CHAPTER 1

Introduction

With the emergence of service-oriented architecture (SOA), Web services are gaining momentum as key elements in enterprise information systems. Meanwhile, building business and scientific workflows using service composition has become an important method for system integration and process reengineering. Therefore, workflow-driven service composition is now a hot topic in both academia and industry. This book focuses on how to design, analyze, and deploy Web service-based workflows for business, scientific, and medical applications. This chapter discusses the state of the art in both technologies, that is, Web services and workflow management, with a focus on their impact on each other.

1.1 BACKGROUND AND MOTIVATIONS

1.1.1 Web Service and Service-Oriented Architecture

In a 1996 report, Schulte and Natis of Gartner Inc. first used the term service-oriented architecture (SOA) to describe a style of multitier computing that helped organizations share logic and data among multiple applications and usage modes [1].

Thomas Erl, who is recognized as a major contributor in the area of SOA, describes it as an architecture that is *open, agile, extensible, federated*, and *composable*—one that is composed of *autonomous, QoS-capable, vendor diverse, interoperable, discoverable*, and

Business and Scientific Workflows: A Web Service-Oriented Approach, First Edition.
Edited by Wei Tan and MengChu Zhou.

potentially reusable services [2]. He further summarizes the following eight principles of SOA [3]:

- *Standardized Service Contract*. Services should expose their interface and their level of service explicitly, in a standard way. That is the only information needed by a user.
- *Service Loose Coupling*. Services may collaborate with each other but their dependencies should be minimized such that in their communication, they only need to know the contract of others.
- *Service Abstraction*. Services should hide their internal workings from the outside world, except the contract. For example, a service should not expose its technical details, such as the programming language, operating system, and database used, to users.
- *Service Reusability*. Services should be designed in a granularity to promote reuse. If a service is only to be used by a single user, then there is no reason to expose it as a service.
- *Service Autonomy*. A service should have control over its behavior and respond to invocation requests. In a service system, a centralized control is not necessary.
- *Service Statelessness*. Services should minimize the state information exposed to peers and other users, to improve the loose coupling and scalability of the architecture. If a service is stateless, the server side does not need to maintain session or state information; more importantly, a request can be routed easily for the load balance purpose.
- *Service Discoverability*. Services are annotated with metadata such that they can be discovered by users.
- *Service Composability*. Services should be able to be composed to fulfill new and more complex requirements. This principle is closely related to reusability, that is, services are composable such that they can be reused in different applications.

Although Schulte and Natis created the term in 1996, the interest in it only revived in 2000s when Web service technology emerged and matured. In W3C's glossary, a *Web service* is a software system for machine-to-machine interaction over a network [4]. A Web service (or

service for short) should have an *interface description* and be invoked in a way prescribed by this description and using HTTP (Hypertext Transfer Protocol) related protocols. W3C's recommendation is to use WSDL (Web Service Definition Language) to describe its interface, and SOAP (Simple Object Access Protocol) over HTTP to invoke it. All these specifications (WSDL, SOAP, and other Web-related ones) are XML (Extensible Markup Language)-based and can be understood and processed by computer programs. We introduce these related standards in more detail later.

The term *service* in the original definition [1] is abstract and does not tie in with any concrete technology. It is Web service technology that quickly makes SOA tangible and practical to many users. SOA principles provide guidance to Web service implementations and differentiate it with competing technologies such as CORBA, DCOM, and Java RMI [5]. Although SOA does not necessarily build upon Web services and Web service technology by itself is not equivalent to SOA, there is a tight relation between these two concepts. In this book we consider Web services as the best implementation technology for SOA, and SOA principles are applied in Web service practices. Therefore, from now on we do not explicitly distinguish between *Web service* and *service* unless otherwise mentioned.

The triangle in Figure 1.1 lays the foundation of all Web service technologies, by describing the interplay of *service client*, *service registry*, and *service provider*. Consider the following scenario. Multiple service providers want to offer services but are not aware of their clients. Neither do the clients know where and how to access desired services. In SOA, every provider *registers* its service into a registry by providing its function, quality of service, provider information, and so on. The service registry acts like the yellow pages and search engine for services. It categorizes services based on different criteria such as functions, providers, quality of service, and offers a flexible search mechanism to be used by clients. Service clients *look up* the service they want, and obtain the reference to it from the registry. They then *bind* to the service's reference and *invoke* it. Currently, UDDI (Universal Description, Discovery, and Integration) is the standard designed for service registry. Service providers register service descriptions using WSDL. Once clients obtain the WSDL of a service from the registry, they use SOAP to interact with the service. This book introduces these standards in more detail in Section 1.2.

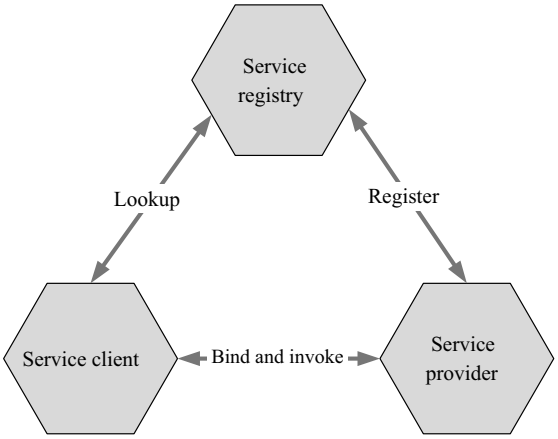

Figure 1.1 The Web service triangle.

1.1.2 Workflow Technology

As defined by Workflow Management Coalition (WfMC), a forerunner in standardizing the workflow technology, workflow is *the computerized facilitation or automation of a business process, in whole or part*. This book focuses on the automation aspect of a business process and thus will not distinguish between a *workflow* and a *business process* unless otherwise mentioned.

