SOA Efforts

Thomas Kohlborn, Marcello la Rosa

Abstract As the Service-oriented architecture paradigm has become ever more popular, different standardization efforts have been proposed by various consortia to enable interaction among heterogeneous environments through this paradigm. This chapter will overview the most prevalent of these SOA Efforts. It will first show how technical services can be described, how they can interact with each other and be discovered by users. Next, the chapter will present different standards to facilitate service composition and to design service-oriented environments in light of a universal understanding of service orientation. The chapter will conclude with a summary and a discussion on the limitations of the reviewed standards along their ability to describe service properties. This paves the way to the next chapters where the USDL standard will be presented, which aim to lift such limitations.

1 Introduction

Broadly speaking, a Service-oriented Architecture (SOA) is a paradigm for arranging and utilizing business capabilities and resources that may be under the control of different business domains [6], for the sake of producing business value. While SOA does not imply the use of technology, it can help structure how technology is deployed and organized within a particular organization, or across a consortium of organizations that need to interact with each other. That said, the underlying concepts of service-orientation are typically realized by developing technical (i.e. electronic) services that communicate with each other over the Web, as opposed to business
services, which may also be carried out manually. In this regard, SOA can also be seen as a principle for designing software architectures which revolves around the notion of “Web service”. A Web service is a self-contained, autonomous, reusable software component encapsulating discrete functionality, which is distributed and accessible over the Internet.

Thus, the term Web service refers to a specific technology approach for implementing a SOA, when the channel of communication is the Web. The following definition of Web service has been proposed by the World Wide Web Consortium in 2004 [24]:

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.

As SOA and Web services have been researched by academia and applied by practitioners for nearly a decade, multiple standards have emerged in this field. These standards propose how a service-oriented environment can be created, how basic service functionality can be described, and how services can be aggregated to provide high-level functionality. In this context, the major standardization bodies are the World Wide Web Consortium\(^1\) (W3C), the Organization for the Advancement of Structured Information Standards\(^2\) (OASIS) the Object Management Group\(^3\) (OMG) and The Open Group\(^4\). This chapter will describe the most mature and widely-used SOA standards promoted by these four consortia, and discuss the shortcomings of these standards when it comes to a more universal understanding of service properties.

The chapter is organized according to Figure 1, which classifies these SOA standards based on their level of abstraction. Accordingly, Section 2 will start by presenting two different styles for describing basic service functionality, i.e. WS-* and REST. These styles manifest themselves into two main description languages, namely WSDL and WADL, which have reached different levels of maturity. Section 2 will also discuss SOAP as a standard message protocol to allow communication among services, and its extensions to model non-functional aspects of a service such as WS-Addressing and WS-Security. Section 3 will describe two different mechanisms that can be used to store and discover Web services described in WSDL, namely UDDI and WS-Discovery. Next, Section 4 will overview two different specifications (WS-BPEL and BPMN) for compositing services according to two complementary models: orchestrations and choreographies. Focusing on the top-level of abstraction of the SOA Stack, Section 5 will present a UML meta-model (SoaML) and two reference models (SOA-RM and SOA Ontology) that can be used to design service-oriented architectures and describe their constituent elements. The

\(^{1}\) http://w3.org
\(^{2}\) http://www.oasis-open.org
\(^{3}\) http://www.omg.org
\(^{4}\) http://www.opengroup.org
chapter will conclude with a summary and a discussion on the shortcomings of these standards along their ability to capture service properties.

2 Service Description

There are two different architectural styles to describe Web services, namely WS-* and Representational State Transfer (REST). The WS-* style builds upon the idea that each service is accessible at one or more locations or endpoints, each described by a Unique Resource Identifier (URI). Each service encompasses multiple operations, which in turn means that each operation is service-dependent. A message that is exchanged between operations or services consists of some metadata (header) and the message body (payload). This architectural style is typically realized by using WS-* standards over SOAP as the messaging protocol. However, other protocols such as HTTP or SMTP may be used. In fact, WS-* services can operate on top of different protocols, which enables the delivery of infrastructure services, such as security, transactions, routing and reliability. In scenarios where a service contract describing the service is already available, this architectural style can be well uti-
ized. Nonetheless, on the downside it requires more specific infrastructure, which induces overhead and can easily be misused.

On the other hand, REST is a design idiom built around the idea of exposing services as resources identified by a URL, instead of exposing single service operations. The state of a resource can be manipulated by HTTP actions such as create, read, update and delete (i.e. HTTP PUT, GET, POST and DELETE). Data to/from a service is typically transmitted using plain old XML (POX) over HTTP, which has the advantage of being very simple and lightweight (other languages can be used in place of POX such as the JavaScript Object Notation – JSON). However, this paradigm depends solely on HTTP and lacks methods for producing well-described service contracts due to its early state of maturity. Moreover, RESTful Web services are completely stateless, which might be a disadvantage in long-running transactions.

