The VLDB Journal (2008) 17:537–572
DOI 10.1007/s00778-006-0020-3

REGULAR PAPER

Deploying and managing Web services: issues, solutions, and directions

Qi Yu · Xumin Liu · Athman Bouguettaya · Brahim Medjahed

Received: 12 August 2005 / Accepted: 19 April 2006 / Published online: 4 November 2006

Abstract Web services are expected to be the key technology in enabling the next installment of the Web in the form of the *Service Web*. In this paradigm shift, Web services would be treated as *first-class* objects that can be manipulated much like data is now manipulated using a database management system. Hitherto, Web services have largely been driven by standards. However, there is a strong impetus for defining a solid and integrated foundation that would facilitate the kind of innovations witnessed in other fields, such as databases. This survey focuses on investigating the different research problems, solutions, and directions to deploying Web services that are managed by an integrated *Web Service Management System* (WSMS). The survey identifies the key features of a WSMS and conducts a comparative study on how current research approaches and projects fit in.

Keywords Service-oriented computing · Interoperability · Web service management system

This research is supported by the National Institutes of Health's NLM grant 1-R03-LM008140-01.

Q. Yu (✉) · X. Liu · A. Bouguettaya
Computer Science Department, Virginia Tech,
Blacksburg, VA 24060, USA
e-mail: qyu@vt.edu

X. Liu
e-mail: xuminl@vt.edu

A. Bouguettaya
e-mail: athman@vt.edu

B. Medjahed
Department of Computer and Information Science,
University of Michigan, Dearborn, MI 48128, USA
e-mail: brahim@umich.edu

1 Introduction

The Web is a distributed, dynamic, and large information repository. It has now evolved to encompass various information resources accessible worldwide. Organizations across all spectra have already moved their main operations to the Web, which has brought about a fast growth of various Web applications. This has dramatically increased the need to build a fundamental infrastructure for efficient deployment and access of the exponentially growing plethora of Web applications. The development of enabling technologies for such an infrastructure is expected to change the business paradigm on the Web. *Web services* have *de facto* become the most significant technological by-product. Simply put, a Web service is a piece of software application whose interface and binding can be defined, described, and discovered as XML artifacts [9]. It supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols. Examples of Web services include online reservation, ticket purchase, stock trading, and auction. Standards are key enablers of Web services [102]. Major industry players took a lead to set up crucial standards. This has greatly facilitated the adoption and deployment of Web services [56]. Three key XML-based standards have been defined to support Web service deployment: SOAP [109], WSDL [113], and UDDI [110]. SOAP defines a communication protocol for Web services. WSDL enables service providers to describe their applications. UDDI offers a registry service that allows advertisement and discovery of Web services.

The fast increasing number of Web services is transforming the Web from a *data-oriented* repository to a *service-oriented* repository [85,93]. In this anticipated framework, existing business logic will be wrapped as Web services that would be accessible on the Web via a Web service middleware [104]. Web services will work as self-contained entities to fulfill users' requests. Cooperation among multiple Web services will additionally improve the *quality* of answers by providing *value-added* services [67]. Web services are anticipated to form the underlying technology that will realize the envisioned "sea of services" [85].

One key challenge for Web services is interoperability. Seamless interoperation is the ultimate goal that Web services aim to achieve. Interoperation moves Web services beyond the elementary framework built upon the three key standards (i.e., SOAP/WSDL/UDDI). It also helps achieve robust service *compositions*. The Web service *composition* refers to combining outsourced Web services to offer *value-added* services [92]. Process modeling languages are a major tool to compose Web services [51]. Typical examples include Business Process Execution Language for Web Services (BPEL4WS, the convergence of XLANG [70] and WSFL [53]) [83], Web Services Choreography Interface (WSCI) [111], and Business Process Management Language (BPML) [16]. A common characteristic of all these modeling languages is that XML is used to depict Web services. The lack of well-defined semantics affects their ability to achieve seamless interoperability in the true sense. The concept of *ontology* is the key to empowering Web services with semantics [38].

Ontologies enrich Web services with expressive and computer interpretable languages. They help capture the semantics of Web services. *Ontologies* were first proposed in the artificial intelligence community to facilitate knowledge sharing and reuse. They aim to construct a shared and common understanding of relevant domains across people, organizations, and application systems [40]. Ontologies function like metadata schema with rich descriptive capacity [31,50]. They facilitate the information exchange of people and computers in terms of both syntax and semantics [58]. A typical example of ontologies is OWL-S, which is a OWL-based Web services ontology [63]. It aims at establishing a framework to describe *semantic Web services*. OWL-S would be a semantic-based substitution of those syntactic-based Web services languages for service description, service registration, and service composition. Thus, integrating ontologies into Web services could not only enhance the quality and robustness of service discovery and invocation, but also pave the way for automated composition and seamless interoperation.

Interoperability has been a central concern for enabling the service-oriented Web [65]. However, the full deployment of Web services requires dealing with additional issues including Quality of Web Services (QoWS), Web service management, and security/privacy. As multiple Web services are expected to deliver similar fuctionalities, QoWS is considered as a key concept in distinguishing between competing Web services [25]. The international quality standard ISO 8402 describes quality as "the totality of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs" [88]. The quality and usage of Web services is controlled and monitored via a set of management mechanisms [107]. We identify two types of management techniques. *Control management* aims to improve the service quality through a set of control mechanisms (e.g., transaction, change management, and optimization). *Monitoring management* rates the behavior of Web services in delivering their functionalities in terms of the QoWS parameters. Security and privacy mechanisms need to be developed to fulfill the requirements of the Web service environment [39, 83]. Web service requests and replies are sent over the Internet, and hence are subject to various security threats. Additionally, users' sensitive information may be divulged to unauthorized third parties during the Web service interactions.

The objective of this paper is to survey the fundamental issues and proposed solutions for successfully deploying and managing Web services. Standards have so far been the key enablers of Web services. The intensive standardization support reflects the strong industry backing of Web services as a key mechanism for deploying programmable functionalities on the Web. This is historically similar to the progress of another key enabling technology, the database management system (DBMS) technology. The DBMS technology has had tremendous progress with the advent of the relational model which was instrumental in giving databases a solid theoretical foundation. Web services still have to have such a strong foundational underpinning. The foundational work is still in its infancy.

Current surveys focus mostly on the technological aspects in the development and deployment of Web services, with little regard to their foundational underpinning [9,82]. These surveys mainly fall into two categories: descriptive or application-oriented. For example, Alonso et al. [9] give a comprehensive description of Web services. However, the focus is mostly on technology-related definitions of the different standards and products. Our survey takes a different approach in two major ways. First, we investigate the different technologies from a foundational point of view, relating each standard/technology to a set of parameters. Second, we investigate the related standards/technologies from a research point of view, focusing on the functionalities that are provided and the fundamental issues that are being addressed. Tsalgatidou et al. [100] present the basic Web service model and sketch the Web service lifecycle in their work. They present an overview of the key Web service standards, including SOAP, WSDL, UDDI, and ebXML. Medjahed et al. [65] reviewed Web service technologies from a business-to-business (B2B) application perspective. It outlines the major features of Web services and examines how they could fit into the B2B interaction environment. Service composition is presented as a powerful tool to combine inter-enterprise applications and enhance the B2B interactions. Papazoglou et al. [82] provide a review of the Web service technologies as an application of service-oriented computing. They examine several major features of Web services. They also discuss how the Web service features benefit the service-oriented architecture. In this paper, we conduct a state-of-the-art analysis of the Web service technologies with a focus on the research issues. Our aim is to lay a foundational framework for Web services, called Web Service Management System (WSMS). The WSMS is a comprehensive framework for the Web service life cycle, including *developing*, *deploying*, *publishing*, *discovering*, *composing*, *monitoring*, and *optimizing* access to Web services. In such a framework, Web services are treated as *first-class* objects that are manipulated as if they were pieces of data. A set of parameters is proposed to analyze and assess Web service technologies within the WSMS.

The remainder of this paper is organized as follows. In Sect. 2, we provide a historical perspective on the analogy between Web services management and database management. In Sect. 3, we give an overview of the proposed WSMS. The WSMS is further presented in a step by step fashion. We first present the standard Web service reference model. We then describe the Web service technologies using a layered stack approach. A set of dimensions of Web services is defined. This set corresponds to the key issues for deploying and managing Web services. The WSMS architecture is then proposed to address these issues. From Sect. 4 to 7, we detail the WSMS architecture by providing a detailed description of each of its components. In Sect. 8, we review the current Web service deployment systems. In Sect. 9, we evaluate these systems by mapping them to the WSMS architecture. We then outline some future research trends.

2 Towards a Web Service Management System (WSMS): historical perspective

Providing a solid framework for the deployment of Web services aims at providing a powerful foundation much like the relational paradigm provided for the database field. In the context of Web services, scientific communities are expected to benefit from the ability to share resources on a large scale that will lead to further innovation, collaboration, and discovery. Governments would be able to better serve citizens and other constituencies by streamlining and combining their Web accessible resources. Businesses would be able to dynamically outsource their core functionalities and provide economies of scale. This would translate into better products at cheaper prices.

Fully delivering on the potential of the next-generation Web services requires building a foundation that would provide a sound design framework for efficiently developing, deploying, publishing, discovering, composing, monitoring, and optimizing access to Web services. The proposed Web service foundation will enable the deployment of *Web Service Management Systems* (WSMSs) that would be to Web services what DBMSs have been to data. We largely draw on the experience and lessons learned from designing the database foundation. The transition from the early file systems to databases is of particular interest. If one looks carefully at the history of relational databases, one can clearly observe a striking parallel with the current situation of Web services. Therefore, understanding the related transition processes and resulting foundational models will provide us with valuable insight and help in designing a sound foundational framework for Web services.

For the purpose of comparison, it is noteworthy to mention that the early file systems were developed for single users who had their own exclusive data space. In addition, because of the state of data storage technology back then, data sizes were very small. Computer systems were more computation bound (CPU bound) as opposed to being data bound (I/O bound). More importantly, the growing deployment of computer systems was coupled with a tremendous growth in data volume and number of users sharing files. The new computing environment posed fundamental challenges in providing uniform data representation, efficient *concurrent* access to and recoverability of data, and ensuring *correctness* and *consistency*. As a result, the nascent database research focused on laying the foundation for the next-generation data management system to address these issues. Early work on the network and hierarchical models set the foundational work in motion with the early standardization-based models (e.g., DBTG model [2,1]). The field of databases enjoyed widespread acceptance only after the relational model was proposed by Codd [25]. What made the relational model a success story is the sound mathematical foundation upon which it is built. The relational model is based on set theory and relational calculus (declarative). This simple, yet powerful paradigm was met with instant success in industry [10] and academia [97]. New concurrency control models were proposed. Optimization techniques based on algebraic principles were also proposed. This activity spurred and sustained the deployment of databases as ubiquitous tools for efficiently managing large quantities of data.

We observe that this historical perspective on managing data is quite similar with the evolution of Web services. Web services deliver complex functionalities over the Web. The early function libraries (e.g., DLL on Windows) were also designed to wrap certain functionalities and make them reusable. The libraries provide a set of APIs, upon which users can manually incorporate the functionalities into their programs. In addition, the function libraries are only locally accessible. The emergence of computer networks present new requirements for sharing and reusing of functionalities. There is a need to integrate applications within or across organizations. Middleware technologies (e.g., RPC, COM, and CORBA) took a first step to support intra-organization interoperability. Web services came as a result to address the inter-organization interoperability. They aim to provide users an integrated access to the functionalities available on the Web. The development of Web services has so far mostly been the result of standardization bodies usually operating on a consensus basis and driven by market considerations. In this context, innovation and long-term market effects are not usually primary concerns. Because of the global nature of the Web, the standardization process has so far been very fragmented, leading to competing and potentially incompatible Web service infrastructures. Many companies have invested very heavily in Web services technologies (Microsoft's .NET, IBM's Websphere, SUN's J2EE, to name a few). These efforts have resulted in a fast-growing number of Web services being made available. The envisioned business model is expected to include a whole community of Web service providers that will compete to provide Web services. It is important that this investment produce the expected results. To maximize the benefits of this new technology, there is a need to provide a sound and clean methodology for specifying, selecting, optimizing, and composing Web services. This needs to take place within a secure environment. The underlying foundation will enable designers and developers to reason about Web services to produce efficient Web Service Management Systems.