The *Workflow Reference Model* [6] published by WfMC in 1995 has laid the foundation in defining a vocabulary and architecture of a workflow system. Despite the drastic technology evolution over the years, this reference model is still applicable to most workflow systems in use today. Here, we use this vocabulary to go over the key concepts in a workflow system and introduce the reference architecture in Section 1.2.2. The reference model defines a common glossary to describe business processes (also known as, workflows) and various artifacts associated with them. It divides the function of a workflow management system into two aspects, that is, *build time* and *run time*. At build time, a *process definition* or *workflow definition* is designed, usually with the help of a modeling tool, as the representation of a business process to be automated. A workflow definition usually contains a set of *activities* and the sequence among them. An activity can be either a *manual activity* that needs human intervention or an *automated*

workflow activity that will be executed by a software application. At run time, a *workflow engine* is the computer software that provides the execution environment for workflows. Given a workflow definition, the engine can start multiple *workflow instances*. For example, an order processing workflow definition can have multiple instances, each of which deals with one particular incoming customer order. There can be many concurrent instances in a workflow engine. In a workflow instance, each activity also has its corresponding *activity instance*.

The workflow technology originated from office automation (OA) area and initially handled the documentation flow among multiple persons or applications. In recent years, workflow has been evolved from a pure IT technology for business process automation, to a much broader concept called business process management (BPM) [7]. BPM is a holistic approach to align many aspects of an enterprise, such as organization, rules, resources, and quality management, in a process centric manner so as to optimize overall operational efficiency and customer satisfaction.

Figure 1.2 illustrates the life cycle of BPM. It starts with the *design* phase. In this phase, first the existing processes (also known as, *as-is* processes) inside an organization, both non-automated and automated ones, are identified. Afterward the *to-be* processes, that is, those to be managed by the BPM system are designed with different criteria. The criteria can be customer satisfaction, response time, or quality of service. A process includes not only a sequenced series of tasks, but also the *organizational unit* responsible for tasks and processes, the

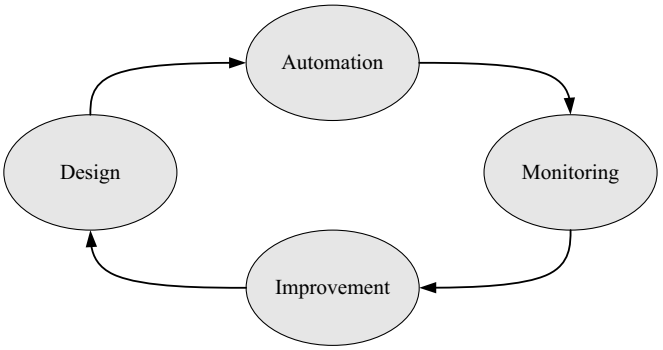

Figure 1.2 The life cycle of BPM.

resources needed, and the *business rules* to control the behavior of tasks or processes.

Second, in the *automation* phase, the business processes designed are translated into a format that can be understood by the underlying process engine (also known as, workflow engine) and become operational in a BPM system (BPMS). Through the BPMS backbone, a process in execution can invoke computer applications, command organizational units, and consume resources to achieve the goal defined in the previous design phase. Nowadays many BPMSs are based on SOA infrastructure. Business processes are translated into BPEL [8] and Web services are adopted as the communication interfaces among processes, applications, and even organizational units.

The designed and later automated processes are monitored in the *monitoring* phase. Usually in a BPMS, the status of running processes and activities are traced, the performance indicators measured, and anomalies reported. Besides the real-time aspect of business process monitoring, the historical data collected in this phase can be used in the subsequent improvement phase.

In the fourth phase, that is, *improvement*, the process model from the design and automation phases, as well as the performance indicator accumulated from the monitoring phase, are put together to provide a retrospect to a process. In 1990s, business process reengineering (BPR) [9,10] emerged. It states that managers need to fundamentally rethink their processes and change them in a dramatic way so as to offer processes with improved customer satisfaction and reduced operational cost. Later when business processes were better managed, a less radical approach compared to BPR, i.e., business process improvement (BPI), became more appealing [11]. The improvement phase in a BPM life cycle addresses the issue of process improvement by examining the process performance indicators, identifying the bottleneck as well as other potentials to improve, and providing guidance to the next iteration of process design.

Business processes are the nexus of many aspects in enterprise management, including quality management, rule management, enterprise resource planning, and business analytics. Here, we briefly discuss these aspects and their relations to BPM. Please note that because concepts such as quality management, enterprise planning, and business analytics are by themselves complex and evolving, we do not mean to make a comprehensive survey.

Six Sigma [12] and total quality management (TQM) [13] are two approaches to identify defects in production processes and improve the quality of process output. Since products are the main output of business processes, processes with associated applications and persons should be responsible for their quality. BPMS can enforce the quality control methodology into the process engine and continuously monitor the process output.

A business rule management system (BRMS) [14] is a software system to define the decision logic of an enterprise, and to provide the result of this decision logic to other enterprise applications at run-time. For example, a business rule may be defined as *if a customer has bought a vitamin product, issue him/her a coupon of the same brand at checkout*. Then at run-time when the checkout system asks the rule engine, it should check the customer's purchase and decide whether or not to issue a coupon. The combination of BPMS and BRMS has two advantages. First of all, business rules can take the responsibility to model the complex decision logic inside a process and drive the decision point at run-time. This separation can make the process model more compact and more adaptive to changes of business requirements. Second, BPMS can record the rules that have been applied and the rule evaluation results. These statistics provides insights into how the rules have governed the operation of a business.

Computer-supported cooperative work (CSCW) addresses how computer systems can support group activities involving multiple people and their coordination [15]. Typical CSCW systems include digital whiteboard, video conferencing, groupware, wiki, and version control software. Since designing a business process is an approach to model how tasks are routed among people, BPMS can be seen as a kind of CSCW systems in terms of its capability to coordinate multiple tasks that involve multiple organizational units. However, typical CSCW systems can better support *ad hoc* coordination or processes compared with BPMS.

Enterprise resource planning (ERP) [16] attempts to integrate all the operations across an enterprise in a single computer system that can serve all departments' essential needs, such as finance, accounting, manufacturing, sales and service, and customer relationship management. Since most enterprise operations are conducted through business processes, BPMS can be cross-department glue to compose multiple individual ERP components into a meaningful business function. Moreover, BPMS can make an ERP system more flexible and

responsive to business needs, by dynamically reengineering business processes and reorganizing the ERP components associated with them.

