flow that departs from the control flow is not possible because for the exchange of data always one of the control flow constructs invoke, receive, or reply has to be used.

In WS-CDL data consumed and produced at the same role is implicitly visible to all activities of that role. However, roles do not share data, unless they are grouped to one participant, then the data is visible to all those roles [RTF]. The explicit modeling of data flow in WS-CDL is possible by using the respective containers of the *interaction* activity. More specifically, the send and receive elements within the *exchange* element of the interaction activity contain an attribute *variable* which denotes the data to be exchanged. Departing from the control flow is not possible in WS-CDL as for a data exchange there always has to be a message exchange which is modeled using the control flow construct *interaction*.

OWL-S also provides a means to model the flow of data. The object property *valueSource* is used for referencing of produced data of other steps of the business process description. This allows for the specification of a data flow that is separate from the control flow.

6.5 Participants

Business Process Modeling not only describes the interaction patterns of Web Services but also deals with the description of the participants offering the Web Services and their roles in these interactions (see section 4.4.4). A participant is modeled in WS-CDL using the construct *participantType*, which groups roles onto a particular entity. In BPEL this grouping is achieved by the construct *partner*.

The concept of roles and their relationship is realized in WS-CDL by the constructs *relationshipType* and *roleType*. A roleType in WS-CDL describes a particular behavior of a participant, a relationshipType gives the interaction relation between them. Accordingly, this is modeled in BPEL using the construct *partnerLinkType*. This partnerLinkType can be derived from the WS-CDL relationshipType, and its attribute *role* from the WS-CDL roleType [MH06].

OWL-S does not support the modeling of relationships between interacting participants. Up to now the Service Model in OWL-S allows only for the description of the single steps of a process, but not of participants and their relationships.

6.6 Transactions

We have described two concepts of transactions in section 4.4.5. On the one hand there are transaction of short duration for which the ACID properties hold. On the other hand there are long running business transactions for which ACID properties are not applicable. The BPEL specification does not provide a means for supporting short or long running business transactions. It has support for the compensation of the effects of long running business transactions, but only within a single business process instance (see section on compensation below). There are no

constructs that ensure compensation of activities across participants. For example if the booking of a flight failed, there is no construct that allows for automatic compensation of the already booked hotel and car. As BPEL relies on the Web Services stack and its infrastructure, it is possible to use the WS-Transaction standard in a generic manner to realize transactional behavior for both short or long running distributed transactions. This is not dealt with in the specification, because it is noted to be outside its scope [ACD+].

As WS-CDL aims at specifying choreographies from a global or local viewpoint (see section 3.4.1, it does not provide a means for modeling transactions of short duration that would be handled internally. For the modeling of distributed transactions or long running business transactions it relies on the infrastructure layer, namely the standard WS-Transaction. It if of course possible to model transactional behavior such as a two phase commit protocol by explicitly specifying the single steps in WS-CDL, but this would lead to confusing choreography descriptions and is not the intended purpose of WS-CDL.

There is no support provided by OWL-S for the description of transactional behavior.

6.7 Compensation

During the execution of a business process it is sometimes required to be able to undo certain parts of the process. This is modeled by the concept of compensation (see section 4.4.6). BPEL offers a direct solution to the handling of compensation by the construct *compensationHandler*. Every scope can be associated with a compensation handler, thus allowing for the defintion of compensation spheres (see section 4.4.6). Inside the compensation handler there are activities secified that compensate the activities of the respective scope. A compensation handler in BPEL can be invoked after the scope completed normally by using the *compensate* activity, which has to be invoked from another scope. This activity has an attribute that names the scope to be compensated.

It is possible to model the compensation concept in WS-CDL using the activity *finalize*. This activity enables a finalizerBlock inside successfully completed immediately enclosed choreographies. The activities enclosed in this finalizeBlock are then the action necessary to undo the effects of this choreography. A drawback of using this as a compensation mechanism is that it is only possible to finalize whole choreographies. If it is necessary to finalize only parts, they have to be put into sub-choreographies.

6.8 Exception Handling

The handling of exceptions deals with occurring situations in the process that are not within the normal flow of the process. Both BPEL and WS-CDL offer an exception handling mechanism. Exception handling in BPEL is called fault handling. As with compensation handling, a fault handler can be associated with every scope. The handler then deals with occurring faults in case there is a case specified for that exception, otherwise it propagates the exception to the next outer nesting level. This concept is similar to that of imperative programming languages, which offer concepts like “try-catch” and also propagate uncaught exceptions to the next level.