2.1 Web services versus data

Despite similarities in nature and history, managing Web services is different from managing data in many significant ways. First, data in traditional DBMSs are passive objects with a set of known properties, e.g., structure, value, functional dependencies, integrity constraints. On the other hand, Web services are active and autonomous entities that have a set of *functions* rather than *values*. Moreover, unlike the data in DBMSs, the run-time *behavior* of Web services when they are invoked may not be precisely known. Second, accessing a service on the Web is similar to accessing data from a distributed DBMS. For example, to access a Web service, the service requester must search in one or more service registries. However, Web services carry more complex information, which makes accessing services a more complicated process. This involves understanding the different services' syntactic and semantic descriptions, selecting the services providing the requested functionality, understanding their communication protocols, and finally engaging in a sequence of message exchange with the selected services. Of significant importance are more complex scenarios where requests for services may require the composition of several Web services; various issues pertaining to individual Web services need to be reconciled before they can be combined. Examples include security protocols and policies, and Quality of Web service for optimization purposes.

3 Overview of the WSMS framework

A variety of definitions about Web services are given by different industry leaders, research groups, and Web service consortia. For example, the W3C consortium defines a Web service as "*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*" [112]. IBM defines Web services as "*self-describing, self-contained, modular applications that can be mixed and matched with other Web services to create innovative products, processes, and value chains. Web services are Internet applications that fulfill a specific task or a set of tasks that work with many other web services in a manner to carry out their part of a complex workflow or a business transaction*". According to Microsoft, "*A Web Service is a unit of application logic providing data and services to other applications. Applications access Web Services via ubiquitous Web protocols and data formats, such as HTTP, XML, and SOAP, with no need to worry about how each Web Service is implemented*". HP defines Web services as "*modular and reusable software components that are created by wrapping a business application inside a Web service interface. Web services communicate directly with other web services via standards-based technologies*". SUN perceives a Web service as an "*application functionality made available on the World Wide Web. A Web service consists of a network-accessible service, plus a formal description of how to connect to and use the service*".

The aforementioned definitions give a high-level description of the major objective and supporting technologies of Web services. Interoperation among machines is the major design goal of Web services. As the supporting standards, WSDL enables XML service description of Web services and SOAP defines a communication protocol for Web services. These definitions give an *outside* view of Web services. In this section, we go a step further by setting up a comprehensive WSMS framework to support the entire Web service life cycle, including *developing*, *deploying*, *publishing*, *discovering*, *composing*, *monitoring*, and *optimizing* access to Web services. We first present the Web service reference model. We elaborate on the three key players in this model. We also identify different layers that enable the interaction in the Web service model. We define a collection of dimensions across these layers. The major components of the proposed WSMS are devised based on these dimensions. Each component provides functionalities that address the issues specified by the corresponding dimension.

3.1 Scenario

We consider a travel agency, named `TravelAgency`, providing the travel arrangement (e.g., transportation, itinerary, and accommodations) for its clients (Fig. 1). Assume a university professor, Joan, wants to attend an international conference in Sydney, Australia. Typical services needed by Joan might include airlines, ground-transportation (e.g., taxi and car rental), accommodation (e.g., hotels and inns), and other entertainment services (e.g., restaurant and opera house). Joan needs to first search for the services that provide travel packages. The search can be conducted in some well-known service registry. We assume that the `TravelAgency` is located by Joan. Joan would then send her request to it. To offer a complete tour package, the `TravelAgency` needs support from its business partners (e.g., `AirCompany`, `Hotel`, `Restaurant`, `CarRental`, and `OperaHouse`) to arrange flights, hotels, cars, and other entertainment facilities. These companies all define the service description for their Web services and publish them on a well-known service registry, whereby the `TravelAgency` can search and locate them. The `TravelAgency` needs to outsource services from these business partners to provide the entire travel package. Since Web services with similar functionalities might be provided by competing business partners, there is a need to optimize access to those Web services or composition thereof with respect to the expected quality. In addition, the Web services need to be accessed in a reliable and secure manner. This example articulates a typical Web service usage scenario. It will serve as a running example to illustrate the various Web service concepts.

3.2 Web service reference model

Three types of participants cooperate to set up a Web service model (see Fig. 2), including: *service provider, service client*, and *service registry*. Web services interact in three primary modes: *service publishing, finding*, and *binding*. Interactions depend on the Web service *artifacts*, which include the *service implementation* and the *service description*.

- *Participants* in a Web services model are categorized into three types:
  - *Service provider* is the owner of the Web services. It holds the implementation of the service application and makes it accessible via the Web.
  - *Service client* represents a human or a software agent that intends to make use of some services to achieve a certain goal.
  - *Service registry* is a searchable registry providing service descriptions. It implements a set of mechanisms to facilitate service providers to publish their service descriptions. Meanwhile, it also enables service clients to locate services and get the binding information.
- *Interactions* with a Web service take place in three modes:
  - *Service publication* is to make the service description available in the registry so that the service client can find it.
  - *Service lookup* is to query the registry for a certain type of service and then retrieve the service description.

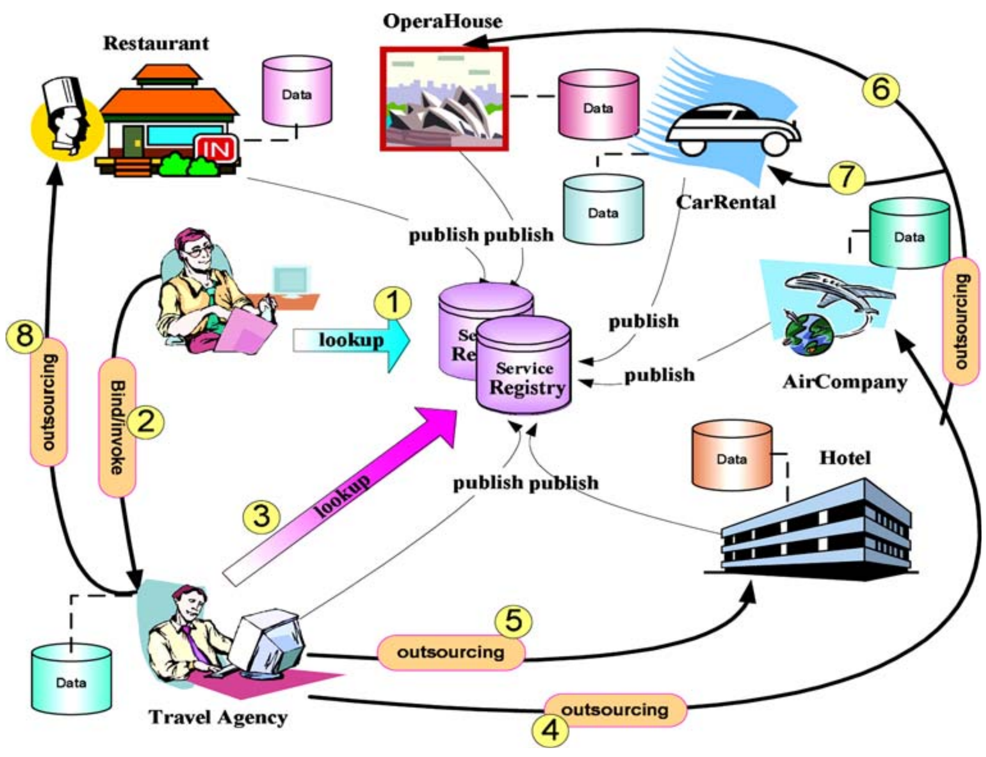

Fig. 1 A travel arrangement scenario

  - *Service binding* is to locate, contact, and invoke the service based on the binding information in the service description.
- *Artifacts* encompass the service implementation and description:
  - *Service implementation* is a network accessible software module realized by the service provider. It could be invoked by a service client or act as a service client to interact with another service provider.
  - *Service description* could contain the syntactic and semantic information of the Web services. The syntactic information describes the input/output of the operations, the binding information, the data types, and so on. The semantic information encompasses the domain of interests, business functionalities, QoWS issues, and so on.

3.3 Web service stack

The Web service stack contains five key layers: communications, messaging, descriptions, discovery, and processes, which are shown along the vertical direction in Fig. 3. It is an extension of the W3C service stack [112]. Similar to the W3C service stack, each stack layer provides certain functionality to support interoperation between Web services and service clients or among Web services. However, we categorize interoperability into two dimensions: syntactic and semantic (see Sect. 3.4 for details). Therefore, our service stack is distinguished from the W3C stack by further identifying the syntactic and semantic interoperability offered by all layers above the messaging layer.

- *Communications*: The underpinning of the Web services stack is the network, where the underlying communications take place. A set of network protocols helps realize the network accessibility of Web services. The wide adoption of HTTP makes it the first choice of standard network protocol for Internet available Web services. Other network protocols could also be supported, such as SMTP.
- *Messaging*: The messaging layer provides a document-based messaging model for interaction with Web services. The messaging model works with a wide variety of network protocols. For example, the messaging model can be combined with HTTP to traverse firewalls. In another case, combination with SMTP enables the interaction with Web services that support asynchronous message exchanges.
- *Description*: The description (or representation) layer is for describing Web services. It wraps Web services and specifies their functionalities, operations, data types, and binding information using a service interface. The WS discovery will rely

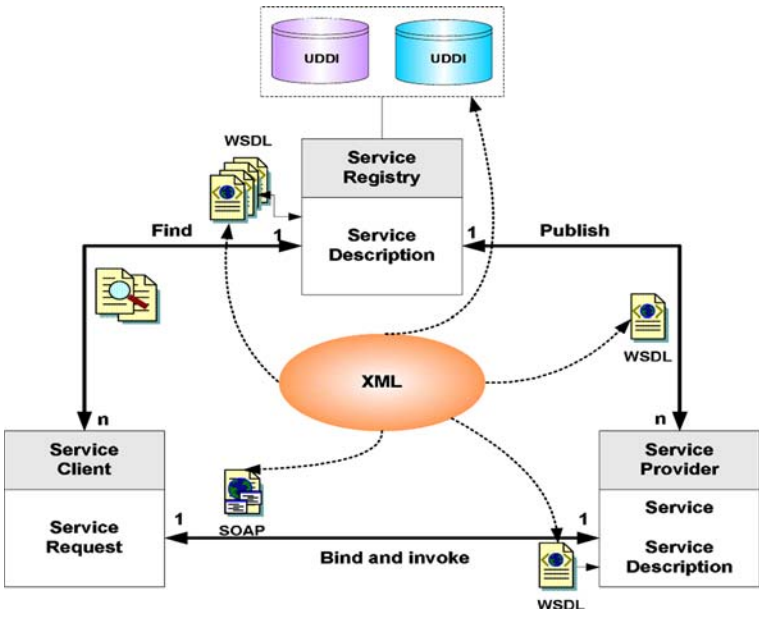

Fig. 2 W3C Web services reference model

on the WS representation to locate appropriate Web services.