Business analytics (BA) [17] uses quantitative methods to understand what happened and why (descriptive), and what will happen next (predictive) in business. Business analytics is closely related to multiple BPM phases in Figure 1.2: data needed in analytics are collected in the monitoring phase while the results of such analytics are used as guidance in the improvement phase and eventually reflected in the design phase.

1.2 OVERVIEW OF STANDARDS

1.2.1 Web Service-Related Standards

Figure 1.3 is a layered architecture of Web service-related standards. From bottom to top, it consists of different layers such as message,

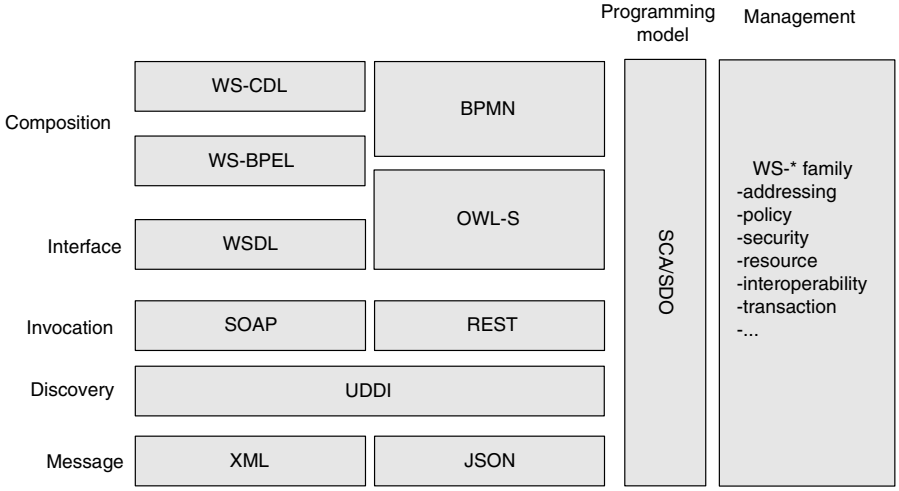

Figure 1.3 Web service-related standards. XML, Extensible Markup Language; JSON, JavaScript Object Notation; UDDI, Universal Description, Discovery, and Integration; SOAP, Simple Object Access Protocol; REST, Representational State Transfer; WSDL, Web Service Definition Language; OWL-S, Web Ontology Language-Service; WS-BPEL, Web Services Business Process Execution Language; BPMN, Business Process Modeling Notation; WS-CDL, Web Services Choreography Description Language; SCA, Service Component Architecture; and SDO, Service Data Objects.

discovery, invocation, interface, and composition. The message layer is about the format of messages exchanged between Web services, and between services and clients. XML [18] and JSON [20] are two major industry standards in this layer. The discovery layer is about how services are advertised and registered such that users can find them. UDDI [21] serves this purpose. The invocation layer defines the protocol specification when services are invoked. Currently, SOAP [22] and RESTful API [23] are the two major protocols where API stands for application programming interface. The interface layer is about the service interface specification that describes how a service can be called, what data structure it expects, and what it returns. WSDL [24] is the standard in this layer and OWL-S [25] offers a semantics-enhanced option. As we mentioned earlier, service composability is an important principle of SOA, and therefore service composition has been a major focus of research by both academia and industry. Among many specifications in this layer, WS-BPEL [8] is the *de facto* industry standard to compose multiple Web services into a business process and to expose this process also as a (composite) service. OWL-S offers semantics-enabled composite service description. WS-CDL [26], from a higher level, defines the peer-to-peer collaborations of services by specifying the ordered message exchanges between them. BPMN [31] is a popular business process modeling specification, and has a tight relation with service composition.

In Figure 1.3, there are also two cross-layer categories, that is, programming model and management. The programming model, which is independent of any specific programming language, defines how to abstract functions as components and use them as building blocks to assemble SOA solutions. Currently, Service Component Architecture (SCA) is a popular SOA programming model; while Service Data Object (SDO) is its data model. Service management involves many cross-cutting and nonfunctional specifications, such as addressing, policy, security, resource, interoperability, and transaction.

Message

XML (Extensible Markup Language) [18] is often used as the message exchange format for interfacing Web services, though this is not required. JSON (JavaScript Object Notation) [20] is also becoming increasingly common. Here, we briefly introduce both languages.

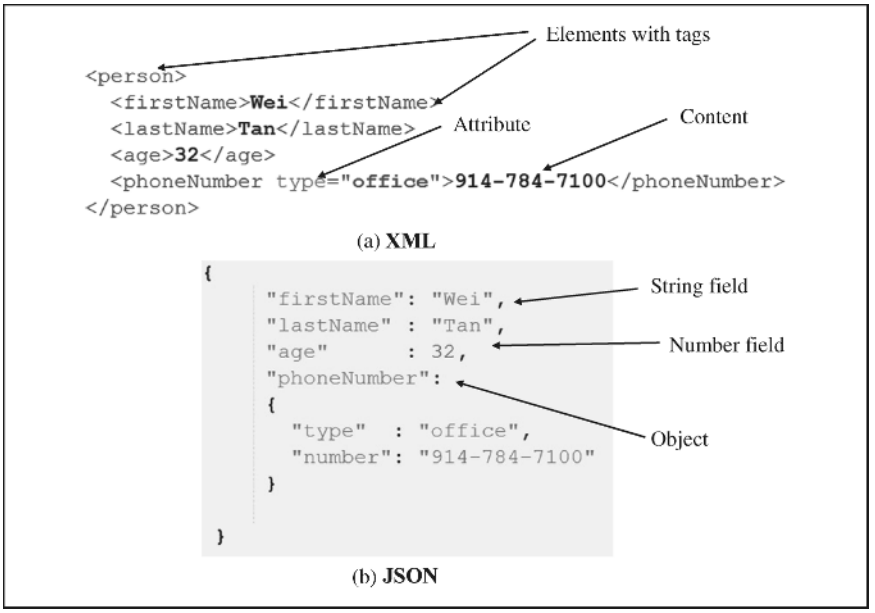

Figure 1.4 (a) XML and (b) JSON presentations of a person.