A choreography description in WS-CDL may have an exceptionBlock associated with it. Within this block at least one exception workunit is defined specifying the exception it catches and the activity to be performed. Exception handling is only possible for a whole choreography, if it is required for parts of it, they have to be put into sub-choreographies.

The description of the handling of exceptions is not supported in OWL-S.

6.9 Message Correlation

A business process description is a static specification of participants and the single steps of their interactions. For actual execution the description is fed into an execution engine which creates instances of the process description and actually performs the single steps. The instances of the process description communicate with their partner Web Services through the exchange of messages. The addressing of these messages however is based on the WSDL ports of the recipient Web Service, not on the single communicating instances. As a description may be instantiated concurrently several times, the engine managing these instances forwards incoming messages to the correct instances, which is known as *instance routing* [ACKM04]. When a messages then reaches the correct instance, this instance in turn has to be able to correlate the incoming message to previously outgoing messages. For example one has to be able to tell apart which message is a response to which request. In this section we deal with this requirement which is known as *messages correlation*. We show 3 different kinds of messages correlation that we identified and give possible solutions for them.

Throughout this section we will use the example of a business relationship involving a customer, a shop, and a parcel service. A possible collaboration between the customer and the shop may be as shown in figure 6.1. The customer performs login into the shop, browses for certain products, and adds one or more items to his virtual shopping cart. In the end the customer concludes his shopping and confirms the ordering of the selected items in his shopping cart. The shop in turn confirms the receipt of the order. The order transaction between the customer and the shop has ended, but the shop and the parcel service, as well as the customer and the parcel service will engage in further interactions. The shop requests the delivery service of the parcel service, thereby transferring goods and transmitting relevent data like the address of the customer. The parcel service commences delivery and notifies the customer upon successful dispatchment.

Every interaction like login or getItemList is carried out by the exchange of messages. Conceptually these are exchanged between the respective instances of the process descriptions. The instances then need the ability to correlate incoming and outgoing messages in order to realize the communication patterns specified in the process descriptions. The Web Services Architecture working group of W3C [wsab] defines message correlation as follows [wsaa]:

> Message correlation is the association of a message with a context. Message correlation ensures that a requester agent can match the reply with the request, especially when multiple replies may be possible.

It is possible to have implicit or explicit realizations for the requirement of message correlation. An implicit solution would rely on capabilities of an underlying protocol, for example the request-response mechanism of HTTP which allows for the correlation of single responses to single requests. An explicit solution would be the usage of message identifiers allowing for the correlation of messages based on it [wsaa].

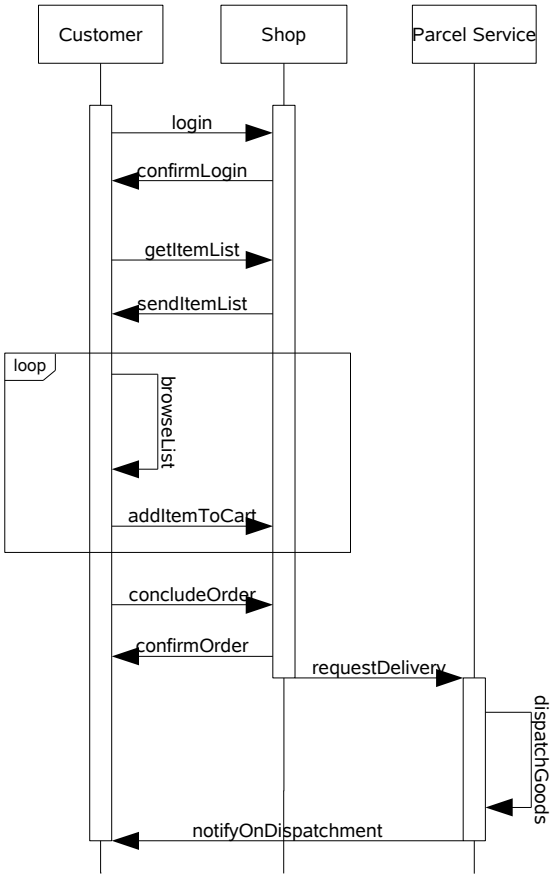

Figure 6.1: Book Shop Example

Figure 6.2 shows the conceptual responsibilities of instance routing and message correlation. The execution engine receiving the message is responsible for routing the message to the proper instance of the process description. This instance is then responsible for putting the incoming message into the right context. In the following we stick to this conceptual point of view for the purpose of a simple description of message correlation. In fact however, both instance routing and message correlation is handled by the execution engine, as the instances are run by the execution engine.