This section will present WSDL as the standard language to describe Web services in the WS-* style, and provide a brief overview of WS-Policy, a specification to enrich WSDL documents with policy information. This discussion will be complemented by an overview of SOAP, the standard protocol for message exchange in this style. and of its most important extensions. The section will conclude with an overview of WADL—an alternative language to WSDL for describing RESTful services.

### 2.1 Web Services Description Language

The Web Service Description Language (WSDL) provides a means to describe a Web Service contract by using XML [16]. The specification is driven by the W3C and is currently published as a recommendation in version 2.0 [27], while version 1.1 [23] is a group note, which does not have the same level of W3C endorsement. WSDL allows one to invoke the service’s operations independently of the service’s implementation, thereby sharing characteristics of classical interface definition languages that are commonly found in middleware technologies. Instead of utilizing naming and directory services, as commonly found in a middleware environment, Web Services operate in rather decentralized environments. Therefore, in WSDL the specific location of the service needs to be explicitly defined, so that other services or applications can invoke the service’s operations.

A WSDL 2.0 document is divided into two parts, namely an Abstract part and a Concrete part as shown in Figure 2.

The Abstract part contains one or more Interface elements and the Types element. An Interface element describes a set of operations offered by the service including the required input and output messages for each operation. For example, a Supplier service may offer service operations to create quotes, to process orders and to generate invoices. Each of these operations needs an input document and may produce an output document in reply. For example, the operation to create quotes accepts ‘request for quote’ documents as input and produces ‘quote’ documents as output (synchronous operation), while the operation to process orders only accepts ‘order’
documents as input but does not produce any output document (asynchronous operation).

The Types element describes the data types of each message used in an Interface using XML Schema [31]. One can either specify their own (complex) data types, such as a type to describe the ‘quote’ or the ‘order’ documents, or use built-in XML Schema simple data types, such as String or Date. However, WSDL is not bound to XML Schema, and different data type languages may also be used by using extension elements.

The Concrete part defines the implementation details necessary to access the service. This part contains one or more Binding elements, and one or more Service elements. A Binding describes the mapping between the various messages specified in an Interface and a transmission protocol such as SOAP (for WS-* services) or HTTP (for both WS-* and RESTful services). It also describes the binding style for each message, e.g. Remote Procedure Call (RPC) or Document-style. A Service element is used to link a Binding with a service endpoint, i.e. the URI where the service can be accessed. Since multiple Service elements can be defined, a service may be accessible via different endpoints, and using different transmission protocols.

WSDL 2.0 provides the capability of modeling eight different message exchange patterns between a service and its consumer, enriching those available in WSDL 1.1. Four patterns (In-Only, Out-Only, Robust-In-Only, Robust-Out-Only) consist of a single message being sent by the consumer to the service or vice versa, which may be replied by a fault message in case of an error. Another four patterns (In-Out, Out-In, In-Optional-Out, Out-Optional-In) consist of an initial message sent to either party which is (optionally) replied by a correlated message. These message ex-

Fig. 2 Structure of a WSDL document
change patterns abstract out binding-specific information like timing between messages, whether the pattern is synchronous or asynchronous (an aspect which was instead predetermined in WSDL 1.1), and whether the message is sent over a single or multiple communication channels.

There are two generic approaches to develop Web services: code-first and contract-first. The former is a top-down approach requiring the development of classes in an object-oriented programming language, which can then be used to generate a WSDL document that describes the functionality offered by these classes as service operations. The latter is a bottom-up approach requiring the development of a WSDL document first, which is then used to create the skeleton of the required classes to be implemented. Existing web programming platforms offer ways to control the mapping between WSDL and object oriented languages. For JAVA, for example, open-source Web service platforms such as Apache AXIS 2 and Oracle Metro Stack can be used. An alternative in the .NET development environment is Windows Communication Foundation (WCF).

As an interface description language (IDL), WSDL offers a language-independent description of the structural aspects of a Web service, while at the same time being well supported by languages such as Java which need to provide or consume the operations described in a WSDL document. Nonetheless, WSDL is a very technical language which restricts its applicability to an IT audience only. Further, WSDL does not provide any capabilities to link service operations to business entities such as processes, objects and capabilities within an organization, nor does it provide any means to specify non-functional aspects of a Web service such as pricing, legal terms and conditions under which the service may be consumed, and service level agreements (response time, availability, etc.).

WS-Policy is a W3C recommendation [28] that aims to partly lift such limitations. It provides a mechanism for service providers to enrich WSDL artifacts with policies on various service aspects such as security or service level agreements. It can also be used by consumers to specify the requirements that must be met by a service provider in order for the consumer to use its services. However WS-Policy is too tightly-coupled to WSDL documents, and misses explicit connections to counterparts in the business domain. Moreover, a Web service policy is typically spread over different WSDL documents, so a coherent picture about what policy belongs to what kind of services is often hard to identify.