XML is a text format designed to carry and store data. It uses tags to organize elements, associated contents and attributes. Figure 1.4a shows the XML representation of a piece of information that describes a person, regarding its name, age, and phone number. Element <person> contains nested elements such as <firstName>, <lastName>, <age>, and <phoneNumber>; <firstName> contains content *Wei*; <phoneNumber> also contains attribute "type" that specifies the type of the phone number is *office*.

Compared to XML, JSON, which is based on a subset of the JavaScript programming language, is a lightweight data-interchange format. JSON is built on two structures, that is, a collection of name/value pairs and an ordered list of values. Figure 1.4b shows the JSON representation of the same person we have just illustrated with XML in Figure 1.4a. The object has two string fields for first name and last name, respectively, a number field for age, and a nested object storing the phone number.

Discovery

UDDI (Universal Description, Discovery, and Integration) [27], as a specification language, defines a universal method for enterprises to

dynamically discover and invoke Web services. UDDI was originally designed as a registry and brokerage system that helped users locate services needed in a dynamic way. For example, assume that the hotel industry adopted UDDI standard for hotel room rate checking and reservation. Hotels could then register their services into a central UDDI directory. Individual travelers or travel agencies could then search the UDDI directory to find any hotel's reservation interface. When an interface is found, users would be able to communicate with the service directly.

A UDDI business registry consists of three components:

- *White Pages.* They give information about the business providing the service. Using white pages, users can look for services based on providers.
- *Yellow Pages.* They provide a classification of the service or business, based on standard taxonomies.
- *Green Pages.* They describe how to access a Web service, including the interface, parameters, and address of the service.

WSDL, SOAP, and UDDI were originally designed to be the three pillars of Web services. However, UDDI does not enjoy as much popularity as WSDL and SOAP do.

Interface

WSDL [24] (Web Services Description Language) is an XML-based interface definition specification for describing Web services and how to access them. WSDL describes a Web service, along with the message format and protocol details for it.

Figure 1.5 shows a skeleton of a WSDL document.[1] Such a document describes a Web service by using the following major elements:

- <types>: *Types* section defines the data types used by the service. Types will be referred by the *message* section.

[1] Here we use WSDL 1.1 specification as an example. This is because that, although WSDL 2.0 specification has been released since 2007, we found many public available Web services are written in WSDL 1.1. For a comparison of WSDL 1.1 and 2.0, please see Reference [28] W3C. "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language." Retrieved January 7, 2012, from http://www.w3.org/TR/wsdl20/.

```
<definitions>

<types>
  definition of data types used in message
</types>

<message>
  definition of messages used by operation as input/output
</message>

<portType>
  A portType contains one or more operations,
  each operation has an input and optionally an output message
    <operation>
        <input message="refer-to-message"/>
        <output message="refer-to-message"/>
    </operation>
</portType>

<binding>
  A binding binds a portType to a communication protocol,
  such as SOAP, HTTP Get, and HTTP Post
</binding>

<service>
  A service contains one or more ports, each port assigns a URL to a binding;
  service requests can be sent to this URL w/ protocols defined in binding
    <port name="..." binding="refer-to-binding">
        <address location="URL to send service request"/>
    </port>
</service>

</definitions>
```

Figure 1.5 Skeleton of a WSDL document.

- <message>: *Messages* are to be referred as the input and output of *operations*.
- <portType>: A *portType* contains a collection of *operations*. Each operation has an input message and optionally an output one. Both input and output messages are defined in the *message* section.
- <binding>: A *binding* binds a *portType* to a specific communication protocol, such as SOAP, HTTP Get, and HTTP Post. By this means a client knows which protocol to use when communicating with the service.
- <service>: A *service*, as a concrete entity to be accessed by clients, contains one or more *ports*, and each port assigns a URL to a *binding*. To access a given port, requests should be sent to its URL, by using the protocols (e.g., SOAP, HTTP Get/Post) defined in the corresponding *binding* element.

```
<Envelope>

<Header>
  application-specific information such as authentication, payment,
  intermediary processing about this message
</Header>

<Body>
  contains the actual SOAP message intended for the endpoint of the message
    <Fault>
      contains error messages for this invocation
    </Fault>
</Body>

</Envelope>
```

Figure 1.6 Skeleton of a SOAP message.

Invocation

SOAP [22] (Simple Object Access Protocol) is an XML-based protocol to let applications exchange information over HTTP or other methods such as Simple Mail Transfer Protocol (SMTP). As we have seen in the WSDL specification, SOAP is a binding method to enable the message exchange with a service.

Figure 1.6 shows a skeleton of a SOAP document that contains the following major elements:

- <header>: The *header* section defines the application-specific information such as authentication, payment, and how an intermediary node should process this message.
- <body>: The *body* section contains the actual SOAP message for the endpoint of the message. The format of the actual SOAP message is defined in the WSDL document of the service.
- <fault>: The *fault* section contains error messages for this invocation. The fault can be from the client side, for example, the message is incorrectly formed by the client; or from the server side, for example, the server cannot process the message because it is too busy.

Here we use a real Web service example to illustrate a SOAP request and its response. We use the *NASDAQ Analytics Web Service*[2]

[2] http://www.nasdaqdod.com/NASDAQAnalytics.asmx?v=xOperations

Figure 1.7 The SOAP request (a) and response (b) of the *GetEndOfDayData* operation in the *NASDAQ Analytics Web Service*. The actual SOAP bodies are highlighted in rectangles. Here we ask the end-of-date data for IBM of January 6, 2012.

that offers historical stock quote data. Figure 1.7 illustrates the actual SOAP request and response of the *GetEndOfDayData* operation.[3] In the SOAP body, we ask the end-of-date stock data, by giving a symbol (IBM), and start and end date on January 6, 2012. In the SOAP header we need to give the user name and password[4] for an authentication purpose. The SOAP response includes *Outcome* (Success), authentication *identity* (from Header of the SOAP request), and *open*, *close*, *high*, and *low* price of the IBM stock, as well as the *volume*.