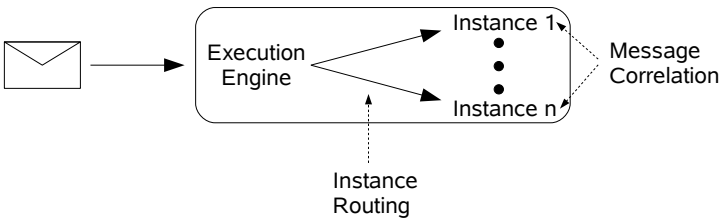

Figure 6.2: Instance Routing and Message Correlation

6.9.1 Approaches of Message Correlation

On the basis of the introductory example we explain 3 types of correlation as we identified them. We deal with the correlation of single request-response messages, the correlation of messages to one conversation, and with the correlation of different conversations. In parallel we introduce possible solutions.

Correlation of single Request-Response Messages

The request-response messaging pattern consists of an outgoing message and a related incoming message. To determine this relationship, that is, which incoming message belongs to which sent message, there is a mechanism necessary. Consider the shop example. The customer performs a login into the shop. This usually consists of two messages. One message, sent by the customer, contains the data required for the login procedure. The other message sent by the shop contains confirmation data on the successful login. The customer now needs to relate the confirmation message to the login message. A possible solution is to rely on the protocol binding of Web Services. They use SOAP as the protocol for exchanging information, which in turn uses HTTP as its primary application layer protocol. HTTP allows for the correlation of single request-response messages by its GET or POST functionality. Thereby the request message is sent encoded within the URL (in case of GET) or encoded within POST data, whereas the response message is then the HTTP response to the GET or POST method, respectively.

Correlation of Messages to one Conversation

A conversation between two participants of a business process may consist of several message exchanges. These messages need a mechanism that allows to relate them to one conversation. Consider again the shop example. Several customers may login at overlapping times, each representing a different conversation with the shop, and add items to their respective shopping cart. The shop needs then to be able to relate incoming messages, like addItemToCart, to the proper conversation.

HTTP binding is not sufficient for the solution of this correlation, as it provides only a mechanism to relate single request-response messages. As messaging relies on the Web Services Stack, it is possible to use WS-Addressing for this purpose. It provides a transport-neutral mechanism for addressing Web Services and identifying messages for the purpose of message correlation. WS-Addressing defines two elements for the header of a SOAP message. The element *messageID* allows for the assignment of a unique identifier to each message, and the element *relatesTo* facilitates the reference to previous messages (see Listing 6.19). These two elements can be used to correlate several message exchanges to one conversation. For example a lookup table is used whose entries are the identifiers of the conversation and the respective message identifiers of the corresponding messages. The scenario could be as follows: The customer generates an ID for its conversation with the shop and enters it into his lookup table. The login message is then correlated to this conversation by matching the ID of the message to the ID of the conversation in the lookup table. The relatesTo element of the incoming login confirmation message then contains this message ID, which can be again correlated to the conversation by looking it up in the table. In this way, it is possible to correlate several message exchanges to one conversation. It may be necessary to keep a messageID stored in the lookup table for more than one reply, in case there may be later replies also relating to this message.

```
<s:Envelope
    xmlns:s="http://www.w3.org/2003/05/soap-envelope"
     xmlns:wsa="http://www.w3.org/2005/08/addressing">
    <s:Header>
        <wsa:MessageID>http://example.com/someotheruniquestring</
            wsa:MessageID>
        <wsa:RelatesTo>http://example.com/someuniquestring</wsa:RelatesTo
            >
        <wsa:To>http://example.com/business/customer</wsa:To>
        <wsa:Action>http://example.com/shop/confirmOrder</wsa:Action>
    </s:Header>
    <s:Body>
        <!-- some content -->
    </s:Body>
</s:Envelope>
```

Listing 6.19: WS-Addressing Example

Correlation of different Conversations

Additionally to the correlation of messages to one conversation there is also a need to correlate conversations between different participants. Consider again the shop example. The customer confirms the ordering of the selected goods. The shop then commissions the parcel service to deliver the ordered goods. The parcel service in turn delivers the goods to the customer. The customer/shop conversation and the shop/parcel service conversation have to be correlated in order to ensure delivery of the ordered goods to the proper customer. In case the customer respresents a larger organization and there are multiple simultaneous orders from the same shop, the customer/shop conversation and the parcel service/customer conversation also have to be correlated, ensuring that the customer can associate the deliveries to the proper ordering. As the example indicates, correlation patterns can become quite complex.