2.2 SOAP and Messaging Specifications

SOAP (Simple-Object Access Protocol) is an XML-based protocol for message exchange between Web services, which uses various underlying transport protocols such as e.g. HTTP, TCP or SMTP. SOAP can be regarded as an asynchronous way to support the exchange of messages between different parties by implementing request-response interactions out of multiple one-way interactions. SOAP defines the format of the exchanged messages as part of a transmission between sender, re-
ceiver and potential intermediaries. However, due to the nature of the protocol, it is stateless and provides no semantic capabilities to interpret the meaning of the messages that are exchanged. W3C published SOAP in version 1.2 as a recommendation in 2007 [26].

A SOAP message consists of an envelope containing a Header and a Body as depicted in Figure 3:

![SOAP Envelope Diagram](image)

**Fig. 3 The SOAP Envelope**

The Header carries the metadata for infrastructure services, such as security, transactions, routing and reliability. As such, it is extensible, but optional. The Body, which is mandatory, contains the payload of a message, i.e. its content, and can be interpreted by the targeted component, i.e. the consumer of a Web service or its provider. Additionally, it can contain optional fault elements which hold errors and status information for a SOAP message.

WSDL and SOAP by themselves are not enough: they just provide mechanisms to realize basic point-to-point communications. Major shortcomings are the inability to capture complex multi-party interactions, and to specify security, reliability, and transactional aspects of a service. To this purpose, a number of WS-* specifications have been defined to extend SOAP messages with various types of non-functional information. Notably, *Web Services Addressing (WS-Addressing)* provides a standard mechanism to specify message routing data within a SOAP Header. Using WS-Addressing, the network-level transport protocol (e.g. HTTP) becomes responsible only of delivering the message to a dispatcher indicated in the destination address, e.g. a Web service run-time server which can interpret the WS-Addressing metadata, and route the message to the right service instance. This information is contained in the Endpoint Reference, which is an XML structure defined in the SOAP Header. It includes the destination URI of the message and any parameter that is required to dispatch the message to the destination. It may also contain optional metadata (such as WSDL) about the service. Moreover, WS-Addressing allows request-response interactions to be decoupled from the lifetime of the underlying HTTP request/response protocol. This is achieved by specifying a special field (ReplyTo) in the SOAP Header which a service provider can use to reply to its requester. Thus, a service does not need to rely on the network-transport level to deliver a response message to the specified recipient. In this way, WS-Addressing enables long-running interactions that can span arbitrary periods of time. The specifi-
cation has been standardized by the W3C Web Services Addressing Working Group in version 1.0 [25].

Another extension to the SOAP Header is Web Services Security (WS-Security). This specification prescribes how to sign and encrypt SOAP messages to ensure integrity and confidentiality of a SOAP message. It also specifies how to attach security tokens such as Security Assertion Markup Language (SAML) [5] and Kerberos [19] to messages to ensure the sender’s identity. The specification has been driven by OASIS and published as a standard in version 1.1 in 2006 [7].

Web Services Reliable Messaging (WS-ReliableMessaging) is also a SOAP extension that focuses on the reliable delivery of messages between distributed applications in the case of failures. The protocol provides different delivery assurances, so that a message is for example delivered at least once, or at most once. The specification has been driven by OASIS and published as a standard in version 1.1 in 2007 [9].

Finally, transactional support for Web services is offered by the Web Service Coordination (WS-Coordination) specification. In this context, a transaction defines two or more service operations that must all be performed with a specific workflow for the transaction to be committed successfully. WS-Coordination, published by OASIS in version 1.2 [12], provides an extensible framework for the support of coordination between distributed applications. Two alternative specifications, both building on top of WS-Coordination, can be used to define the boundaries of a transactional context. These are the Web Service Atomic Transaction (WS-AtomicTransaction) [10], more suitable for short-lived transactions, and the Web Service Business Activity (WS-BusinessActivity) [11], more suitable for long-running transactions, both published by OASIS.

Many other (minor) WS-* specifications exist, however their description falls outside the scope of this introductory chapter to SOA standards. For a consolidated overview of existing Web service standards until 2006, the interested reader is advised to visit http://www.innoq.com/soa/ws-standards.

2.3 Web Application Description Language

An alternative language to WSDL for the description of Web service contracts is the Web Application Description Language (WADL). More generally, WADL aims to provide a machine-readable description of HTTP-based Web applications, which is platform- and language-independent. As such, this specification can be used to describe RESTful Web services. In this regard, WADL focuses on the resources that are needed and provided by a service and their interrelationships, contrarily to WSDL which focuses on the service operations.

A WADL document is described in XML, according to the structure illustrated in Figure 4.