REST (Representational State Transfer) is a lightweight alternative to WSDL/SOAP based Web services. While WSDL/SOAP is still dominant in enterprise applications, REST services are becoming more popular on the Web. For example, companies such as Yahoo, Google, and Facebook have all adopted REST as their service model, while deprecating many of the WSDL/SOAP counterparts. Key architectural features of REST services include the following:

- *Use URI to Organize Web Resources.* In the REST paradigm, every piece of data (also known as, resource) users can access is exposed using a URI (uniform resource identifier). For example,

[3] http://www.nasdaqdod.com/NASDAQAnalytics.asmx?op=GetEndOfDayData

[4] An account with limited free access to this service can be obtained by registration at the Nasdaq data on demand Web site: https://www.nasdaqdod.com/.

http://en.wikipedia.org/wiki/URI is the URI of the article "uniform resource identifier" in Wikipedia.

- *Use HTTP Methods to Access Resources.* Use PUT to create new resources, use GET to obtain resource properties, use POST to update a resource, and DELETE to remove it. For example, you can use the Wikipedia REST API [29] to obtain, create, update, and delete resources, i.e., Wikipedia articles.
- *Make Interactions Stateless.* By "statelessness," we here mean that, a request is self-contained and does not require the server to retrieve or maintain state information for it. Statelessness of a service makes it easy to use and allow the server for easy load-balancing during run time.

The aforementioned *NASDAQ Analytics Web Service* also provides a REST interface, and the end-of-day data for IBM on January 6, 2012 can be obtained by using the HTTP GET method shown below:

```
GET http://ws.nasdaqdod.com/v1/NASDAQAnalytics
.asmx/GetEndOfDayData?
Symbols=IBM&StartDate=1/6/2012&EndDate=1/6/
2012&MarketCenters=Q,B
```
[5]

Service Composition

The following discussions focus on service composition specifications including Web Services Business Process Execution Language (WS-BPEL), Web Services Choreography Description Language (WS-CDL), Ontology Web Language – Service (OWL-S), and Business Process Model and Notation (BPMN).

Web Services Business Process Execution Language (WS-BPEL). Among service composition specifications, WS-BPEL [8] is a dominant one because it is not only approved by OASIS (Organization for the Advancement of Structured Information Standards) as an industry standard, but also execution-oriented and supported by major software vendors as well as the open source community. WS-BPEL stands for Web Services Business Process Execution Language, called BPEL for short.

[5] To use this REST API you need an account with limited free access. It can be obtained by registration at the Nasdaq data on demand Web site: https://www.nasdaqdod.com/.

BPEL defines a meta-model and an XML-based grammar to describe the behavior of a business process that is composed of Web services as well as exposed as a Web service. A BPEL process defines the execution structure on how multiple Web service invocations are coordinated to achieve a business goal. The main constructs and their functions are listed below while a reader may refer to [8] for a comprehensive description.

- <partnerLinks>: It refers to the communication channels between a process and partners. The process can invoke a partner and/or be invoked by a partner.
- <variables>: It defines the variables used by the process; <assign> activity is used to manipulate these variables and, in turn, change the state of the process.
- <receive>: It is used to receive messages from an external partner.
- <reply>: It is used to respond to an external partner with messages. Typically, a <reply> activity is associated with a <receive> one to implement a WSDL request-response operation on a particular partner link. By this means, a BPEL process exposes a WSDL operation to be invoked by others.
- <invoke>: It is used to call a WSDL operation of a Web service.
- Structural constructs such as <sequence>, <if-else>, <while>, <repeatUntil>, <forEach>, and <flow> define the sequential (<sequence>), conditional (<if-else>), repetitive (<repeatUntil> and <forEach>), and parallel (<flow>) execution of the activities contained in them.
- <faultHandlers>: It is used to handle exceptions and faults in a process.

As seen in the description above, another feature that makes BPEL favorable is that it is tightly integrated with many XML specifications such as XML Schema, WSDL, XPath (XML Path language), and XSLT (Extensible Stylesheet Language Transformation). XML Schemas are used as data type definitions; XPath and XSLT provide the support for data assignment; and WSDL is used to model partner services as well as the interface exposed by a BPEL process.

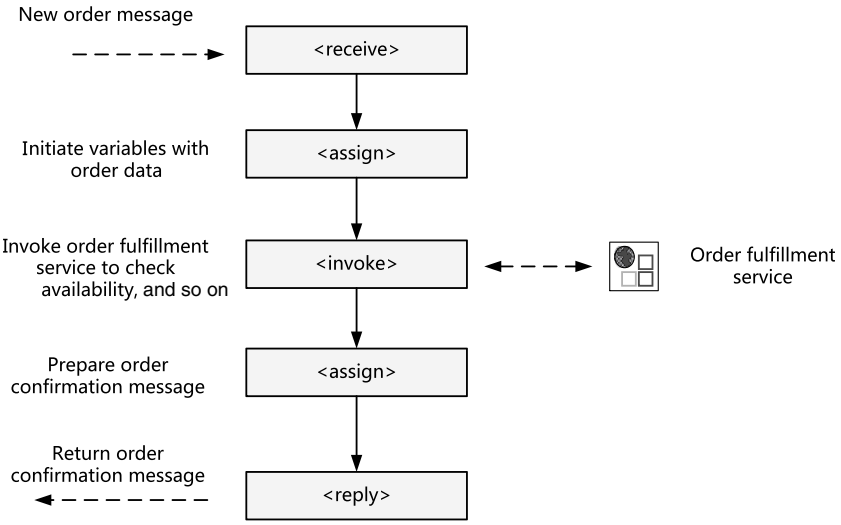

Figure 1.8 An online book selling process in BPEL.

Think of an online book selling application that is implemented in BPEL, as illustrated in Figure 1.8. A customer (through either a Web interface or software application) sends a new order message to the <receive> activity. The process initiates internal variables with the order received in the first <assign> activity, and then invokes the order fulfillment Web service (through the <invoke> activity) to determine the availability of the book, payment status, and so on. When the order fulfillment service returns a result, an order confirmation message is prepared in the second <assign> and the process responds to the customer regarding the status of the order, that is, successful or not.