– *Discovery*: The discovery layer is for locating and publishing Web services. It enables the usage of Web services in a much wider scale. The service providers can store the service descriptions in a service registry via the publication functionalities provided by the WS discovery. Meanwhile, service requestors can query the service registry and look for interested services based on the stored service descriptions.
– *Processes:* The processes layer supports more complex interactions between Web services, which enables Web service interoperation. It relies on the basic interaction functionalities provided by the technologies at lower layers in the Web service stack. For example, it needs Web service discovery and representation for querying and locating Web services based on their descriptions. The selected Web services are used to construct the process, which consists of a sequence of coordinated Web services.

3.4 Key dimensions for building a WSMS

The variety of the Web service technologies constitutes a rich solution space of Web services. Each technology has specific design requirements depending on the usage scenarios. Therefore, it is important to determine the relevant requirements for deploying and managing Web services. In this section, we identify a set of dimensions to evaluate the Web service technologies. These dimensions are in line with the key requirements for deploying and managing Web services. We take the Web service stack as a starting point and extend it to address the developing trends of Web service technologies. The dimensions are defined according to the vertical layers of the Web service stack (see Fig. 3). They include *interoperability*, *security & privacy*, *Quality of Web Services (QoWS)*, and *management*.

Interoperability This dimension refers to the extent to which participant Web services would cooperate to accomplish a common objective. For example, in the aforementioned scenario, the `TravelAgency` needs to interoperate with the `AirCompany` and `Hotel` to serve its clients. The common objective of the three parties is to provide satisfactory services for travelers. Good interoperability is a must for them to achieve this goal. Web services are designed to bring together applications from geographically distributed and heterogeneous environments and provide interoperability among them [94]. Interoperability could be achieved via three main approaches: *standards*, *ontology*, and *mediation*. A *standard* is a specification or format that has been approved by a recognized standardization organization or is accepted as a *de facto* standard by the industry. Several standardization efforts in Web services have been initiated by a focused group of companies, and have then been adopted by different organizations such as OASIS (Organization for the Advancement of Structured Standards) and W3C. These consortia aim at standardizing the different aspects of Web service interactions (e.g., message format, interaction protocols) [9]. An *ontology* is a *formal* and *explicit* specification of a *shared conceptualization* [28]. "Conceptualization" refers to an abstraction of a domain that identifies the relevant concepts in that domain. "Shared" means that an ontology captures *consensual* knowledge. The development of ontologies is often a cooperative process involving different entities possibly at different locations (e.g., businesses,

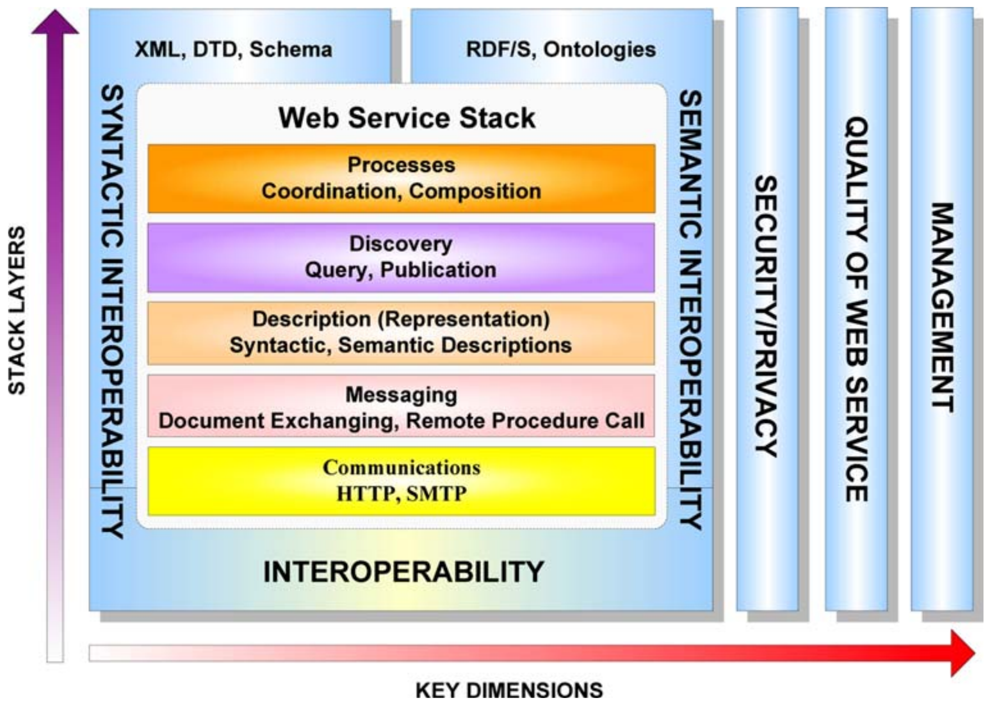

Fig. 3 Web service stack and key dimensions

government agencies). All entities that agree on using a given ontology commit themselves to the concepts and definitions within that ontology. "Explicit" means that the concepts used in an ontology and the constraints on their use are explicitly defined. "Formal" intends that the ontology should be machine understandable and describe using a well-defined model or language called *ontology language*. *Mediators* provide an integrated view or mediated schema over multiple heterogeneous and autonomous services [46]. This schema represents generally a synthesized view over a specific application domain. Users access the integrated view through a uniform interface. Each service is connected to a wrapper that enables its participation in the system. It translates between the serviceć6s concepts and those at the mediator level.

Interoperability is the core functionality that Web services endeavors to achieve. Interoperation occurs at two levels: *syntactic* and *semantic*.

– *Syntactic interoperability* is concerned with the syntactic features of Web services. Examples of syntactic features include the number of parameters defining a message and the data types of those parameters. XML helps achieve syntactic interoperability by encoding syntactic information into XML documents. Additionally, XML provides platform and language independence, vendor neutrality, and extensibility, which are all crucial to interoperability.

– *Semantic interoperability* is the most challenging issue for achieving the truly seamless interoperation. It deals with semantic properties of Web services. Examples of semantic features include the domain of interest of a Web service and the functionality provided by an operation. The envisioned Semantic Web is gaining momentum as the potential silver bullet for empowering Web services with semantics.

Interoperability could be achieved at different layers as depicted in Fig. 3 [65]. *The communication layer* provides protocols for exchanging messages among remotely located partners (e.g., HTTP, SOAP). It is possible that partners use different proprietary communication protocols. In this case, gateways should be used to translate messages between heterogeneous protocols. The objective of interoperability at this layer is to achieve a seamless integration of the communication protocols. *The content (description) layer* provides languages and models to describe and organize information in such a way that it can be understood and used. Content interoperability requires that the involved systems understand the semantics of content and types of business documents. For instance, if a Web service receives a message that contains a document, it must determine whether the document represents a purchase order or a request for quotation. Information translation, transformation, data mediators, and integration capabilities are needed to provide for reconciliation among disparate representations, vocabularies, and semantics. The objective of interoperability at this layer is to achieve a seamless integration of data formats, data models, and languages. *The process layer* is concerned with the conversational interactions (i.e, joint business process) among services. Before engaging in a transaction, the service providers need to agree on the procedures of their joint business process. The semantics of interactions among partners must be well defined, such that there is no ambiguity as to what a message may mean, what actions are allowed, what responses are expected, etc. The objective of interoperability at this layer is to allow autonomous and heterogeneous partners to come online, advertise their terms and capabilities, and engage in peer-to-peer interactions with any other partners. Examples of concepts for enabling process interoperability include process wrappers and application adapters [19,13].

Security & privacy Security is an important issue for deploying Web services. Web services enable interoperation at the risk of letting outside intruders attack the internal applications and databases since they open up the network to give access to outside users to these resources [42]. Web service security needs to be concerned with the following aspects: *authentication*, *authorization*, *confidentiality*, and *integrity*. *Authentication* is used to verify a claimed identity while *authorization* is to check whether a user is authorized to perform a requested action. *Confidentiality* is to ensure that information is disclosed only to authorized recipients by, for example, encrypting the message. Lastly, *integrity* refers to the protection of the information from being tampered with by, for example, putting digital signatures on the messages. Privacy is another major concern of Web service deployment [89]. During service interactions, personal data or business secrets (e.g., billing information, shipping address, or product preference) might be unintentionally released [5]. Conventional privacy protection mainly relies on law enforcement and restriction of social values. Emerging technologies for preserving privacy in Web services include digital privacy credentials, data filters, and mobile privacy preserving agents [91].

QoWS The proliferation of Web services is expected to introduce competition among large numbers of Web services that offer similar functionalities. The concept of QoWS is considered as a key feature in distinguishing between competing Web services [104]. *QoWS* encompasses different quality parameters that characterize the behavior of a Web service in delivering its functionalities. These parameters can be categorized into two major quality classes: *runtime* quality and *business* quality. Quality parameters in these two classes can be further extended to include more quality aspects of Web services to fulfill the requirements of different application domains, such as *Accessibility*, *Integrity*, and *Regulatory*.

Management Web service management refers to the control and monitoring of Web service qualities and usage. Web service management mechanisms are highly coupled with the QoWS of a Web service. We identify two types of management: *control* and *monitoring* management.

- *Control management* aims to improve the service quality through a set of control mechanisms. Typical control mechanisms include Web service transaction, Web service change management, and Web service optimization. Transactions help improve the reliability and fault-tolerance of Web services. Change management deals with the highly dynamic environment of Web services. It takes a series of actions to identify the changes, notifying the coupled entities, and adopting appropriate operations to response the change. Web service optimization helps users identify Web services and/or their combinations to best fulfill their requirements.
- *Monitoring management* rates the behavior of Web services in delivering its functionalities in terms of each QoWS parameter. Monitoring Web service behavior would be crucial in either calculating the QoWS parameter values or assessing a Web service claim in terms of promised QoWS.

Control and monitoring management might sometimes work cooperatively. For instance, the Web service optimization would need the monitoring process to get the QoWS parameter values of different Web services and/or their combinations. These values would guide the optimization process to return the optimized solutions that best fulfill users' requirements.