WS-Addressing is not sufficient fo this kind of correlation, because the relatesTo mechanism does not provide a means to correlate to messages across conversations. In order to relate to its purchase the customer would require the parcel service to provide a relatesTo ID of the conversation customer/shop. This is not possible with the relatesTo mechanism of WS-Addressing, because the parcel service has no knowledge on the message IDs of the conversation customer/shop. The parcel service is only able to relate to messages of his own conversations, but not to messages of conversations of third party. A possible solution is the use of additional identifiers inserted into the body of the SOAP messages. The identifiers do not necessarily have to be the same for different conversations. The customer/shop conversation may use identifiers like customerID, whereas the shop/parcel service conversation may use identifiers like commissionID. The correlated exchanges may also nest or overlap, so that messages may carry identifiers of different conversations at the same time. For example the messages of the customer/parcel service conversation may contain identifiers representing a certain commission and a certain customerID of the customer, taken from the customer/shop conversation. Such a solution would then be specific to the application and to the Web Services composition standard used. It would be in the responsibility of the description of the business process to specify a means for message correlation. The developers of a choreography of business processes have to plan which messages need to carry which tokens in order to realize proper correlation. Note that the correlation of single request-response messages as well as the correlation of messages to one conversation can also be addressed by this last mechanism. As we

describe in the next paragraph, the languages BPEL and WS-CDL offer means for this kind of correlation.

6.9.2 Message Correlation in Composition Languages

In this section we deal with the realization of message correlation offered by the composition languages BPEL and WS-CDL. They both provide mechanisms that allow for the aforementioned kind of application specific correlation. They introduce identifiers to be transmitted in each message, and expressions for the extraction of these identifiers out of the messages. OWL-S however does not provide for the specification of message correlations.

Realization in BPEL

BPEL provides a declarative mechanism for the specification of correlated groups of messages. For all messages in the respective correlation group, a set of *correlation tokens*, a *correlation set*, is defined which is shared by all those messages. The tokens reference those parts of the messages that contain the correlation data. A correlation set is defined within a scope and is visible only to that scope and all enclosed scopes. Correlation sets are specified within the BPEL document (see listing 5.1). Listing 6.20 shows a definition of a correlation set, specifying that messages belonging to the correlation set "PurchaseOrder" have to carry a property named "customerID".

```
1 <correlationSets>
2     <correlationSet name="PurchaseOrder" properties="customerID" />
3 </correlationSets>
```

Listing 6.20: Definition of a Correlation Set

The activities invoke, receive, and reply as well as the onMessage branches of pick activities and the onMessage branches of event handlers can contain correlation set names. If there is no correlation set given, it is assumed that either an underlying protocol sees for the correlation or no correlation is needed. Listing 6.21 shows the example of a receive activity that makes use of a correlation set named "PurchaseOrder". This indicates, which identifiers are expected to occur in messages handled by this receive activity. The attribute *initiate* indicates that the correlation is started by this activity. This means that the sending customer populates the correlation set with a customerID and the receiving shop adopts the customerID from the message. In this case, the customer is called *initiator* of this correlation, and the shop is called *follower*.

```
1 <receive partnerLink="Customer" portType="OrderPT">
2     <correlations>
3         correlation set="PurchaseOrder" initiate="yes" />
4     </correlations>
5 </receive>
```

Listing 6.21: Use of a Correlation Set

The position of the correlation tokens relative to some other elements (typically the parents) within a message is fixed for all messages belonging to a particular correlation set. It is specified by a query expression, where BPEL relies on the language XPath. This means that a sender of a message needs to know where the receiver expects the tokens to be placed. Listing 6.22 shows the definition of the path to the element representing the property "customerID".

```
<propertyAlias propertyName="customerID"
        messageType="POMessage" part="PO"
        query="/PO/CID" />
```

Listing 6.22: Definition of a Property

Realization in WS-CDL

Message correlation can be realized with WS-CDL using the *channelType* construct. It is the principle mechanism for realizing communication relationships between participants. The initiator of a conversation always contacts the respective other service via its correspondent channelType (see figure 6.3). In the shop example, the customer contacts the shop by using the channelType