The Grammar element provides a container for the data schemas used to describe the format of data exchanged by the service. These definitions can be included inline
or referenced from an external document. While no specific data format is mandated, the WADL specification describes the use of XML Schema and RelaxNG [3] for this purpose.

The Resources element is where the various resources on offer or needed by a service can be specified. Each Resource is described by a URI where the resource is available, by an optional set of sub-resources, and by specifying the relationships with other (sub-)resources. Moreover, each resource has a resource type, described in the Resource Type element, which lists all the HTTP methods that can be applied to that resource (e.g. a GET or a POST), including the required inputs and outputs for each method. The available methods that can be associated with a resource are defined in the Method element, and pointed to from each resource type definition, while the formats of the input (i.e. HTTP request) and output (i.e. HTTP response) messages are those specified in the Grammar element. Each input message may have a set of HTTP parameters, which are defined in the Param element. Finally, the Representation element describes the representation of a resource state, e.g. an XML document or a simple text document. The elements Resource Type, Method, Representation, and Param are optional as their content can be directly defined within a Resource element. They are used to specify resource behavior that is expected to be supported by multiple resources.

WADL can be regarded to as being the REST counterpart to WSDL 1.1. However, WSDL 2.0 also supports the ability to describe RESTful Web services, thus the two specifications are competing with each other. For such reason, while WADL is currently a W3C member submission [30], the consortium itself has “no plans to take up work based on this submission” [1].

Similar to WSDL, there exist software packages that can generate client side software stubs from a WADL file, such as Glassfish WADL. However, WADL suffers from the same problems that affect WSDL when it comes to describing service properties. It just limits its scope to functional aspects of Web services, and addresses them at a technical level only. As such, WADL is also an IDL that can only be used by a technical audience.
3 Service Discovery

Once services are described, they need to be made discoverable in order for service consumers (i.e. humans or machines) to identify and use them. Moved by this purpose, the Universal Description, Discovery and Integration (UDDI) registry has been developed and standardized in its current version 3.0.2 by OASIS [4]. The registry is based on XML, is platform-independent, and allows service providers to register their services and locate other Web services. In particular, the registry can be queried by SOAP messages to provide access to WSDL documents, which point out how to interact with the services registered in the directory. Therefore, UDDI is tightly-related to WS-* service, described in WSDL.

The register consists of three components: i) White Pages, ii) Yellow Pages and iii) Green Pages. White Pages contain information about the provider of the service, such as their name and address. Thus, the registry can be queried for services for which the service provider is already known. Yellow pages categorize services based on underlying taxonomies, such as the Standard Industrial Classification,⁵ the North American Industry Classification System,⁶ or the United Nations Standard Products and Services Code.⁷ Green Pages are comparatively technical in nature as they describe how to access the Web Services being stored in the repository. In particular, they provide information about the service bindings. This includes the endpoint and parameters of the service as well as the references to specification of relevant Interfaces as described in a WSDL document. As services can have multiple bindings, a service can be related to multiple Green Pages. The OASIS Technical Committee that was responsible for developing UDDI completed their work in 2007. Thus this specification will not be further extended in the future [2].

UDDI is a technical discovery mechanism, which has no explicit links to elements in the business domain. And while it offers multiple extension mechanisms to address these lacunae, it provides limited insights into their concrete utilization. For example, pointers to other documents beyond WSDL that accompany a service can be specified in UDDI, but search capabilities that leverage those documents are limited due to a lack of standardization in regard to the required structure of these documents. In other words, UDDI does not prescribe how the information that is deposited in the White, Yellow, and Green Pages should be described.

An alternative discovery mechanism to UDDI is the Web Service Dynamic Discovery (WS-Discovery), which has been standardized by OASIS in version 1.1 in 2009 [13]. Instead of utilizing a centralized registry, this specification proposes a multicast discovery protocol to locate services in a local network. In this way, services that match the requirements return a response directly to the requester. Unfortunately WS-Discovery remains a technical discovery mechanism which suffers from the same shortcomings of UDDI.

---

⁵ http://www.sec.gov/info/edgar/siccodes.htm
⁶ http://www.census.gov/eos/www/naics
⁷ http://www.unspsc.org
4 Service Composition: Orchestration and Choreography

Often the consumption of a single service is not sufficient to fulfill a certain business goal. In this case, service composition becomes relevant. In service composition, services are no longer executed individually, but coordinated through business processes. A business process describes the logical and temporal order in which a number of activities have to be executed to achieve a given goal. For example, a business process for order fulfillment describes interactions among two parties: a Supplier and a Buyer, and examples of activities of this business process are “Make a request for quote”, “Emit order”, etc.

Business processes are typically described in a diagrammatic fashion by means of process models, where control-flow dependencies among activities can be enriched with information on the resources performing these activities and the data being exchanged by these activities. In service composition, the activities of a process model are performed by Web service operations. This means that the execution of an activity leads to the invocation of a specific Web Service operation.