3.5 The WSMS architecture

In this section, we present the WSMS architecture. The design of the WSMS architecture leverages the research result in DBMSs. Web services will be treated and manipulated as first-class object in the proposed WSMS. The key components in this architecture are modeled after those in DBMSs. The functionality of each component aims to address the issues raised by the key dimensions. The interoperation framework consists of six subcomponents: communication, WS messaging, WS discovery, service registry, WS representation, and WS processes. The collaboration of these

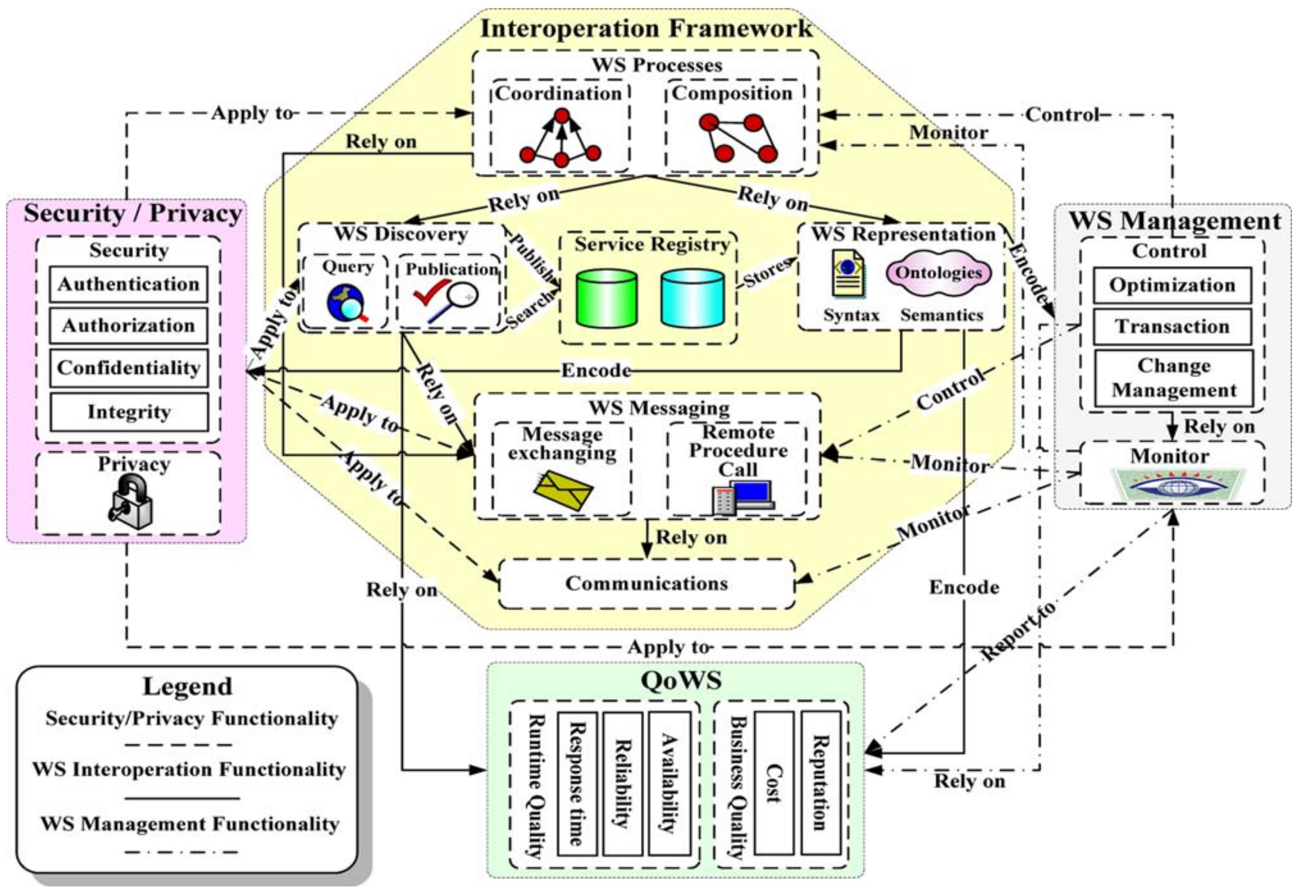

Fig. 4 The WSMS architecture

subcomponents provides mechanisms to efficient access and interoperation with Web services. The security/privacy component guarantees that the access and interoperation can be conducted in a secure and controlled manner. The QoWS component lays out a set of quality metrics that can be used to advertise and discover Web services. Some of the metrics can also be used to specify quality level agreement, such as payment, price, etc. The management component offers monitoring, transaction, change management, and optimization functionalities. The proposed WSMS provides value added features to enable reliable and optimized deployment of Web services. The architecture also reflects the relationship among different components. In what follows, we give a detailed description of the major functionality of each component (Fig. 4).

The WS Interoperation Framework The interoperation framework is at the core of the WSMS architecture. It addresses the interoperability issue of Web services through the collaboration of its six subcomponents. WS-messaging combines with an underlying communication protocol (e.g., HTTP and SMTP) to enable basic interaction with Web services. A Web service takes the incoming message as the input to one of its methods and responds with the output of the method as a returning message. WS-representation defines the Web service interface containing a set of supported methods. It specifies the signature of each method, which is similar to IDL in the middleware systems. However, WS-representation goes beyond the IDL-like syntactic service description. It incorporates more expressive language constructs (e.g., ontologies) for describing the properties and capabilities of Web services in an unambiguous and computer-interpretable manner. Other information could also be specified in WS-representation, such as quality of service parameters, security and transaction requirements, etc. The semantic service description caters for the loosely coupled interoperation between Web services. It helps Web services determine the functionality, requirement, and quality of their interoperation partners. Descriptions of Web services are stored at service registries. WS discovery provides the query and publication functionalities for locating and publishing Web services in a service registry. Interaction with the registry is through WS messaging. WS processes rely on the basic functionalities provided by the discovery, presentation, and messaging components to support the complex interactions between Web services. The processes involve the invocation of a sequence of Web services. Service coordination defines the external interaction protocols for a WS process, whereas service composition defines the schemas for its internal implementation.

The WS security/privacy component The security/privacy component ensures that interactions with Web services are conducted in a secure fashion while sensitive information can be preserved as required. The security mechanisms need to be applied to all aspects of Web services, including messaging, query, publication, coordination, composition, control, and monitoring. Typical security functionalities that can be implemented by the security module are auditing, authentication, access control, and data encryption. Privacy is usually expressed by policies which reflect that the habits, behaviors, actions or other rights of the service users must be protected. Instead of relying on laws and social values, the privacy module enforces the policies from a technological perspective.

The QoWS component The QoWS component records the quality aspects of a Web service. It reflects the runtime and business requirements of Web services, such as response time, availability, reliability, cost, and reputation. Because it is anticipated that there will be multiple competitors to provide similar functionalities, the WS query process uses QoWS as a major criteria to select the "best" Web services. The QoWS component provides functionalities to define appropriate metrics to characterize *QoWS* and devise techniques to use it in optimizing service-based queries.

The WS management component The WS management component is for monitoring and controlling the interactions with Web services. The monitoring module examines the behavior of the underlying communication, WS messaging, and WS processes, and reports the runtime and business properties of Web services to the QoWS component. The control module provides transaction, optimization, and change management mechanisms to deliver the functionalities of Web services in a reliable, adaptive, and optimal fashion.

Figure 5 describes the proposed WSMS architecture using the W3C concept map model [112]. In this figure, rectangles represent concepts and lines with arrows represent relationships. The key components in the WSMS architecture are represented by the concepts. In addition to these key components, the concept map also includes the Web service (represented by WS) concept. The functionalities of each component are reflected by its relationships with the WS concept or other components.

The proposed WSMS provides a foundational framework for the Web services. It formalizes the steps in the entire Web service life cycle. The design of each component in the WSMS architecture follows key research issues (called dimensions) in Web service environments. Since the Web services are designed to achieve seamless interoperability, the interoperation framework stays at the core of the WSMS architecture. The framework is composed of several horizontally separated layers; each layer contains components that provide corresponding interoperation functionalities. The security/privacy, QoWS, and management components offer supplementary support (e.g., security, privacy, control, and monitoring) for the interoperation framework. These functionalities are orthogonal to the horizontal layers in the interoperation framework; they can be applied across these layers. This is different from the *Extended Service Oriented Architecture* (ESOA) [82]. The ESOA contains three horizontal layers: basic service, service composition, and management layers. The separation of different layers is based on the advancement of their functionalities. The basic service layer provides simple functionalities such as messaging, description, and discovery. The composition layer provides the functionalities for the consolidation of multiple service into a single composite service. The management layer provides advanced administration capacities for managing critical Web service-based applications. Additionally, the ESOA extends the *Service Oriented Architecture* (SOA) by addressing the new requirements introduced by Web services. The layers added in ESOA provide functionalities that overlap with existing SOA layers. However, the WSMS architecture is not developed by adding layers to an existing architecture. We take consideration of the key research issues of Web service when designing the WSMS architecture. Therefore, there is a clear functionality separation of different components in a WSMS.

In [80], Web service management approaches have been investigated to support production-quality Web service applications. The Web service management framework relies on a manageability information model and management infrastructure services to make Web services measurable and manageable. Specifically, the manageability information model describes the manageability information of Web services. Management infrastructure services define standard interfaces for the management functionalities, such as metering, monitoring, mediation, etc. We take an integrated approach and a broader scope to present the WSMS architecture. Management approaches investigated in [80] are complementary to those covered in the management component and fit into the the proposed WSMS

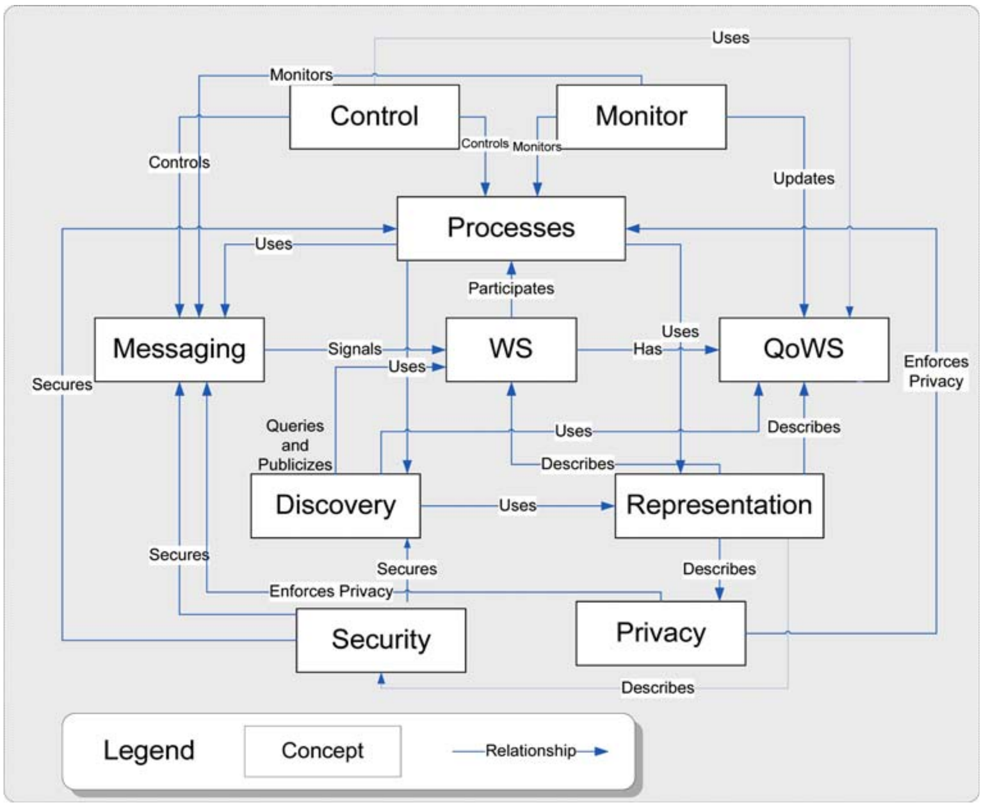

Fig. 5 A concept-map description of the WSMS architecture

architecture. There are some other research effort underway to provide architectural support for Web services such as Web Service Management Framework (WSMF) [39], Web Service Architecture (WSA) [112], Web Services Conceptual Architecture (WSCA) [52], Semantic Web enabled Web Services architecture (SWWS) [22], Service Bus [57], and Web Service Interoperability framework (WS-I) [118]. These architectures have addressed several similar components covered in our WSMS architecture. These components are designed to address the key research issues in the entire Web service life cycle. Unlike these architectures that present the components in an isolated manner, we take an integrated approach to investigate the functionalities of each component in the proposed WSMS architecture. We further examine the relationship between these components and how they could collaborate to set up the WSMS to enable the entire Web service life cycle including developing, deploying, publishing, discovering, composing, monitoring, and optimizing access to Web services.