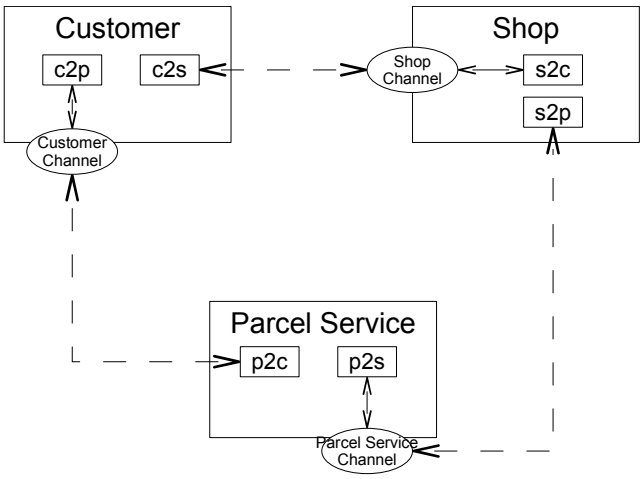

Figure 6.3: WS-CDL channelType communication

"Customer2ShopChannel" offered by the shop. The relationship is defined at the customer by the "c2s" relationshipType and at the shop by the "s2c" relationshipType.

```
<channelType name="Customer2ShopChannel">
    <description type="documentation">
        Customer to Shop Channel Type
    </description>
    <roleType typeRef="ShopRole"/>
    <reference>
        <token name="http://www.example.com/shop"/>
    </reference>
    <identity usage="primary">
        <token name="customerID"/>
    </identity>
</channelType>
```

Listing 6.23: Definition of a Channel

Listing 6.23 shows the definition of the channel "Customer2ShopChanel" located at the shop. The reference element refers to the respective Web Service to be used. The element *identity* is important for the correlation of messages. It allows for the association of a unique identity to each instance of a channelType. The set of tokens given within the *identity* element is seen as primary key for

instances of that channelType [RTF]. As for every conversation a new channelType instance is created, this can be used for the correlation of messages. For the usage attribute of the identity element there are 4 values possible. *Primary* indicates that the specified set of tokens is primary key for messages exchanged on that channel. *Alternate* means that this set of tokens may be used as a new primary key in subsequent exchanges, once a message has contained both primary and alternate sets of tokens. *Derived* is used in addition to Primary and denotes that the channel containing the derived set of tokens as primary key is correlated with this channel. This is used to correlate different channels. Listing 6.24 shows the correlation of the channels OrderChannel and DeliveryChannel by the usage of derived. The value Association of the attribute usage allows for the correlation of this channel instance to a previous instance of the same channel type.

```
<channelType name="OrderChannel">
    ...
    <identity usage="primary">
        <token name="OrderId" />
    </identity>
    <identity usage="derived">
        <token name="DeliveryId" />
    </identity>
</channelType>

<channelType name="DeliveryChannel">
    ...
    <identity usage="primary">
        <token name="DeliveryId" />
    </identity>
</channelType>
```

Listing 6.24: Correlation of Channels

The part of a message that contains a token is specified using a *tokenLocator* which is defined within the WS-CDL document as shown in listing 6.25. If an identity is defined using more than one token, for example “customerID” and “orderID”, this is considered as a composed primary key. Then there has to be given a separate tokenLocator for every token.

```
<tokenLocator tokenName="customerID" informationType="purchaseOrder"
    query="/PO/CID" />
```

Listing 6.25: Definition of a tokenLocator

A token itself is specified as in listing 6.26. It acts as an alias for a fragment of a message that is used in a choreography.

```
<token name="customerID" informationType="xsd:int" />
```

Listing 6.26: Definition of a token

6.10 Summary

This chapter has highlighted the capabilities of the languages BPEL, WS-CDL, and OWL-S. We have first dealt with the common and complementary goals of these languages. BPEL allows for orchestration, that is, it provides a means to model business processes from the viewpoint of a

single participant. WS-CDL facilitates the modeling of a choreography, that is, it offers constructs for the specification of a business process as seen from a global perspective. OWL-S is intended to support both choreography and orchestration by providing semantic descriptions of Web Services that allow for automatic discovery, invocation, and composition.

Secondly we have considered the languages BPEL, WS-CDL, and OWL-S in relation with workflow patterns that have been identified to appear regularly when modeling business processes. The languages have sufficient support for the specification of basic and most advanced business processes, but lack some capabilities that are reported to be necessary for modeling some complex workflow patterns.

The last part of this chapter has dealt with the correlation of messages. We have presented 3 types of correlation that we identified: correlation of single request-resonse messages, correlation of messages to one conversation, and correlation of different conversations. In parallel, we have given possible realizations of these correlation types. We then introduced the solutions that the languages BPEL and WS-CDL offer.