There are two types of process models, namely choreography and orchestration, which capture two different viewpoints in a service composition. A choreography model describes the global business process of the interactions that occur among all participants. In the order fulfillment example, this means capturing all communication activities (i.e. send and receive) that occur among the Carrier, the Supplier and the Buyer. An orchestration model is the projection of a choreography onto a single participant: it describes the order in which the various service operations need be invoked from the perspective of that participant. For example, if a communication activity in a choreography model is “Make a request for quote”, this activity will correspond to the communication activity “Send request for quote” in the orchestration model of the Buyer, and to the communication activity “Receive request for quote” in the orchestration model of the Supplier. Furthermore, while choreography models focus on communication activities only, an orchestration model needs to also specify the internal activities that are required to create or consume the messages being exchanges by that particular participant. For example, activity “Send request for quote” in the Buyer’s orchestration model needs be preceded by an activity “Prepare request for quote” to compile the ‘request for quote’ document to be sent, while activity “Receive request for quote” in the Supplier’s orchestration model needs be followed by an activity “Emit quote” to create the quote that will be sent back to the Buyer.

The restriction of an orchestration model to communication activities only is called behavioral interface. This artifact, also known as public process or business protocol, describes the dynamic aspects of a Web service, i.e. the order in which its operations have to be provided or consumed, while hiding the internal activities which may contain business-sensitive information. Thus, a behavioral interface complements a WSDL or WADL document (called structural interface), which describes the static aspects of a Web service such as its operations and the content of its messages.
From a service design perspective, a choreography model could be used by business/IT analysts to frame a B2B collaboration among various parties, and then each involved party could create its own orchestration model by projecting the choreography to the activities that are related to that specific party, and then refining this model by adding internal activities. Vice versa, one could expose their own service’s structural and behavioral interfaces, and once a prospective consumer for that service has been identified, a choreography involving both participants could be framed against these interfaces. However, as we will see in this section, languages such as WS-BPEL and BPMN that support orchestration and choreography modeling, do neglect important aspects such as information about delivery channels, payment details, obligations on interactions, service level agreements, and any rights that participants may have in regard to the global business process. As such, these languages alone are inadequate to be used in business contexts.

4.1 Web Services Business Process Execution Language

WS-BPEL (BPEL for short) is an XML-based language used to define behavioral interfaces and orchestration models, which has been standardized by OASIS in version 2.0 [8]. BPEL process models can be deployed to a BPEL engine and be automatically executed.

At its core, BPEL is an imperative programming language, since it allows the creation of process models which are essentially structured in blocks of instructions (barring a few exceptions). It supports the typical constructs of an imperative programming language such as Java. For example, one can create Scopes (like Java routines) and scoped variables (like Java local variables), Assigns (to capture internal activities that manipulate data), as well as to handle exceptions (like the Throw and Catch constructs in Java). Further, there are constructs to establish the order of activities, such as Sequence (to sequentialize activities, like the Java semicolon), Flow (to execute multiple activities at the same time, like Java Threads), and constructs to route control such as While, RepeatUntil and If-Then-Else.

In addition, BPEL has a few more features which are specifically designed to coordinate Web service operations through business process models, thus providing an advantage over traditional object-oriented languages in this respect. First, BPEL is built on top of WSDL 1.1. The language provides three communication activities (Invoke, Receive and Reply) which are directly mapped to Web service operations described in an underlying WSDL document. This is achieved by defining an extension to WSDL called partnerLink type where one can specify whether the BPEL process acts as a provider or consumer for the particular WSDL operation. Second, BPEL supports XML natively. BPEL variables are typed according to an XML Schema type which can be specified in an external XML Schema document or mapped to a WSDL message type (e.g. one can have a variable whose type is that of a ‘request for quote’ message). This eliminates the impedance between XML and object data structures, since there is no need to unmarshal XML
documents into objects (e.g. a Java object) to manipulate them. Third, BPEL offers a set of activities designed to model process-specific aspects, such as asynchronous interactions, multiple sequential or concurrent executions of a scope (ForEach), race conditions between incoming messages and timers (Pick), explicit modeling of parallelism and synchronization (Flow), transactional support (Compensate), and complex multi-party interactions (Correlations). Fourth, since BPEL relies on WSDL, it supports the full stack of WS-* specifications, such as WS-Addressing or WS-Security, which can be seamlessly integrated into BPEL processes. Finally, while the BPEL specification does not come with a visual notation to represent BPEL process models, many tools offer a diagrammatic view of a BPEL model which facilitates their understanding and editing.