4 Web service interoperation framework

Web services are designed to bring together *distributed* and *heterogeneous* applications in a large *scale* and provide interoperability between them [94]. *Distribution, heterogeneity*, and *scalability* are the three key issues for application interoperation. Computer networks bridged isolated machines to form a *distributed* system. Different applications can thus build up communications via various network protocols. This enables the primary form of interoperation between applications. Interoperation at this stage requires complex low-level programming, which is time consuming and error prone. Remote Procedure Calls (RPCs) take a step further by hiding the complexity of building distributed applications. They abstract away the *heterogeneity* of applications from different aspects, such as platforms and languages. RPC enables simpler and more efficient interoperation between applications. A middleware takes RPC as its kernel technology and further evolves it by incorporating security, faulty tolerance, and failure recovery mechanisms. It provides a full-fledged interoperation solution in an intra-organization *scale*. Because organizations are rushing to move their main operations to the Web, there is a need for interoperation of applications from a much wider spectrum. Web services are the result of the various standardization efforts to enable application interoperation on the Web.

Web service interoperation occur at two levels: syntactic and semantic levels. We evaluate the key technologies that support syntactic and semantic interoperation in this section. These technologies will be mapped onto the corresponding layers in the Web service stack based on their functionalities: WS messaging, WS representation, WS discovery, and WS processes. After presenting the supporting technologies at each layer, we follow with a discussion of how further development of these technologies can address the current open issues.

4.1 Messaging layer

Web services are heterogeneous and loosely coupled. The Web service environment can no longer fulfill the symmetrical requirement of conventional middleware, where communication protocols link uniform object models and libraries. In addition, Web services may be deployed at different organizations across the Internet, which requires the communication protocols to work through firewalls. The use of document-based messaging model caters for the loosely coupled interaction relationships. To deal with heterogeneity, the messaging model works with a wide variety of transport protocols. For example, interaction with Web services that sit behind firewalls requires the messaging model to be combined with HTTP. In another case, combination with SMTP enables the interaction with Web services that support asynchronous message exchanges [9]. In addition to the document style interactions, the messaging model also supports RPC style interactions. This adapts Web services to the legacy applications developed on the conventional middleware, which communicate with RPCs. In this section, we present several technologies that support WS messaging: SOAP, WS addressing, and WS routing.

SOAP [109] SOAP is a WS messaging standard that enables communication among Web services in a WSMS. It provides a lightweight messaging framework for exchanging XML-based messages. SOAP is independent of languages and platforms. It supports two types of communications: messaging and RPCs. It is designed to work with a great variety of transport protocols. The predominant protocol that is combined with SOAP is HTTP, which helps achieve the synchronous communication. A SOAP message contains one XML element (`Envelope`) and two child elements (`Header` and `Body`). The `Envelope` defines the namespaces for the remaining content of a SOAP message. The `Header` is an optional element. It can carry auxiliary information in a SOAP message for supporting authentication, transaction, and payment. The `Header` enables a SOAP message to be extended in an application-specific manner. The `Body` is the mandatory part of a SOAP message. It specifies the information to be carried from the initial message sender to the ultimate message receiver. For message exchanging, the `Body` part encompasses the exchanged information, while for RPCs, `Body` includes the remote method name, its associated address, and the arguments thereof.

WS-routing [106] SOAP does not specify the actual message path along which a SOAP message is to travel. The message path consists of several intermediaries that a SOAP message will visit. SOAP needs to rely on the underlying transport protocols and follow their message path models. Therefore, it is impossible to specify which intermediaries the SOAP message can visit when it travels to its destination. WS-routing enables SOAP path to be specified as a SOAP message header. It consists of a sequence of references to the intermediary SOAP nodes. The sequence specifies the visiting order of the SOAP message. When a message arrives at an intermediary node, the node will remove the reference thereof and send the message to the next node based on the specified order. An intermediary node can also incorporate new nodes or remove existing nodes to change the message path.

WS-Addressing [69] WS-Addressing defines a protocol-neutral mechanism to address WS messages. It helps identify WS endpoints through predefined XML elements. It specifies an endpoint reference as the combination of an address and several reference properties. Thus, the target WS instances can be uniquely located. WS-Addressing defines how WS instances endpoints are described. It also specifies how address description can be encoded in SOAP messages. The SOAP messages will carry standardized header blocks, which can identify a unique WS instance.

WS-Reliability [98] WS-Reliability is a SOAP-based protocol for addressing reliable messaging requirement. The reliability features include guaranteed delivery, duplicate elimination, and message ordering. WS-reliability specifies how to reply to a message to guarantee reliable messaging. An acknowledge message or a failure message is sent as a reply. The reply is correlated with a message by references to the message ID. A message ID is globally unique. It is also used in duplication elimination. Messages with the same message ID are treated as duplicates. WS-reliability uses a sequence number mechanism to track and enforce the message order. WS-reliability is defined as the SOAP head extensions.

SOAP and its complementary specifications provide a simple and lightweight messaging framework for exchanging XML-based messages between Web services. However, the downside of XML-based messaging is that it represents information in a rather expensive way, which demands intensive document parsing. When a large number of Web services participate in the interaction, the message exchange would become very frequent. This could greatly reduce the performance, which brings along the scalability problem. Two alternatives to the XML-based types are attaching binary data to the SOAP messages or using URLs that point to the actual data. Another trend that extends SOAP would be along with the asynchronous message exchange. The RPC style interaction requires tightly coupled relationships, where applications are strongly dependent on each other. When interactions occurs across multiple organizations, the coordination would become significantly challenging. Asynchronous interactions avoid direct invocations by batching and routing requests via queues. Therefore, asynchronous SOAP would be a technological choice to achieve cost effective interactions.

4.2 Representation layer

The representation layer is used to describe Web services. It wraps Web services and specifies their functionalities, operations, data types, and binding information using a service interface. The discovery layer relies on the representation layer to locate appropriate Web services. Since XML and ontologies are the two predominant languages for defining the service interfaces, we identify two types of Web service representations: XML-based and ontology-based.

4.2.1 XML-based representation

In this section, we describe a general framework for the XML-based representation that is symbolized by *WSDL*. We will then describe more application-specific frameworks that facilitate interoperabilty at the representation layer.

WSDL [113] WSDL is the current industry standard for WS description. It goes beyond IDLs in the conventional middleware in two major ways. First, WSDL specifies the mechanisms to access a Web service in addition to the service name and signatures. Second, WSDL defines the location to invoke a Web service, whereby the service requestor can locate the service and interact with it using SOAP. WSDL mainly focuses on the syntactic description of Web services using XML. It does not allow the specification of Web service semantic features. For example, no constructs are defined to describe document types (e.g., whether an operation is a request for quotation or a purchase order).

A WSDL document describes programming interfaces and accessing formats of a Web service. It defines a Web service at two levels: the *application* level and the *concrete* level. WSDL helps service requestors access a Web service through a clear separation of the abstract and concrete descriptions of a Web service. At the abstract level, the WSDL description includes three basic elements: `Type`, `Message`, and `PortType`. `Types` define the type of the data that are relevant for the information exchange. Although WSDL can support any type system, it adopts XML Schema Definition (XSD) as the canonical type system to achieve maximum interoperability and platform neutrality. `Message` represents an abstract definition of the transferred data. A message consists of one or more logical parts. Each part associates with a type of XSD or other type systems. These logic parts provide a flexible way to describe the abstract content of messages. `PortType` is a set of abstract operations provided by an endpoint of a Web service. An endpoint of a service can support four transmission primitives: *one-way*, *request–response*, *solicit–response*, and *notification*. At the concrete level, the WSDL description provides information of binding a concrete service endpoint. It specifies three aspects of binding information: communication protocols, data format specifications, and network addresses. `Binding` describes the first two aspects. It specifies the communication protocol, such as SOAP and HTTP. It also specifies the data format of the operations and messages. `Port` and `Service` elements describe the network addresses. A `Port` specifies a single address for binding a service endpoint. A `Service`, on the other hand, aggregates a set of related ports.

The WSDL documents describe primary information for accessing Web services. WS-Policy [96] and WS-Agreement [44] are two specifications that complement WSDL. WS-Policy provides a framework to enable Web services to specify policy information. The policy is used to convey conditions on an interaction between two Web service endpoints. WS-Policy enables the Web service provider to specify a condition under which it provides the service. The Web service requester uses the policy to convey the information that which kind of service it wants. WS-Agreement provides the mechanisms for service providers to specify agreement terms for the usage of their services. These specifications would be used for customizable description of web services.

Other XML-based frameworks There is a large number of XML-based frameworks for enabling interoperability at the representation layer. Interactions between different partners require that the involved systems understand the semantics of content and types of the exchanged documents. In what follows, we describe a representative set of XML-based interoperability frameworks. An exhaustive list of frameworks can be found in [20]. Web services may adopt these frameworks to capture the semantics of documents processes. For example, Web services may use cXML to carry out interactions among businesses according to these frameworks [20].

eCO introduces *xCBL* (XML Common Business Library) to define business documents [20]. xCBL consists of a set of XML core documents that are used to represent common interactions in business transactions. The main motivation for establishing core documents is that some concepts are common to all business domains and thus can be expressed in a common format. Examples of such core documents are: purchase orders, invoices, date, time, and currencies. Business partners may use and extend these documents (e.g., adding new elements) to develop their own business documents. Businesses are not limited to a specific set of pre-defined documents. However, this may hamper interoperability since companies would need to be aware of newly created documents.

cXML (*Commerce XML*) consists of an XML-based schema language and a protocol for online purchasing transactions [20]. It targets business transactions that involve non-production Maintenance, Repair, and Operating (MRO) goods and services. cXML defines a set of XML DTDs to describe procurement documents in the same spirit as xCBL (e.g., order request, order response). It provides the following elements for describing product catalogs: Supplier, Index, and Contract. The supplier element describes general information about a supplier (e.g., address, ordering methods). The index element describes the supplier's inventory (e.g., product description, part numbers, classification codes). The contract element describes the negotiation agreements between a buyer and a supplier on product attributes (e.g., price, quantity).

Universal Business Language (UBL) defines a generic XML interchange format for business documents [73]. It aims to resolve the discrepancy in XML documents used in different application domains. UBL consists of three key components: a library of reusable data components, a set of common business documents, and customization of UBL for specifying trading relationships. The data library and the common business documents are both defined as XML schemas. The standardization of the XML business schemas provides a commercial syntax for business documents that can be universally understood and recognized.

RosettaNet [3] aims at standardizing product descriptions and business processes in information technology supply chain applications. RosettaNet's supply chain include information technology products (e.g., boards, systems, peripherals, finished systems) and electronic components (e.g., chips, connectors). The *RosettaNet Business Dictionary* contains vocabulary that can be used to describe business properties (e.g., business name, address, tax identifier). The *RosettaNet Technical Dictionary* contains properties that can be used to describe characteristics of products (e.g., computer parts) and services (e.g., purchase order). The *RosettaNet Implementation Framework* specifies content of messages, transport protocols (HTTP, CGI, email, SSL) for communication and common security mechanism (digital certificates, digital signatures).

ebXML (*Electronic Business XML*) [34] defines an architecture and a set of specifications for Web service interactions. It is sponsored by UN/CEFACT and OASIS. At the content layer, companies interact through *business documents*. A *business document* is a set of information components that are interchanged as part of a business process. Business documents are composed of three types of components: *core components*, *domain components*, and *business information objects*. *Core components*, stored in the *core library*, are information components that are re-usable across industries. *Domain components* and *business information objects* are larger components stored in the *domain library* and *business library* respectively. Core components are provided by the ebXML library while domain component and business information objects are provided by specific industries or businesses.