CHAPTER

SEVEN

Conclusion

7.1 Summary

The realization of business processes using Web Services plays a major role in an increasing number of organizations. Based on these Web Servcies it is possible to create new functionality by combining them into a new course of action. This combination of Web Services is generally called composition. Specialized languages for web services composition are developed and already in use that facilitate this realization. They provide constructs for the modeling of participants and their interactions and sequencing them. After a quick overview of the nature of Web Services we have examined the basic terms that are used when describing the composition of Web Services. The purpose of this part of the thesis was to clarify the meaning of the different terms and their relationships. The three most important terms among them are the following:

- **Composition:** Composition considers generally the assembling of participants and denoting of their exchanged messages. It is also seen as a way to master the complexity of business processes as there are complex Web Services realized by combining several simpler Web Services.
- **Orchestration:** Orchestration refers to one participant of a business process and the modeling of its message exchanges.
- **Choreography:** A choreography takes into account all participants of a business process and the modeling of all message exchanges.

Related to these terms are the points of view one can take up in considering composition. A description from the *global viewpoint* takes into account all participants and the ordering of their exchanged messages. The *local viewpoint* is then a projection of these orderings onto a single participant, describing the allowed sequences of messages from that participant's point of view. These two viewpoints describe only the publicly observable behavior of one or all participants, respectively, but do not reveal internal details. In contrary, when considering composition from an *internal viewpoint*, the internal realization is specified.

The modeling of business processes is part of the larger concept of Business Process Management. We have delineated the other phases, automation, monitoring, analyzation, and improvement order to bring modeling in relation to them.

We have addressed principles that languages should provide for the modeling of business processes. They should allow for specifying control flow, data flow, participants, transactions, compensation, and exception handling. On the basis of these principles we have examined the capabilities of the three languages BPEL, WS-CDL, and OWL-S. All basic and most advanced control flow constructs are either directly supported by the three languages or emulatable. Only few quite sophisticated control flow patterns are not representable. The languages BPEL and WS-CDL also provide constructs for modeling participants, transactions, compensation, and exception handling, whereas OWL-S lacks this capability.

Special attention has been paid to the correlation of messages as this remains somewhat unclear throughout specifications. Thereby we have firstly differentiated between instance routing, which is the correct delivery of messages to the destined process instance, and message correlation, which denotes the correct correlation of incoming and outgoing messages. Then we have examined three types of message correlation that we identified from publications. They are outlined in the following:

- **Correlation of single Request-Response Messages:** A process instance needs to correlate incoming and outgoing messages in order to know which message is a response to which message sent. A solution is possible by using the HTTP request-response mechanism.
- **Correlation of Messages to one Conversation:** A conversation between partners typically lasts for several message exchanges. The partners have to be able to correlate different incoming messages to a conversation with a particular partner. For this purpose the standard WS-Addressing can be used. It offers a header field *relatesTo* that facilitates this kind of correlation.
- **Correlation of different Conversations:** In more advanced business process the conversations of different partners often are interrelated. As this kind of correlation is highly dependent on the business process the solution is provided at the application layer. The languages BPEL and WS-CDL both offer means to model such message correlations. Up to now OWL-S does not provide for the modeling of message correlations.

7.2 Outlook

In this thesis the modeling of business processes has been considered by ordering their activities using constructs like sequences, loops, and branches. Branching takes place based on conditions that are specified within the process description for each branch connection. A moderately complex business process can easily consist of several branches each associated with particular conditions. This leads to complex process descriptions both in their textual syntax and graphical representations as all potential paths are to be modeled. The logic behind such a process description is then often difficult to grasp. Additionally, in case the conditions have to be changed the process description itself has to be modified since the conditions are defined within it.

To overcome those difficulties that come along with the modeling of decisions within a business process description, a different approach is used in some of todays applications. There Business

Rules are used for controlling the aspects of a business process. However, those applications focus on effectively applying Business Rules to avoid complex process descriptions and more often than not lack support of modeling these businesss processes in a specialized language. They typically rely on imperative programming languages like Java for modeling the process flow. In so doing the basis of decision making is easily changeable if a proper rules repository is used, whereas a change in the process flow is relatively expensive. For this, it would be necesssary to rewrite and deploy the respective code, e.g. written in Java, instead of just altering the process description modeled in a specialized language.

The authors of [Eni05] denote both approaches to be highly related, to be "just different sides of the same coin". A process could be regarded as an implementation of rules, and rules are used for decision making within processes. The software vendor Oracle argues that the combined usage of Business Rules and Business Process Management will result in reduced application modification costs and increased agility. That is, changes to an application can very rapidly be delivered into production [Ora].

There are different solutions to the combined usage of Business Process Management and Business Rules in developement:

- **Loose Coupling:** One possible solution to the combination of processes and business rules is according to the authors of [Enix] the usage of both a Business Process Management System and a Business Rules Environment. The integration of these tools is often described as the approach of "loose coupling", a concept where minimal assumptions are made on the interacting partners. The benefit of doing so is that one can apply process and business rules functionalities for Business Process modeling while sticking to established products of both areas.