Web Services Choreography Description Language (WS-CDL). Different from BPEL, WS-CDL [26] is not targeted at describing one executable workflow. From a global viewpoint, it defines the peer-to-peer collaboration of services by specifying the ordered message exchanges between them. That is to say, it abstractly describes the global observable behavior of multiple parties in a collaboration. While BPEL is called a service orchestration language, WS-CDL is usually referred to as a service choreography language.

Ontology Web Language–Service (OWL-S). OWL-S specification [25] is based on OWL (Ontology Web Language) [30] and defines an ontology of a service model. It has three elements, that is, *ServiceProfile*, *ServiceGrounding*, and *ServiceModel*. ServiceProfile describes "what it does." It contains properties such as category, input, output, precondition, and results. ServiceModel describes "how it works," and contains an abstract process model that defines the state transition and how a client can interact with it through message exchange. ServiceGrounding describes "how to access it," that is, the message format, communication protocol, port number, and so on.

Business Process Model and Notation (BPMN). The aim of BPMN [31] is to provide a uniformed, intuitive, and graphical notation for business process models. It can be used as a common language among multiple stakeholders to share the knowledge of business processes: business analysts can use it to express the business goal and function to be achieved; IT experts can use it to implement a workflow that automates the process; managers can use it to monitor the execution and examine the performance of business. Although not originally designed for SOA, it is closely related to the concepts in it. For a few examples, it has a model element called a service task to depict a service. BPMN to BPEL mapping is also a part of its specification. It also contains a collaboration model to describe process choreographies.

Programming Model

Service Component Architecture (SCA) [32] is an industry standard proposed by major software vendors such as IBM, Oracle, and SAP for a generic, language-independent programming model for SOA. Using SCA, users can specify what interfaces a component exposes (i.e., the services to be used by others), what interfaces a component needs to invoke (i.e., the services to be referenced), how a component is implemented (Java, BPEL, etc.), and how many components are wired to compose a larger one. Service Data Object (SDO) defines a data model in SCA and its mapping to Java objects, XML schema, and so on. SCA and SDO are currently supported by software products such as IBM Business Process Manager [33] and Oracle SOA Suite [34].

Management

There are many standards addressing the management and non-functional aspects of Web services. Here, we only list some from W3C's *Web Service Activity Statement* [35]. For a more comprehensive list, please refer to [3,35–37].

- *WS-Policy* describes the policies (security, quality of service, etc.) of entities in a Web services-based system.
- *WS-Addressing* specifies how to describe an endpoint reference of a Web service, and policies to transmit and route messages among endpoints.
- *WS-Transfer* defines a SOAP-based protocol to create, update, and delete Web service-based resources.
- *WS-Enumeration* defines a SOAP-based protocol to enumerate a sequence of XML elements in large data sets, such as logs, event streams, and message queues.

1.2.2 Workflow-Related Standards

In Section 1.2.1, we have discussed Web service-related standards. Some of them, such as BPMN and BPEL, are related to workflow. As the theme of the book is Web service-based workflow, next we introduce workflow-related standards. Among them, the most important one is WfMC's Workflow Reference Model [6], which has laid the foundation in defining the structure of a workflow system. The reference model first defines a general glossary to be used in workflow systems, and we have used this glossary when we introduce workflow technology. The reference model also includes a reference architecture of a workflow system. In this architecture, the core component of a workflow system is one or more workflow engines, and each engine has five interfaces interacting with other major components in the system. Figure 1.9 illustrates the WfMC Workflow Reference Model. Now we introduce what a workflow engine is and how it interacts with other components via five interfaces.

A workflow engine is the software to create, execute, and manage workflow instances, and facilitate their interactions with human and automated applications. A workflow system mainly consists of one or more workflow engines that form *workflow enactment services*.

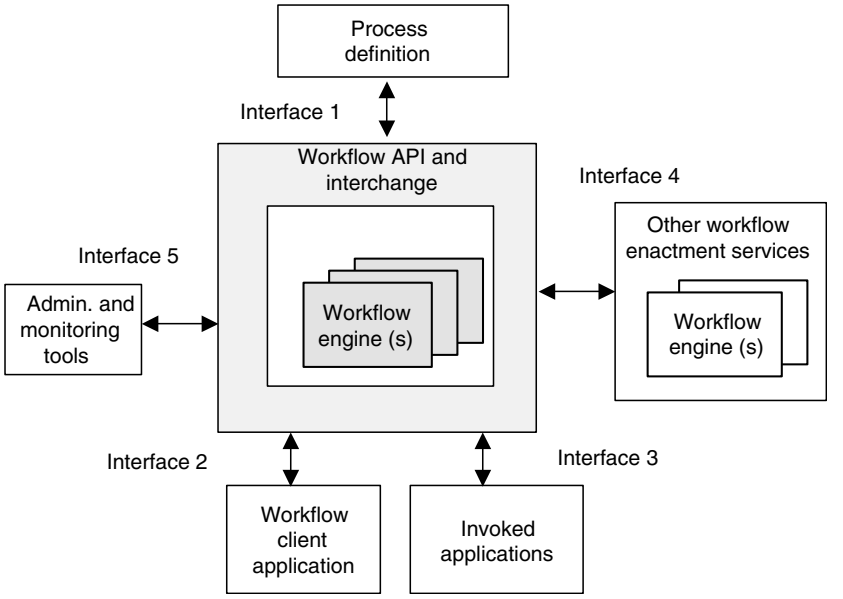

Figure 1.9 The WfMC Workflow Reference Model. *Source*: Reproduced from Reference [6].

A workflow enactment service interacts with other five components via five interfaces as illustrated in Figure 1.9.

Interface 1: Process Definition

Interface 1 defines the interchange format for a process definition, and the API to get it. It is the logic separation between workflow build-time and run-time. Users may use any tool to build a process definition that can be fetched by an enactment service via Interface 1 by means of file transfer or API. To facilitate the exchange of process definitions, in Interface 1, WfMC also defines a meta-model and process definition specification, that is, XML Process Definition Language (XPDL) [38]. XPDL is similar to BPEL and BPMN in various aspects but has its own features. Its meta-model for a process includes the following elements:

- *Activity*, its start and end *events*, and *participants* and *applications* to perform it.
- *Gateways*, such as AND/OR split or join structures.