4.2.2 Ontology-based representation

As a non-trivial extension to WSDL, ontologies enable the semantic description of Web services. Ontologies originate from the AI field. They are mainly used to model knowledge-based systems. A major distinction between Web services and knowledge systems is that Web services are dynamic. Therefore, using ontologies to model the dynamic service processes and behaviors would be an important future trends in both Web service and ontology areas [30,29]. The ontology-based WS representation enriches the Web service description with machine interpretable semantics. It uses the ontology language constructs to describe the functionalities of services. WS-discovery can locate appropriate Web services based on the functionality information of Web services. The ontology-based model can also specify the behavior of service operations, including the preconditions and effects of performing an operation of a Web service. WS-processes can rely on the behavior information to decide the order of different Web services that participate in the process.

OWL-S [27] OWL-S is an OWL-based ontology for describing Web services. OWL (*Ontology Web Language*) is a revision of the DAML+OIL web ontology language incorporating lessons learned from the design and application of DAML+OIL. OWL-S (an extension of DAML-S, a DAML+OIL upper ontology for Web service) aims at setting up a framework for semantic Web services. OWL-S depends on ontologies to realize the automatic service discovery, invocation, composition and interoperation, and execution monitoring. The `Service` class sits on top of the hierarchical structure. It consists of three components: a `ServiceProfile`, a `ServiceModel`, and a `Service Grounding`.

The service profile represents the capacities of the service. It helps a searching agent discover services that fulfill certain requests. Specifically, an OWL-S profile contains three types of information: information about the service provider, functional description, and a host of service characteristics. The provider information contains the contact information of the service providing entity. The functional description specifies the transformation introduced by the service. It provides the required inputs and the generated outputs. It also specifies the preconditions needed to execute the service and the expected execution effects.

The service model describes the execution process of the service. OWL-S defines the `ProcessModel`, as a subclass of `ServiceModel` to give a detailed perspective on how a service operates. A process can generate two kinds of effects: data transformation and state transition. For data transformation, the process receives some inputs and produces some outputs, whereas for state transition, the process starts based on some preconditions and produces some effects. The process model consists of three types of processes: *atomic*, *simple*, and *composite*. An atomic process is directly invocable by receiving an input message and returning an output message. The atomic process needs to be grounded to a WSDL operation to enable the execution. Similarly, the input and output of the process also needs to be grounded to the corresponding WSDL messages. A simple process mainly serves as a tool for abstractions. It may provide a view of some of the specialized uses of atomic processes. It may also simplify the presentation of some composite services for planning and reasoning. Simple processes are not grounded to any WSDL operations, so they are not invocable. A composite service combines several other processes (either composite or non-composite) with a set of control constructs. OWL-S defines a minimal set of control constructs, which consist of Sequence, Split, Split + Join, Choice, Unordered, Condition, If-Then-Else, Iterate, Repeat-While, and Repeat-Until. A composite service is not associated with a grounding. Therefore, it is not invocable either.

A grounding enables access to the service. It specifies the description elements needed to interact with the service. The grounding specifies the concrete level of the service description. In contrast, the service profile and service model specify the abstract part of the service description. The core function of a grounding is to realize the abstract inputs and outputs of an atomic process as concrete messages. The messages serve as a vehicle to carry the inputs and output in some transmittable format. OWL-S depends on WSDL to ground its atomic processes due to the similarity between the grounding and WSDL's binding.

WSMF and **WSMO** [39,48] Web Service Modeling Ontology (WSMO) is an ontology for describing several aspects of semantic Web services. It takes Web Service Modeling Framework (WSMF) [39] as a starting point. WSMF consists of four main parts, including *goals*, *ontologies*, *mediators*, and *Web services*. The *goal* defines the problems that a Web service is expected to address. The *ontology* defines the formal semantics for the terms used in other elements of WSMF. The *mediator* is used to address interoperability problems, such as data mismatches and process sequence mismatches. The *Web service* part defines several elements to describe a Web service, including the precondition, post-condition, data flow, control flow.

The WSMO refines WSMF and defines these elements in a formal fashion. The WSMO definition of a Web service consists of four parts, including *nonFunctionalProperties*, *usedMediators*, *capability*, and *interface*. The `nonFunctionalProperties` describe the properties that are not directly related to a Web service's functionality, such as service providers, cost, performance, reliability, security, etc. These properties are mainly used to help software agents discover and select Web services. They can also be used for negotiation. The `usedMediators` define the mediators used by a Web service. For example, a Web service can use the concepts and relationships elsewhere by importing ontologies through ontology mediators (`ooMediators`). It can also use `wgMediator` to address interoperability problems between a Web service and goals. The `capability` defines the functionalities of a Web service. It helps software agents locate a desirable Web service. A capability can be modeled by using preconditions, postconditions, and effects. The `interface` describes how Web service functionalities can be fulfilled. It can be used to help software agents invoke and combine Web services. The WSMO definition describes the interface of a Web service from a twofold view, including *choreography* (from user's prospective) and *orchestration* (form other service provider's perspective). The choreography describes the information about how to interact with a service to make use of its functionalities, such as Message Exchange Pattern (MEP). The orchestration describes the information about how a Web service is outsourced to provide a value-added service. It has a tight relationship with the Problem Solving Pattern (PSP), which specifies a sequence of activities to achieve a given requirement. The *Web Service Modeling Language* (WSML) [47] provides a formal syntax and semantic for WSMO. It is based on well-known logical formalisms, including first-order logic, description logic, and logic programming.

In a nutshell, the XML-based service representation describes the syntactical information of Web services. Similarly to IDLs, it specifies the input and output format of a set of operations offered by a Web service. However, the XML-based service representation does not usually provide rich semantics of services, which are paramount for discovery and composition of loosely coupled Web services. The ontology-based service representation enriches the Web service description with machine interpretable semantics. It uses the ontology language constructs to describe the functionalities of services. In addition, it can also specify the behavior of service operations, including the preconditions and effects of performing an operation of a Web service. Since different ontologies might be used for service description, a major issue of ontology-base a service representation is mapping between disparate service ontologies.

4.3 Discovery layer

The discovery layer provides a query and publication framework for publishing and locating Web services. It enables the usage of Web services in a much wider scale. Service providers can store the service descriptions in a service registry via the publication functionalities provided by WS discovery. Meanwhile, service requestors can query the service registry and look for their interested services based on the stored service descriptions. The standardization effort that aims to provide a WS discovery framework is UDDI.

UDDI [110] UDDI defines a standard to publish and query Web services in a WSMS. It integrates Web service description and discovery to help service requestors locate their desirable services. The core component of UDDI is a service registry. The registry stores the general information of Web services. The information can facilitate service requestors in querying different Web services. UDDI provides a service query API to locate appropriate Web services. It also defines a service publication API for service providers, who can use it to advertise their services.

UDDI encodes three types of information about services: *white pages*, *yellow pages*, and *green pages*. The white pages contain the contact information of business entities and the Web services they provide. UDDI uses two elements to express white pages information, including `businessEntity` and `businessService`. The `businessEntity` describes a business organization that provides Web services. It specifies the information of service providers, including names, brief description, and contact details. Each `businessEntity` may include one or more `businessService` elements. The `businessService` describes a collection of related Web services offered by an organization specified by a `businessEntity`. The yellow pages contain the classification information of businesses and services. Different business or services are classified based on their functionalities. Examples of such categories include restaurants, hotel, etc. The yellow pages specify the category to which the businesses or services belong. The green pages contain the information about technical details of Web services, including `bindingTemplate` and `tModel`. The `bindingTemplate` describes the information about concretely binding an endpoint of a service. The `tModel` describes the technical specification of a Web service, such as a Web service type, a protocol used by Web services, or a category system.

UDDI provides well-defined programming interfaces for users to inquire and update the service registry data. The inquiry API and publication API are both based on SOAP messages. The inquiry API enables service requesters to find and get details about Web services. The publication API enables service providers to publish services, update services information, and update the security mechanism.

The major usage mode of current Web service discovery technologies is design time discovery [9]. Users query the service registry, retrieve the service description, and write the client code to invoke the service. To accessing Web services at large, it requires WS discovery to support dynamic binding. However, a significant challenge for dynamic binding is to unambiguously describe Web service using machine interpretable languages. In addition, issues such as fault-tolerance and load-balancing also need to be addressed [9].

Semantic UDDI [78] An approach for empowering the UDDI registries with semantic capabilities is presented in [78]. It shows how DAML-S Service Profiles can be mapped into the UDDI records providing a way to record semantic information within a UDDI registry. It also shows how the UDDI registries can be modified to use the semantic information provided by DAML-S. The proposed approach identifies two types of attributes in a DAML-S Service profile: UDDI-like and DAML-S specific attributes. The UDDI-like attributes are mapped directly from the DAML-S Service Profile to UDDI records as they already exist in UDDI. This is the case with provenance information such as the name and address of the service provider. The DAML-S specific attributes, such as geographicRadius, qualityRating, precondition, and effect, are represented using the tModel mechanism defined in UDDI. The mapping of DAML-S specific attributes requires the specification of a set of fifteen (15) UDDI tModels, one for each attribute. BusinessService records use these tModels to index the values they store from the DAML-S Service profile they intend to represent. One of the tModels, the DAML-S tModel, has a special meaning; it states that the service advertised has a DAML-S service representation, and its value is the URI of the DAML-S service that is represented by the current profile. A DAML-S matching engine uses ontology-based information for search requests to obtain the UDDI keys which are in turn used to retrieve the service descriptions from the UDDI registries.

ebXML Registry [34] Another example for a Web service registry standard is the ebXML (electronic business XML) registry. The ebXML registry acts as a database for data regarding business to business communication. It follows a similar concept like the UDDI registries, but is broader in scope. An ebXML registry is capable of storing arbitrary data including XML documents, text documents, images, sound and video [33]. Instances of such content are referred to as a *RepositorytItems*. RepositorytItems are stored in a content repository provided by the ebXML Registry. In addition to the RepositoryItems, an ebXML Registry is also capable of storing standardized metadata that may be used to further describe RepositoryItems. Instances of such metadata are referred to as a RegistryObjects.

The ebXML registry includes two major specifications. The *ebXML Registry Information Model* specification defines the types of metadata and content that can be stored in an ebXML Registry. The companion document *ebXML Registry Services and Protocols* defines the services provided by an ebXML Registry and the protocols used by clients of the registry to interact with these services. The ebXML registry offers two separate interfaces, the *LifeCycleManager Interface* and the *QueryManager Interface*. The LifecycleManager handles the submission of objects, the classification schemes of object and the removal of obsolete objects from the registry. The QueryManager interface enables the discovery of Web services. It consists of two parts allowing search with SQL expressions and Filter expressions, respectively.

WS-MetadataExchange [4] Web services use metadata (i.e., service description information) to describe what other Web services need to know to interact with them (e.g., WSDL description). The Web services metadata exchange (WS-MetadataExchange) enables the retrieval of metadata associated with a Web service. While service registries, such as UDDI and ebXML registries, are used to store metadata about Web services, WS-MetadataExchange is intended for the retrieval of such metadata. However, it is not intended to provide a general purpose query or retrieval mechanism for other types of data associated with a service, such as state data, properties and attribute values. The WS-MetadataExchange defines two request–response interactions for getting service's metadata: *Get* and *Get Metadata*. When the type of metadata sought is clearly known, (e.g., WSDL), a requester may indicate that only that type should be returned by using the *Get* request. When additional types of metadata are being used, or are expected, or when a requester needs to retrieve all of the metadata relevant to subsequent interactions with an endpoint, a requester may indicate that all available metadata, regardless of their types (e.g., WSDL, WS-Policy, etc.), are expected. This is done by sending a *Get Metadata* message. The WS-MetadataExchange specification is an initial public draft release and is up for review for later submission to a standards body, likely OASIS.