  The downside however is that there are still two separate applications. Their distinct object models and data models need to be synchronized in order to being able to work together. This means that developers need to have knowledge on internals of the respective other system, somewhat breaking the premise of loose coupling. For the alteration of a business process, there will be changes necessary in both applications, doubling the amount of work.

- **BPM and Business Rules with a combined Object Model:** The authors of [Eni05] envision an application with merged BPM and Business Rules functionality that will eliminate the disadvantages of the above solution. The rules and process definitions are stored in the same repository and are based on a common object model. Thereby, business rules are not only to be used for better decision making, but to drive the entire Business Process Management System. By using one object model for business rules and processes the need for synchronization does not apply. This ensures consistency between business rules and processes.

- **Rule-based Process Modeling:** Another feasable solution is the modeling of business processes solely based on rules. That is, not only branching decisions but the whole process flow is modeled using rules. A promising approach is the application of reactive rules, which control the dynamics of a business process. Within the area of reactive rules, the interest in rules following the pattern of event - condition - action (ECA) is gaining momentum. That is, on an incoming event and under a given condition a certain action is executed.

  XChange is such a language based on the concept of ECA rules. It allows for the reaction on events and the communication of events with other Web Services. Events are exchanged for example using SOAP for message formatting. As the authors of [BEPR06] show, it is

possible to realize a Web Service by modeling its business process using this rule-based language. However there are also some difficulties to be overcome. These include the notion of state, as ECA rules usually do not maintain state information for the single process instances. A closely related problem is the realization of message correlation. This is difficult to realize using XChange ECA rules. One reason is the event-centric approach of ECA rules as opposed to the activity-centric approach. The last one allows for a easy specification of activity sequences, for example for realizing message correlation, whereas the first approach would require complex creation of events for this. A possible solution may be the usage of an infrastructure layer that provides services like message correlation, but the actual business logic is further specified by rules.

7.3 Conclusion

We have investigated the composition of Web Services, its application to business process modeling, and the comparison of the composition languages BPEL, WS-CDL, and OWL-S. Through this we arrive at the conclusion that development in the field of Web Services composition is widely advanced in the case of BPEL, and is in an initial state in the case of WS-CDL and OWL-S, but in full activity. The combination of business processes and business rules is a promising approach in advancing business process modeling. Even more the realization of business processes purely by rule-based languages is a challenging direction of research.

BIBLIOGRAPHY

[ACD+] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. Business Process Execution Language for Web Services Version 1.1. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel.

[ACKM04] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services: Concepts, Architectures and Applictions*. Springer-Verlag, 2004.

[All01] Rob Allen. *Workflow: An Introduction*. Workflow Management Coalition, 2001. www.wfmc.org/information/Workflow-An_Introduction.pdf.

[amh04] Answers.com. The American Heritage® Dictionary of the English Language, Fourth Edition, Houghton Mifflin Company, 2004. http://www.answers.com.

[ans06] Answers.com. Britannica Concise Encyclopedia, Encyclopædia Britannica, Inc., 2006. http://www.answers.com.

[Bar] Martin Bartoniz. BPMS, BPML, BPEL, BPMN, WMS, XPDL. http://www.bpm-guide.de/articles/17.

[BDO05] Alistair Barros, Marlon Dumas, and Phillipa Oaks. A Critical Overview of the Web Services Choreography Description Languages (WS-CDL). *BPTrends Newsletter*, 3(3), March 2005. http://www.bptrends.com/deliver_file.cfm?fileType=publication&fileName=03-05%20WP%20WS-CDL%20Barros%20et%20al.pdf.

[BEPR06] François Bry, Michael Eckert, Paula-Lavinia Patranjan, and Inna Romanenko. Realizing business processes with eca rules: Benefits, challenges, limits. In *Principles and Practice of Semantic Web Reasoning, 4th International Workshop*, pages 48–62, 2006.

[bmi06] Business Modeling & Integration Domain Task Force, 2006. http://bmi.omg.org.

[cho] W3C Choreography Working Group. http://www.w3.org/2002/ws/chor/.

[DS05] Schahram Dustdar and Wolfgang Schreiner. A survey on web services composition. *International Journal of Web and Grid Services*, 1(1):1–30, 2005.

[Dus03] Schahram Dustdar. Service Composition. Technical report, 2003. http://www.unitn.it/convegni/download/icsoc03/panel/P_P2_Dustdar.pdf.