Fig. 5 An example of a BPEL executable process from Oracle JDeveloper
BPEL differentiates between two levels of processes, namely abstract process and executable process. The former captures behavioral interfaces and as such it only contains Invoke, Receive and Reply activities, as well as routing constructs, and cannot be executed by a BPEL engine. The latter is used to model fully-fledged orchestrations, where communication and routing activities are interleaved with internal activities like Assigns to model the underlying process logic. As an example, Figure 5 shows an extract of the executable BPEL process for the Supplier participant in our order fulfillment example. For example, activity “prepareQuote” is an Assign (internal activity), while activity “SendQuote” is a Reply (communication activity). The notation used is that of the Oracle JDeveloper 11 BPEL editor.

BPEL lacks a mechanism to model user activities, i.e. those activities that require user input such as the preparation of a ‘request for quote’ document, or the approval of an ‘invoice’ document. During the execution of an instance of a BPEL process, these activities can be exposed to a process user via an electronic form accessible through the user’s worklist application (an interaction paradigm typical of workflow management systems). To obviate this limitation, OASIS is in the process of standardizing WS-HumanTask [14], a specification that defines the scope of user activities, including their properties, behavior and a set of operations to manipulate their data, and a BPEL extension to support WS-HumanTask, namely WS-BPEL Extension for People (BPEL4People) [15].

Another limitation of BPEL is that it only suitable to a technical audience. For example, the lack of a standard notation hampers its use among business analysts, while its strong dependence on XML makes it hard to use for people who are not sufficiently proficient with XML and its related specifications. For example, the only way to manipulate variables’ data in BPEL is via XPath [22] expressions, which are verbose and require a deep understanding of the XML structure of the variables to be addressed.

### 4.2 Business Process Model And Notation

The Business Process Model and Notation (BPMN) provides a language and a graphical notation for the specification of business processes. The latest version (2.0) has been standardized by OMG in January 2011 [18] after a negotiation process lasted over three years.

The primary objective of BPMN 2.0 is to provide a means for all stakeholders to understand and model business processes. Thus, the standard needs to be intuitive enough to be used by business analysts while being expressive enough to represent complex semantics and implementation details for a technical audience. This led to the following main innovations in BPMN over its previous version 1.2 [1]:

- **executable semantics**, similar to BPEL, the semantics of the various modeling constructs has been formally defined, such that BPMN models which incorporate sufficient implementation details can be deployed to a native BPMN engine and be automatically executed;
SOA Efforts 15

- **interchange format**, the specification now provides a standard XML-based serialization format which is machine-readable, thus facilitating tool interchange while preserving semantic integrity, and automatic execution;
- support for both **orchestration and choreography modeling**, with the aim to facilitate proper integration between Business and IT people BPMN 2.0 supports new constructs specifically designed to capture various service composition viewpoints at different levels of abstraction.

Contrarily to BPEL, BPMN comes with its own visual notation, which resembles common flowcharting techniques which most users are already familiar with. However, its expressiveness goes well beyond flowcharts. The language itself is graph-oriented and not block-structured like BPEL, and provides over 100 modeling elements spanning from simple constructs (tasks, gateways, events, subprocesses) to more complex constructs (exception handling, compensation, transactional support, escalation, synchronization signals and more). BPMN also offers constructs to model organizational participants and their subdivisions (e.g. organizations, departments, roles, single persons) as well as business objects (including software systems, information and physical artifacts). Despite the richness of its meta-model, the specification defines four **conformance classes** with the purpose of facilitating the use of BPMN models and their exchange by different stakeholders and tools. These are:

1. **Process Modeling**, suitable at a business level for requirements analysis and communication purposes, this class does not contain implementation details;
2. **Process Execution**, suitable at a technical level for automation in a BPMN engine, this class must specify implementation details such as mapping to Web service operations, format of the messages being exchanged and communication protocols;
3. **BPEL Process Execution**, while the aim of BPMN is to be directly executable in its native XML format, the specification still provides a translation between BPMN modeling constructs and their BPEL counterparts. This class is a restriction of the previous class to remove those BPMN constructs that cannot be directly mapped onto BPEL constructs, due to BPEL's block-structure nature;
4. **Choreography Modeling**, suitable at a business level to frame B2B collaborations, this class contains the constructs for modeling choreographies.

In particular, BPMN offers different types of diagrams to model B2B collaborations at different levels of abstraction. **Conversation** diagrams sit at the top level of abstraction. They provide a simplified view on the order relation of the messages being exchanged among two or more business participants. Figure 6 shows an example of the conversation between the Supplier and the Buyer in our order fulfillment process. As we can observe, at this level of abstraction it is not possible to distinguish mandatory messages from optional ones, nor to group messages in interactions.