4.4 Process layer

The messaging, representation, and discovery layers achieve simple interactions with Web services, where service requestors invoke a single operation. The process layer supports more complex interactions between Web services. It relies on the basic interaction functionalities provided by the technologies at lower layers in the Web service stack. For example, it needs the discovery and representation layers for querying and locating Web services based on their descriptions. The selected Web services are used to construct the process, which consists of a sequence of coordinated Web services. The benefits offered by the process layer lie in four major aspects. First, Web service processes enable organizations to outsource existing Web services, which avoids developing new applications from scratch and ensures a rapid time-to-market deployment. Second, it reduces complexity, because complicated services can be incrementally constructed out of relatively simple designs. Third, application development based on Web services reduces business risks as reusing existing services avoids

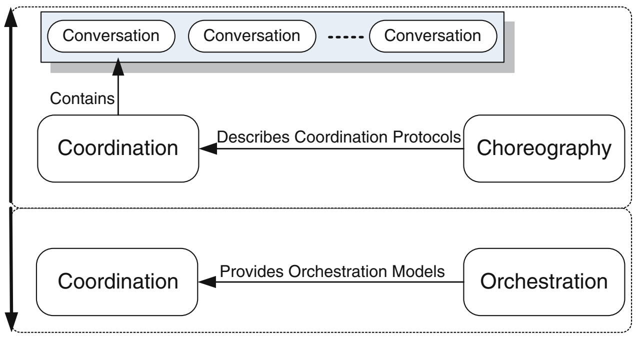

Fig. 6 Key concepts in the process layer

the introduction of new errors. Finally, the possibility of outsourcing the "best-in-their-class" services allows companies to increase their revenue.

Figure 6 illustrates the relationships between the key concepts in the process layer. Web service coordination and composition offer technological support to construct complex Web service processes. *Coordination* and *composition* are highly related but focus on different aspects of a process. Coordination provides external protocols for enabling collaboration among multiple Web services, whereas composition refers to the process of developing a composite Web service. A *composite* Web service is defined as a conglomeration of outsourced services [67]. Three concepts that are related to coordination and composition are *conversation*, *choreography*, and *orchestration*. A conversation is a sequence of message exchanges between a service and its requester [17]. A coordination protocol contains a set of justified conversations. Coordination protocols are described by a choreography, which tracks the messages exchanged by multiple Web services. Although a choreography involves multiple parties, no single party has a full control of the conversations. Orchestration refers to an executable Web service process. It is controlled by a single party. An orchestration model (e.g., statecharts, Petri Nets, pi-calculus) specifies the order and conditions to invoke the component services in a composite service achieved through the composition process.

4.4.1 Web service coordination and choreography

Coordination protocols specify the procedures that participant Web services need to follow to achieve their interoperation. They are analogous to service interfaces, which describe how to interact with Web services. The implementation details are hidden from service requestors. Coordination protocols are public documents which can be published in some service registries. This enables design-time discovery and runtime binding. Coordination protocols are described by a choreography. Current proposals that support Web service coordination and choreography including WS-Coordination, WS-CDL, and BPSS.

WS-coordination [74] WS-Coordination aims to setup a framework for supporting coordination protocols. The framework includes two major components: coordinator and participants. The participant relies on the coordinator to interact with other participants. WS-Coordination defines a coordination protocol, a coordination type, and a coordination context to specify the interactions between the participants and their corresponding coordinators. Specifically, a coordination protocol defines a set of rules to guide the interaction between a coordinator and its participants. A coordination type is a collection of related coordination protocols. A coordination context specifies a data structure, which marks the messages that belong to the same coordination. There are three modes of interactions between the participants and their corresponding coordinators: activation, registration, and protocol-specific interactions. In the activation mode, a participant sends a request to its corresponding coordinator for setting up a new coordination context. Once the participant initiates an instance of coordination type, the new context is created. In the registration mode, a participant registers with a coordinator to join the execution of a coordination protocol. The coordinator will notify the participant when the coordination protocol is being performed. In protocol-specific interactions mode, the participants and their coordinators exchange protocol-specific messages. The three modes of interactions are supported by either of the two architectures: central and distributed coordinations. In the former, all participants communicate with a single coordinator, whereas in the latter, each participants communicate with its own coordinator. WS-Coordination identifies the key component for building a coordination framework. However, it does not specify a language to define the coordination protocols. Such a language is specified by WS-CDL, which will be introduced next.

WS-CDL [115] *Web Services Choreography Description Language* (WS-CDL) is an XML-based language that describes the cross-enterprise collaborations of Web services from a common viewpoint. It describes the rules and agreement to which the participant services need to conform. The main feature of WS-CDL is that neither it is an executable process language (which is different from WSFL or XLANG), nor it depends on any specific process implementation language. The advantage of this feature is that WS-CDL can support interoperable collaborations between Web services regardless of their supporting platforms or programming models.

The WS-CDL describes a Web service participant in two aspects: *static coupling* and *dynamic coupling*. The static coupling specifies the static collaboration contact-points of the service participant. It contains the information about *roles* and *relationships*. Each Web service participant needs to show the observable behavior so as to join collaborations. Such behavior is specified by a role. Based on the role, a relationship describes the mutual commitment between two participants. Participants are obliged to conform to their roles and relationships with others to join collaborations. The dynamic coupling contains the information about the communication and synchronization between Web service participants in a collaboration.

The core of WS-CDL is the definition of *choreography*. A choreography constructs a collaboration of Web service participants based on their roles. A choreography contains several `react` elements. The react element identifies the actions that react to the availability of variable information and constraint conditions. A choreography also contains a recovery mechanism to support transactional interaction between Web service participants. A choreography defines a *recovery block* using a `recover` element. Once an exception occurs in a recovery block, a compensation activity would be performed to recover from exceptional conditions. In addition, a choreography can be reused to create a new choreography.

BPSS [34] The Business Process Specification Schema (BPSS) of ebXML is available in UML and XML versions. The UML version only defines a UML class diagram. It is not intended for the direct creation of a business process specification but provides a representation of all the elements and relationships required for its creation. The XML version allows the creation of XML documents representing ebXML-compliant business process specifications. ebXML provides a set of common business process specifications that are shared by multiple industries. These specifications, stored in the business library, can be used by companies to build customized business processes. Interactions between business processes are represented through choreographies. To model collaboration in which companies can engage, ebXML defines Collaboration Protocol Agreements (CPAs). A CPA is an agreement by two trading partners which specifies in advance the conditions under which the trading partners will collaborate (e.g., terms of shipment and payment).

4.4.2 Web service composition

Web service composition is a process that combines outsourced Web services to offer *value-added* services [92]. Web service composition is different from traditional application integration, where applications are tightly coupled and physically combined. Web services adopt a document-based messaging model, which supports the integration of loosely coupled applications that are across multiple organizations. WS composition can be conducted in three different fashions: process/programming, interaction, and planning.

Process-based composition Most existing Web service composition techniques require programming to some extent for constructing the orchestration model [23,14,24]. Composers first need to study the component services that are described using WSDL or some ontology languages and understand the functionalities of the services and the supported operations. A further step analysis requires to identify the way operations are interconnected, services are invoked, and messages are mapped to one another. The process-based composition scheme makes the process of composing service demanding for composers. Composers need to be domain experts who are familiar with the service description language, the service orchestration algebra, and the corresponding programming skills. Since common users cannot act as a service composer, the programming-based scheme hinders common users from composing Web services at large.

BPEL4WS is a process-based composition approach. It combines the key features of XLANG and WSFL [84]. BEPL4WS models the behavior of a business process based on the interactions with the involved business partners. It establishes grammar guidelines to describe the business process based on XML, including its control logic and message format. It uses WSDL to model the services in the process flow. It also depends on WSDL to describe the external services that are needed by the process. A major design goal of BPEL4WS is to separate the public aspects of business process behavior from the internal ones. The separation helps businesses conceal their internal decisions from their business partners. Moreover, internal changes of the process implementation no longer affects the public business protocol. Therefore, BPEL4WS has both abstract and executable business processes to support the separation. An *abstract process* also refers to the business protocol. It specifies the public aspects in the business interactions. Specially, an abstract process only deals with the exchanges of public messages between business partners. It is isolated from the execution of a process flow. Therefore, an abstract process will not release any internal details of the process to its partners. An *execution process* contains the logic and state of a process. It specifies the sequence of the Web Service interactions conducted in the business process of each business partner. A business process consists of several steps called activities. BPEL4WS supports two types of activities, including *basic activities* and *structured activities*. Basic activities manage the interactions of the process with its external resources. Its major task is to receive, reply, and invoke Web services. Structured activities control the overall process flow. They specify the sequence of Web services in the flow. BPEL4WS also defines a set of control constructs to enable the sequencing mechanism, which is similar to XLANG. BPEL4WS provides a set of mechanisms to support transactions and handle exceptions. It uses a *scope* element to aggregate several activities in a transaction. When an error occurs, a compensation procedure will be invoked, which is also similar to XLANG.

Interactive composition The interactive composition scheme blurs the distinction between composers and common users. Composers are required to have a clear goal and know the tasks that need to be performed to accomplish the composition. Common users can be guided through a set of steps to finish a composer's task. The composition scheme will work interactively with the common users to help them achieve the orchestration model. The orchestration process can start from users' goals and work backward by chaining all related services. It can also start from some initial states and achieve the users' goals by adding services in the forward direction. At each step, the scheme will choose a new service based on the task specified by the users. The interactive scheme can also capture the constraints and preferences during the interaction process. The constraints and preferences can serve as additional criteria to select services for the composition.

An interactive composition approach is proposed in [95]. It adopts the OWL ontologies to model the component services. The service model specifies the input, output, precondition, and effects (IOPE) of services. The proposed approach also implements a tool for automatic translation from WSDL to OWL-S, which enables the support of WSDL-based component services. The data types are defined by XML-schema and message exchange between component services relies on the data flow approach. The composed service are specified using OWL-S. The interactive composition can be performed by chaining component services in either the forward or the backward directions. At each step, the composition scheme adds a new service based on the users' selection. Existing component service in the orchestration model can serve as a criterion to filter candidate services. Only the services that match the IOPE properties of existing services can be selected by the system and presented to the users.

Planning-based composition The planning-based composition scheme aims to relieve users from the composition processes as much as possible. It relies on AI planning techniques for automatic service composition. In this context, users are allowed to submit a declarative query specifying the goal he/she wants the composite service to achieve together with some of the constraints and preferences that need to be satisfied. Based on the user's query, the composition scheme can derive a corresponding orchestration model with all constraints and preferences satisfied. The planning scheme regards services as actions that are applicable in states. State transitions are specified using the preconditions of some actions. A transition will lead to some new states, in which the effects of some actions are valid. Based on this, the composition scheme recursively adds new services until users' goals have been achieved. The states of existing service in the orchestration will determine the selection of the new services. For example, the preconditions of the new services should be satisfied via the effect of some existing services.

A representative planning-based composition approach is presented in [64]. It is based on situation calculus to compose Web services. More specifically, it adopts Golog, which is a logic programming language, and makes some extensions to adapt it to Web services. The situation calculus enables software agents to reason about Web services. Web services are modeled as actions, which is similar to the classical AI planning problem. Web services are associated with some preconditions and generate some effect under these preconditions. Simple Web services are categorized into two groups. The Web services in the first group perform information collection actions. The Web services in the second group perform world-altering actions. Composite Web services perform complex actions by composing simple Web services from both groups. Users can specify their requests and constraints, which can be transformed using situation calculus. Users' constraints can be used to customize the predefined generic composition templates. This helps generate the specific composition plans that fulfill users' requirements.