[Eni05] Business Rules are from Mars & and Processes from Venus. 2005. www.enix.co.uk.

[GM06] Giuseppe De Giacomo and Massimo Mecella. Automatic Web Service Composition. In *IEEE International Conference on Services Computing (SCC 2006)*, 2006.

[Hav05] Michael Havey. *Essential business process modeling*. Theory in practice. O'Reily, 2005.

[KBR+05] Nickolas Kavantzas, David Burdett, Gregory Ritzinger, Tony Fletcher, Yves Lafon, and Charlton Barreto. Web Services Choreography Description Language Version 1.0. Candidate recommendation, W3C, November 2005. http://www.w3.org/TR/ws-cdl-10/.

[LK06] Beate List and Birgit Korherr. An evaluation of conceptual business process modelling languages. In *Proceedings of the 2006 ACM Symposium on Applied Computing (SAC)*, pages 1532–1539, 2006.

[LR00] Frank Leymann and Dieter Roller. *Production Workflow - Concepts and Techniques*. Prentice Hall PTR, Upper Saddle River, New Jersey, 2000.

[MAB+] David Martin, Anupriya Ankolekar, Mark Burstein, Grit Denker, and Daniel Elenius. OWL-S 1.1 Release. Technical report. http://www.daml.org/services/owl-s/1.1.

[Mar06] David Martin. OWL-S: Semantic Markup for Web Services. Technical report, 2006. http://www.daml.org/services/owl-s/1.0/owl-s.html.

[MH06] J. Mendling and M. Hafner. From WS-CDL Choreography to BPEL Orchestration. Technical report, 2006. http://wi.wu-wien.ac.at/home/mendling/publications/TR06-CDL.pdf.

[MM04] Nikola Milanovic and Miroslaw Malek. Current solutions for web service composition. *IEEE Internet Computing*, 8(6):51–59, November 2004.

[mw06] Merriam-Webster Online Dictionary, 2006. http://www.m-w.com.

[OHE99] R. Orfali, D. Harkey, and J. Edwards. *Client / Server Survival Guide*. Wiley, 1999.

[Ora] Oracle Business Rules. 2007, http://www.oracle.com/technology/products/ias/business_rules/index.html.

[RTF] Steve Ross-Talbot and Tony Fletcher. Web services choreography description language primer version 1.0. Technical report, W3C. http://www.w3.org/TR/2006/WD-ws-cdl-10-primer.

[SOA] W3C XML Protocol Working Group. http://www.w3.org/2000/xp/Group.

[Sta04] Stallings. *Operating Systems, internals and design principles*. Prentice Hall, 2004.

[UDD] UDDI Version 3.0.2. http://uddi.org/pubs/uddi_v3.htm.

[vdAW01] Wil M. P. van der Aalst and Mathias Weske. The P2P Approach to Interorganizational Workflows. In *Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE'01)*, pages 140–158, 2001.

[web] Webopedia Computer Dictionary. `http://www.webopedia.com/TERM/W/Web_services.html`.

[WfM06] Workflow Management Coalition, 2006. `http://www.wfmc.org`.

[Whi04] Stephen A. White. *Process Modeling Notations and Workflow Patterns*. The Object Management Group (OMG), 2004.

[Whi06] Stephan A. White. *Business Process Modeling Notation*. Business Process Management Initiative, 2006. `http://www.bpmn.org/Documents/OMG%20Final%20Adopted%20BPMN%201-0%20Spec%2006-02-01.pdf`.

[Wor] Workflow Management Coalition. *WfMC Terminology & Glossary*. `http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf`.

[wsaa] W3C Web Services Architecture Note. `http://www.w3.org/TR/ws-arch/`.

[wsab] W3C Web Services Architecture Working Group. `http://www.w3.org/2002/ws/arch/`.

[WSD] Web Services Description Working Group. `http://www.w3.org/2002/ws/desc`.

[WSG] Web Services Glossary. `http://www.w3.org/TR/ws-gloss`.

[WvdAD+05] Petia Wohed, Wil M.P. van der Aalst, Marlon Dumas, Arthur H.M. ter Hofstede, and Nick Russell. Pattern-based Analysis of BPMN. Technical report, 2005. `http://www.bpm.fit.qut.edu.au/projects/babel/docs/BPM-05-26.pdf`.

[WvdADtH02] Petia Wohed, Wil M.P. van der Aalst, Marlon Dumas, and Arthur H.M. ter Hofstede. Pattern Based Analysis of BPEL4WS. Technical report, 2002. `http://is.tm.tue.nl/staff/wvdaalst/publications/p175.pdf`.