A Conversation diagram can be refined into a **Choreography** diagram, which specifies the logical and temporal dependencies among the various messages being exchanged, from the viewpoint of the single interactions. For example, at this level one can introduce data-driven decisions between messages, race conditions between messages and a timer, and parallel messages. Figure 7 shows the refinement of the...
Conversation in Figure 6, where messages have been grouped into interactions (e.g. message ‘Quote’ is the reply to message ‘Request for Quote’), and a race condition has been specified between messages ‘Order confirmation’ and ‘Order cancelation’.

The next level of abstraction is represented by the **Collaboration** diagram, where the choreography of the messages being exchanged is matched by the activities that have to occur within each participant. The projection of a choreography to a specific participant can manifest itself into two types of diagrams within a Collaboration: **Public (or abstract) process** and **Private (or internal) process**. The former corresponds to a behavioral interface since it exposes communication tasks only, while the latter corresponds to an orchestration model where the internal tasks are also modeled. Figure 8 shows the Collaboration diagram. In particular, we can observe that the Buyer is represented as a private process (e.g. activity “Prepare request for
quote” is internal) where the Supplier is represented as a public process, without showing any details of its internal realization.

Fig. 8 An example of collaboration diagram in BPMN 2.0, with both private process (Buyer) and public process (Supplier)

BPMN addresses BPEL’s main limitation of being skewed toward a technical audience only, by providing different types of diagrams to model B2B collaborations at different abstraction levels. However, BPMN does not (yet) provide any concrete mechanism to automatically derive the abstract process model of a specific participant from a choreography model, or to automatically expose a set of choreography interactions from a private or abstract process.

5 Meta and Reference Models for SOA

While by the mid 2000s the notions of service description, discovery and composition had reached a certain level of maturity, as evidenced by the proliferation of standards and research approaches in these fields, there was still little support for understanding and designing service-oriented architectures as a whole. Moved by this purpose, OASIS defined a Reference Model for Service Oriented Architecture (SOA-RM) in 2006. Similarly, The Open Group drafted an alternative reference model in the form of an ontology for Service-Oriented Architectures (SOA Ontology) in 2010. Finally, OMG developed the Service-oriented architecture Modeling Language (SoaML) in 2009 with the aim to facilitate the model-driven design of service-oriented architectures. These three initiatives will be the topic of this section.
A major shortcoming of these specifications, as it will be shown, is their limited support for describing complex (composite) services and processes that involve multiple interacting participants in a business network.

5.1 Reference Model for Service Oriented Architecture

SOA-RM is promoted by OASIS with the purpose to clarify the core notions in the SOA domain. The OASIS-RM Technical Committee was formed in 2003 with the objective to develop a model that could put clarity in the SOA domain, and meantime, foster the creation of specific service-oriented architectures. The outcome, the reference model, was approved for standardization in 2006 [6].

SOA-RM took the form of an abstract framework defining the significant relationships that exist among the entities involved in a service-oriented architecture. It consists of a minimal set of unifying concepts, axioms and relationships and is independent of specific technologies and concrete implementation details. According to this reference model, SOA is “a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains.” Thus, SOA provides a mechanism for matching the needs of a service consumer to the capabilities offered by a service provider.

The central construct of the SOA-RM is the service, which is defined as “a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and it is exercised consistent with constraints and policies as specified by the service description.” Around this notion, three dynamic concepts are of relevance in SOA-RM:

1. Visibility: One has to ensure that the service provider and consumer are able to see each other. This is true for any type of relationship between service provider and consumer regardless of the specific instantiation of these entities (e.g. applications or humans). Visibility is influenced by three factors: awareness, willingness and reachability. The service consumer needs to be aware of the existence of the service provider; both need to be willing to interact with each other and the service provider needs to be reachable by the service consumer.

2. Interaction: The interaction is dependent on the visibility because service consumer and provider cannot interact if they do not see each other. In order to understand what is needed for interacting with a service, the description needs to cover an information model and a behaviour model. The former model details the information that will be exchanged when interacting with the service, whereas the latter model depicts “the actions that may be invoked against the service”.

3. Real World Effect: The result of any interaction is a real world effect, which can instantiate itself as the information that is returned by the service or a change in the state of entities that are involved in the interaction.
The above concepts are considered to be dynamic aspects as they relate to the interactions with services. Besides, SOA-RM defines three additional aspects that relate to services themselves. These are:

1. **Service Description**: A service description contains all the information that is needed for a consumer to decide if the service is of relevance for the specific context. Additionally, the description contains information that is needed to actually use the service. Thus, a service description facilitates dynamic aspects such as visibility and interaction.

2. **Contract and Policy**: A contract is a formal agreement between two or more parties, whereas a policy constrains the use, deployment, or description of a specific entity as defined by its owner.

3. **Execution Context**: The execution context includes all technical and business elements that are somehow of relevance for the interaction between service providers and consumers, such as infrastructure elements and process entities, as well as any policies and contracts that may be in force. Thus, each interaction has a specific execution context.