- *Swimlanes* to lay out a process according to different participants.
- *Workflow relevant data* that are used in each workflow instance at run-time.

Interface 2: Workflow Client Application

Interface 2 is the way to deal with manual activities in a workflow. The client application interface allows participants to retrieve the tasks together with the related data, finish the task, and submit the execution results.

Interface 3: Invoked Applications

Interface 3 is the way to deal with an automated activity in a workflow. This interface allows the engine to invoke an application when a workflow instance reaches an automated activity. There are various protocols to invoke a remote application and a Web service interface is becoming a predominant one among them.

Interface 4: Other Workflow Enactment Services

Workflows in different systems may need to communicate with one another. For example, an order processing workflow of an online merchant like Amazon, may need to talk to a payment workflow of a payment provider like PayPal. Therefore there is a need to define an interface for multiple workflow enactment services to collaborate. Interface 4 defines how two enactment services can set up communication, transfer a process definition, delegate the execution of a subprocess, and transfer data between them.

Interface 5: Administration and Monitoring Tools

Interface 5 defines the administration and monitoring interfaces such as user management, auditing, and resource control.

There exist other standards in workflow and BPM. However, BPEL, BPMN, and XPDL represent the most influential ones. For a more comprehensive discussion about workflow-related standards, please refer to [7,39].

1.3 WORKFLOW DESIGN: STATE OF THE ART

This book focuses on the design (i.e., build-time) aspect of service-based workflow systems. This section surveys the related work and categorizes them into five areas, that is, automatic service composition, mediation-aided service composition, verification of service-based workflows, support for decentralized execution, and scientific workflows. This categorization does not imply that the first four areas are related to business workflows and the last one to scientific ones. Instead, it reflects the fact that the research on business workflows has a longer history and more comprehensive literature coverage. Many of the studies in the first four categories are applicable to both business and scientific workflows, although most of them were originally targeted at business workflows. On the other hand, we have a separate subsection for scientific workflows to discuss their specific topics.

1.3.1 Automatic Service Composition

Composability and reusability are among the eight principles of SOA, as proposed by Erl [3]. Services need to be reusable to make sense for their presence, and need to be composable so as to be reusable. Methods to compose services in a full or semiautomatic way play an important role in SOA, due to the large number of candidate services and the complexity that is required to perform such a composition. Manual composition can be tedious, error-prone, and more importantly, not able to yield satisfactory solutions in a timely manner. Automatic service composition means that given a goal or an abstract process, a desired composite service that consists of the existing services is constructed in an algorithmic way.

Automatic service composition methods can be classified into two categories, that is, planning based [95] and optimization based [43,44]. The former transforms the service composition problem into an AI-planning method, in which the goal state of the composition is given and a chain of service invocations is constructed from the initial state to reach it. They mainly concern the behavior, that is, the sequence of service invocations, of a composite service.

The optimization-based methods mainly concern the nonfunctional aspect, that is, QoS of a composite service [44] after the desired function is fulfilled. QoS constraints include cost, time, availability, and reliability. Optimization methods such as linear programming [43] and

genetic algorithm [44] are used to solve the QoS constraints and yield the optimal solution(s).

Automatic composition attracts the most attention in SOA research because of its importance. However, most of the approaches are theory-oriented and usually require a substantial effort to build a model for each candidate service first. For example, planning-based approaches require an IOPE (Input, Output, Precondition, and Effect) structure for each service operation; optimization-based ones usually require the QoS attributes of each operation as input to the optimization problem. Such a prerequisite affects the applicability of these theoretically sound but heavyweight approaches, and also calls for more lightweight ones.

1.3.2 Mediation-Aided Service Composition

The service orchestration and choreography specifications, and most of the automatic service composition methods, assume the direct composition among services. This is possible under the following assumptions:

1. Services in a composition consent to the same vocabulary in message exchange.
2. The incoming messages of one service are the exact ones provided by its partner(s); the outgoing messages of one service are the exact ones consumed by its partner(s).
3. Services in a composition consent to message exchange sequences.

Services may become partially compatible if any one of the above assumptions no longer holds. Dealing with such cases leads to two major methods, that is, configuration [46] and mediation [47,48], to make partially compatible services work with each other. The former is a heavyweight approach that equips a service with additional variable points such that it can work smoothly with more partners.

Configuration-based approaches require the modification of service implementation. On the other hand, mediation-based approaches are lightweight and less intrusive than the former. The basic idea behind it is that, if two services cannot be directly composed with each other, an adaptor is developed to mediate the mismatch on message formats and/or sequences. Benatallah et al. [49] provide a summary of the mismatch patterns in service composition. Based on the work in

[49], Kongdenfha et al. [50] propose the use of an aspect-oriented programming (AOP) approach to weave adaptor code into services. Brogi and Popescu [51] present a method to automatically generate a mediator between two BPEL services. Nezhad et al. [52,53] propose an automata-based method to synthesize partial compatible services.

1.3.3 Verification of Service-Based Workflows

The issue of verification is not a new topic in workflow research [54]. However, in an SOA paradigm, this problem has unique features to be explored. First, the model elements in specifications such as BPEL are much more complicated than those in traditional workflow specifications such as XPDL. BPEL concepts such as correlation set, dead path elimination, compensation, and fault handling are unique, which brings complexity in verification. Second, because service-based workflows usually interact with each other by message exchange, the correctness of a workflow relies on not only its internal logic, but also how its partners collaborate with it. Even if a workflow is correct from a single-process point of view, its composition with another one may still fail because these two workflows do not agree on their interactions.

Based on the formal methods used, the researches in this area can be classified into several categories, that is, Petri net, automata, and process algebra-based ones.

Petri Net-Based Verification

Ouyang et al. [55] propose a comprehensive Petri net formalism for various BPEL model elements, including basic activities, structured activities, event handler, control link, and fault handling. Martens et al. [56] try to verify the choreography of multiple BPEL processes. Hinz et al. [57] transform BPEL into Petri nets, and then use CTL (Computational Tree Logic) and a model-checking tool to verify various temporal properties.