SOA-RM also provides a notion of Process Model. Accordingly: “the process model characterizes the temporal relationships and temporal properties of actions and events associated with interacting with the service.” However this aspect is intentionally underspecified. For example, while orchestration and choreography may be part of a process model, the orchestration of multiple services is not addressed in the reference model. This is because the focus of this initiative is on modeling what services are and what key relationships are involved in modeling services.

### 5.2 Service Oriented Architecture Ontology

Moved by a similar purpose to SOA-RM, the SOA Ontology has been standardized by The Open Group in 2010 [21]. This standard relies on the following definition of *service*, which is purposefully agnostic to the context in which a service is applied, i.e. a business domain or an IT domain:

*A service is a logical representation of a repeatable activity that has a specified outcome. It is self-contained and is a 'black box' to its customers.*

As such, the SOA Ontology aims to be used by different user categories: i) business people and system/software designers, to clarify on the SOA concepts and how they can be implemented within an organization; ii) solution architects, to provide metadata for architectural artifacts; and iii) architecture methodologists, to provide a component of SOA meta-models.

The ontology itself is specified in the Web Ontology Language (OWL) [29] and consists of a set of classes that capture the core concepts of SOA, their properties and interrelations. There are classes to define the effects of a service interaction; the various elements involved in an interaction (i.e. people, organizations, entire systems
or single tasks), the events they generate, and the policies that may apply. Other classes describe specific aspects of a service, such as a service interface specifying what type of information a service can provide, and a service contract, specifying the effects of an interaction with that service, such that guaranteed service level agreements can be specified. Finally, a set of classes defines how services may be composed (i.e. via an orchestration, a choreography or a more abstract collaboration).

Although this ontology defines the relationships among the different classes, their specific application remains context-generic. Applied to a specific domain or context the ontology needs to be populated by “SOA OWL class instances of things in that domain”. The ontology can be extended by importing other ontologies or classes.

5.3 Service oriented architecture Modeling Language

SoaML is an open-source UML profile and meta-model for designing service-oriented architectures, which is driven by OMG [17]. The objective of SoaML is to offer a comprehensive language to support service design within a model-driven development approach. This initiative was moved by the observation that existing enterprise architecture standards such as TOGAF [20], or general-purpose modeling languages such as UML, were insufficient for capturing all required concepts proper of a SOA in a standard and unambiguous way.

Instead of taking one specific perspective on service design, SoaML accommodates different viewpoints to offer a consistent and cohesive approach to describing service-oriented architectures. The meta-model includes constructs to identify services, their interdependencies and requirements, for specifying service functional capabilities, and the protocols and message exchange patterns, as well as non-functional aspects such as service consumers and providers, consumer expectations and the policies for using and providing services. The standard also provides an extensibility mechanism to integrate SoaML design artifacts with other OMG meta-models, such as BPMN 2.0, and a mechanism to define classification schemes for services.

In line with the model-driven development paradigm proper of UML, one can automatically generate code stubs from the various SoaML artifacts, e.g. deriving the code for a Web service provider or for its client. However, the specification neither limits SOA to be applicable to a purely technical level, nor a service to be purely realized by software components.

6 Summary and Discussion

This chapter provided an overview of the most mature standards in the SOA domain. First, it described two different architectural styles to design services, namely WS-
and REST, and their respective specifications, namely WSDL, SOAP and their extensions, and WADL. Subsequently, the chapter covered two discovery mechanisms for Web services, i.e. UDDI and WS-Discovery, before shifting the focus from single services to service composition, where multiple services can be organized together to provide business value. The chapter introduced different viewpoints in service composition, i.e. behavioral interfaces, orchestration and choreography models, and discussed two different standards that can be used for capturing such views, namely WS-BPEL and BPMN. The chapter completed the discussion on SOA standards by overviewing three specifications that provide guidance for the design of service-oriented architectures, namely SOA-RM, SOA Ontology and SoaML.

This chapter also highlighted the limitations of these standards with regard to their capability of describing service properties, and especially non-functional aspects of services and business processes, which are relevant in a business context. It is worth noting that there has been an effort of recent standardization initiatives such as SoaML and SOA Ontology, to move SOA away from a solely-technical domain, despite their standardization bodies traditionally have a technical focus. SoaML, for example, explicitly includes both technical and business aspects in its specification, and so it does the SOA Ontology, which even uses a car-wash example that is clearly non-technical. However, while providing valuable distinctions among the various concepts of a SOA, these specifications do not provide sufficient depth to allow a universal description of services. They do not consistently describe the functional aspects of a service, in light of the link with the various entities that are involved in the provision and consumption of such services, with the channels through which a service can be provisioned and consumed, and especially with non-functional aspects such as pricing, legal terms and service level agreements. These limitations have triggered the USDL initiative, which aims to provide a universal language for describing both technical and business services.

References