Automata-Based Verification

Su et al. [58,59] focus on the automaton model for services and apply model checking via LTL (linear temporal logic). A special contribution of their research is a technique called synchronizability analysis to tackle the problem of state space explosion brought about by

asynchronous messaging. Their result shows that, if a composite Web service is synchronizable, its conversation set remains the same when asynchronous communication is replaced with synchronous communication. Thus, a synchronous communication model can be used in LTL model checking. Kazhamiakin et al. [60] develop a set of parametric communication models in service composition. These models range from synchronous communications to asynchronous ones with complex buffer structures. In addition, they develop a technique to associate a service composition with the most adequate communication model that is sufficient to capture all the behaviors of the composition. Using this model, the analysis before the actual verification can speed up the verification.

Process Algebra-Based Verification

Process algebra [61] is an algebraic approach to the modeling and analysis of concurrent processes. Its advantage is that it provides not only temporal logic model checking, but also bisimulation analysis through which whether two processes have equivalent behaviors can be determined. Foster et al. transform BPEL into a kind of process algebra called a finite state process (FSP), and then use a model checking tool to verify properties like whether the implementation satisfies the abstract design specifications [62], whether two services are compatible [63], and whether the composition of BPEL services satisfies the properties defined in WS-CDL [64]. A formal BPEL model based on π-calculus (a kind of process algebra based on Calculus of Communicating Systems) can be found in [65]; a π-calculus-based technique to analyze the behavioral substitution of Web services is proposed in [66].

1.3.4 Decentralized Execution of Workflows

Workflow systems are often built on a client/server architecture in which a single engine takes the responsibility for the operation of a whole process. In many circumstances, this sort of centralized systems may not fully meet the requirements when a workflow is across organizations or security boundaries. Partitioning an integrated workflow into small fragments, each of which is orchestrated by one engine, is a preliminary requirement for its decentralized execution. A team from IBM India Research Lab has conducted a series of studies in the

decentralized execution of composite BPEL services [67–70]. They have investigated how to partition a BPEL program into multiple parts, especially the partitioning of fault-handling code; they model partition policies to improve execution performance; and they consider how to partition the model when dataflow is constrained. Recently, a process-mining-based model fragmentation technique has been proposed for distributed workflow execution [71].

1.3.5 Scientific Workflow Systems

Besides the business community, the scientific community has shown growing interest in workflow technology and has exploited its power in Grid computing, scientific investigation, and job flow management [72,73]. Essentially, a *scientific workflow* is a specialized workflow orchestrating computation and data manipulation tasks into a process of scientific value. A scientific workflow system becomes prominent with the arising interest in *data-intensive science*, or *e-Science* [74]. In e-Science, scientists are facing an enormous increase in raw data from various resources, such as telescopes, instruments, sensor networks, accelerators, and supercomputers. For example, in high-energy physics, the main detectors at the Large Hadron Collider (LHC) produced 13 petabytes of data [75] in 2010. In bioinformatics, 1330 molecular biology databases [76] were reported in 2011. Among them, *GenBank*, the US NIH DNA sequence database, contains more than 286 billion entries for more than 380,000 organisms [77]. To conduct any nontrivial analysis using large amounts of data, scientists need the help of a workflow system.

There are many scientific workflow systems available and the edited book [72] provides a good summary of them. Each of them provides a graph-based interface for service composition, with an underlying workflow metamodel. The workflow metamodels used by these service-based systems are either adopted from industry standard or homegrown.

GPEL and OMII-UK have adopted BPEL. Adopting BPEL can bring advantages such as rigorously defined model syntax and semantics, readily available software tools, and portability of workflow specifications. However, scientific workflows have a particular focus on data flow (versus control flow in business workflows) and parallelism (versus the complex logic in business workflows), and are tightly integrated with the underlying computation infrastructure. To deal

with these unique features of scientific workflows, many systems have their own workflow metamodels.

1.4 CONTRIBUTIONS

Besides this chapter, Chapter 2, which introduces the Petri net formalism, and Chapter 9, which provides a summary, the remaining six chapters can be categorized into methodologies and applications (Table 1.1).

Chapters 3–5 cover methodologies. Many service composition approaches are heavyweight, that is, they require much input such as semantic notation. This book presents a lightweight approach, by making the best use of the data structure and relations embedded in service descriptions, and designing a data-driven composition method. It is also observed that in real-life scenarios, services do not exactly match one another. Hence, this book presents a method to analyze this

Table 1.1 Contributions of This Book

Chapter	Problem	Contribution	Category
3	A lightweight service composition approach	Data-driven service composition	Methodology
4	Compose partial compatible services	An mediation-aided approach to analyze and compose services	Methodology
5	Web service selection under QoS constraint	QoS aware Web service functional configuration	Methodology
6	Application in healthcare	Web service composition techniques in a healthcare service platform	Application
7	Application in e-Science	Web services and scientific workflows in e-Science	Application
8	Social network meets services computing	Network analysis and reuse of scientific workflows	Application

phenomenon and to add a mediator to glue partially compatible services. Finally, there may be many configurations providing identical functionality with different QoS; this book gives a Web service functional configuration model using Petri nets and uses a novel linear programming formulation to find the configuration with the best QoS.

Chapters 6–8 cover applications. There is a strong demand to compose various health services for the creation of personalized healthcare service systems. This book presents our experience in building a public healthcare information service platform and in applying service composition in such a platform. This book also discusses the wide application of Web services and workflows in the e-Science domain. Specifically, it introduces the design and implementation of caGrid Workflow Toolkit that supports service discovery, composition, and orchestration in the cancer Biomedical Informatics Grid (caBIG). Finally, the proliferation of social network services has started to impact services computing, especially in the e-Science domain. This book presents a network analysis on myExperiment, an online biological workflow repository, and reveals the usage pattern of services in scientific workflows. Based on this network model, we develop a GPS-like system that provides guidance and recommendations to domain scientists when they perform service composition to fulfill their research needs.