Web service processes involve complex interactions among different Web services across organization boundaries. Automating complex Web service processes help achieve the seamless interoperation between applications on the Internet. Several major issues need to be addressed pertaining to the WS processes. For example, WS interactions need to take place in such an environment that security is ensured and privacy is preserved. Web services are usuallyautonomous and their availability fluctuating. Failures and changes are expected to occur frequently. New services are continuously emerging on the market. The security, privacy, transaction support, change management, monitoring, and QoWS components in the proposed WSMS are required functionalities to address the above issues. The following sections give a detailed introduction of each of these components.

5 Web service security and privacy

Web services use the Internet to perform their functionalities and interact with service users. This calls for securing Web services against various attacks and protecting service users' private information. Traditional mechanisms ensure data security by fulfilling three requirements: (1) *data confidentiality*, which refers to protecting data against unauthorized disclosure, (2) *data integrity*, which refers to protecting data against unauthorized or improper modification, and (3) *data availability*, which refers to protecting and recovering from data access denials [15]. Solutions include k-anonymity, data encryption, digital signature, and various access control policies. Web service security and privacy extend these mechanisms and tailor them to the complex features of Web services.

5.1 Security

Web services are accessed by receiving SOAP messages as inputs, performing requested actions, and sending SOAP messages as outputs. Therefore, the main objective of securing Web services is to provide mechanisms for ensuring that services act only on the authorized requests and for ensuring SOAP message confidentiality and integrity. Web service security focuses on authenticating service requests and communication infrastructure. Securing XML data in Web-based environment introduces more requirements than in a traditional data security scenario. First, it is required to build up confidential systems between Web service that are autonomous and unknown to each other. As a result, Web service security heavily relies on Public Key Infrastructure (PKI). Second, it is required to provide flexible, declarative specification on user credentials and profiles. Besides user login information, other attributes (such as age, location, etc.) might be included in access control policies [59].

Traditional security techniques can be used to build up a secure transport channel for exchanging messages. Typical examples include *Secure Sockets Layer (SSL)* and *Transport Layer Security (TLS)*. SSL has been widely used in securing Web documents [71]. It works between HTTP and TCP network layers. SSL relies on public- and private-key encryption mechanism. It is also combined with digital certificates, which depend on third-party authority to authenticate users. TLS provides connection security based on encryption. The SSL and TLS enable point-to-point (server-to-server) secure sessions. However, they cannot deal with the scenario where a SOAP message is routed via more than one server. In this case, the credential of the sender needs to be delivered in the SOAP message, which would cause scalability problems.

A few industry standards are already emerging for securing Web services, including *Security Assertion Markup Language (SAML)* and Web *Services Security protocol (WSS)* [42]. SAML includes security information, such as user identity and access right, into its defined schema for document structure. It enables interactions between systems with different security architectures. SAML works with SOAP to exchange authentication, attribute, and authorization assertions for securing Web services. WSS is a newly ratified standard by Oasis [75]. It enables Web services to carry security data in the headers of SOAP messages. The WSS relies on two security technologies – XML Digital Signature and XML Encryption – to include both encrypted data and digital signatures in XML documents [107,108]. In addition, it uses security tokens, such as SAML assertion and Kerberos authentication tickets, to validate digital signatures [42].

5.2 Privacy

Web services offer a more convenient and efficient environment for Web users. Users can perform tasks, conduct business, and access high-quality information with great ease. However, to interact with Web services, Web users generally need to divulge sensitive information, such as Social Security Number (SSN), income, occupation, and home address. The user–service interaction may cause violation of Web users' privacy due to the releasing of their personal information. *Privacy*, in general, refers to "the control an individual has over information about himself or herself" [116]. Web service privacy concentrate on privacy enforcement when Web user's privacy is to be released to third parties. In this case, the user's privacy is no longer under the control of the Web service. More specifically, *Web privacy* refers to "the right of Web users to conceal their personal information and have some degree of control over the use of any personal information disclosed to others" [89].

Standardization efforts have tried to address the Web privacy protection issue. A typical example is the W3C's

Platform for Privacy Preference (P3P) [114]. P3P standardizes the way to encode privacy policies in XML for Web sites. It also specifies the mechanisms to locate and transport privacy policies. However, the major focus of P3P is to enable Web sites to convey their privacy policies [65]. It is not directly applicable to Web services. Two promising trends towards developing effective techniques for privacy preservation are based on ontologies and reputation, respectively. Ontologies can add semantics to the specification of privacy models and privacy policies [101]. Semantics in privacy models can achieve a common understanding of the sensitive degrees and the context. For example, ontology-based privacy model can answer two questions: how sensitive the information is; and under what conditions the information has that sensitive degree [101]. Defining a *privacy ontology* may allow Web services to properly perform users' tasks while preserving their privacy [89]. Kagal et al. [55] proposes to incorporate privacy policies into OWL-S descriptions and requester profiles. It defines algorithms to check policy compliance and integrates them in the service selection process of the OWL-S matchmaker. Privacy policies are specified in *Rei*, a logic-based language that allows rules and constraints to be defined over domain specific ontologies [54]. The *reputation-based* approach enables interacting Web services to have a better understanding of their mutual behavior. This information can guide Web services to preserve users' privacy when interacting with other Web services [90].

6 Quality of web services

The concept of quality of service (QoS) has been widely used in middleware and networking communities [11, 60]. Research efforts in these communities mainly focus on the performance of network and devices. There has been a surge in adapting the QoS concept to Web services in line with their fast growth. *Quality of Service* or *Quality of Web Service* (QoWS) could encompass a number of quantitative and qualitative parameters (non-functional properties) that measure the Web service performance in delivering its functionalities. We present a *taxonomy* of QoWS parameters to clearly identify the different quality aspects of Web services. The definition of this taxonomy is based on a summarization of different QoWS parameters from existing literatures [26,121,62]. Figure 7 shows the QoWS taxonomy including the different QoWS types and their relationships. The root type QoWS has two subtypes, including *runtime* quality and *business* quality. These quality parameters can be evaluated using quantitative or qualitative measurements.

Runtime quality represents the measurement of properties that are related to the execution of an operation. We identify five runtime quality classes: *response time*, *reliability*, *availability*, *accessibility*, and *integrity*.

– *Response time* Response time measures the expected delay between the moment when a service operation is initiated and the time the operation sends the results. This parameter can be quantitatively measured.
– *Reliability* Reliability measures the ability of a service operation to be executed within the maximum expected time frame. This parameter can be quantitatively measured.
– *Availability* Availability measures the probability that the service operation is operating at any given moment and is available to perform its function on behalf of its users. In another word, a high available service operation is one that will most likely be working at a given instant time. This parameter can be quantitatively measured.
– *Accessibility* Accessibility measures the degree that a service operation is capable of serving a Web service request. It may be expressed as a probability measure denoting the success rate or chance of a successful service instantiation at a point in time. There could be situations when a Web service is available but not accessible. High accessibility of Web services can be achieved by building highly scalable systems. Scalability refers to the ability to consistently serve the requests despite variations in the volume of requests. This parameter can be quantitatively measured.
– *Integrity* Integrity measures how a service operation maintains the correctness of the interaction in respect to the source. Proper execution of Web service transactions will provide the correctness of interaction. A transaction refers to a sequence of activities to be treated as a single unit of work. All the activities have to be completed to make the transaction successful. When a transaction does not complete, all the changes made are rolled back. This parameter can be qualitatively measured.

Business quality allows the assessment of a service operation from a business perspective. We identify three business attributes: *cost*, *reputation*, and *regulatory*.

– *Cost* Cost measures the units of money that a service requestor needs to pay to invoke service

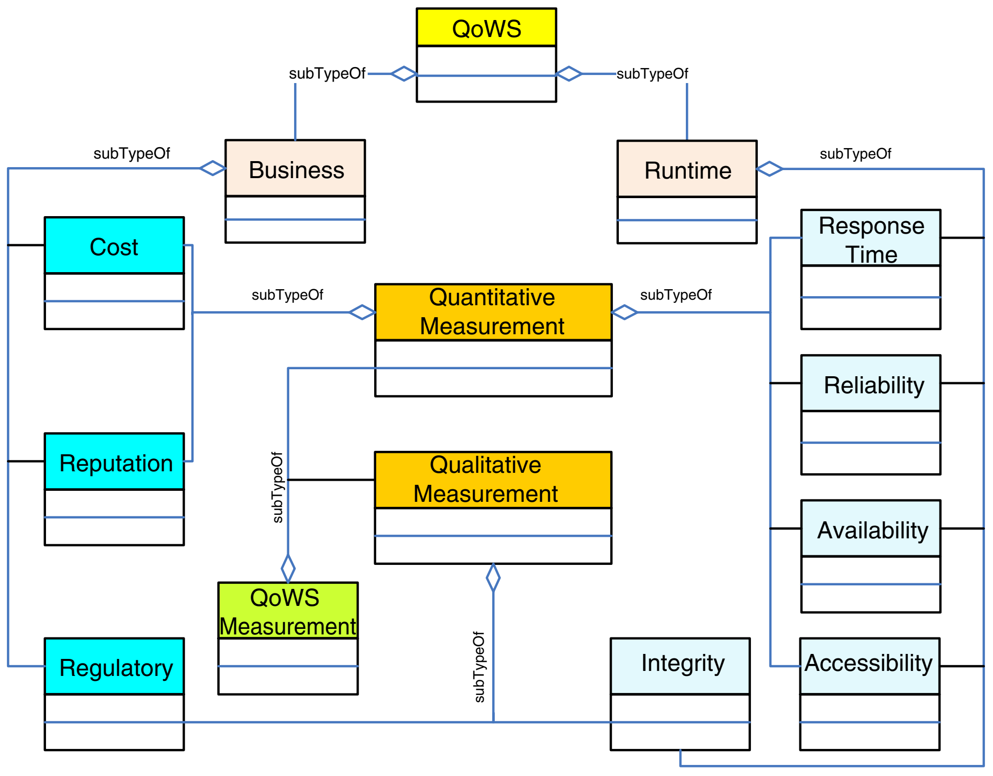

Fig. 7 A taxonomy of QoWS parameters

operation. This parameter can be quantitatively measured.

– *Reputation* Reputation measures the trustworthiness of a service operation based on user feedbacks. Users are prompted to rate service operations on a scale after using them. The reputation corresponds to the average of collected ratings. This parameter can be quantitatively measured.
– *Regulatory* Regulatory measures whether a Web service in conformance with the rules, the law, compliance with standards, and the established service level agreement. Web services use a lot of standards such as SOAP, UDDI, and WSDL. Strict adherence to correct versions of standards (for example, SOAP version 1.2) by service providers is necessary for proper invocation of Web services by service requestors. This parameter can be qualitatively measured.

The QoWS taxonomy is not meant to be exhaustive by addressing all aspects of QoWS. Other quality parameters can be included to fulfill the requirements of different application domains. The taxonomy generalizes a set of important QoWS parameters, which have been emergent in current literatures. The definitions convey the meanings of the each quality parameter in the context of Web services. They also give a hint of how these quality parameters can be quantitatively or qualitatively measured. A typical WS deployment system would address only some quality aspects defined in this QoWS taxonomy. The business requirements and usage scenarios decide which quality aspects a system needs to address.

7 Web service management

The WS management offers a set of management capabilities to *monitor* and *control* service qualities and service usage. Web service management mechanisms are highly coupled with QoWS of a Web service. *Monitoring management* aims to rate the behavior of Web services in delivering their functionalities in terms of each QoWS parameter. Monitoring Web service behavior would be crucial in either assessing QoWS parameter values or ensuring a Web service to maintain its promised QoWS. *Control management* aims to improve the service quality through a set of control mechanisms. Typical control mechanisms include Web service transaction and coordination, Web service change management, and Web service optimization. Transactions help improve the reliability and fault-tolerance of Web services. Change management deals with highly dynamic environment of Web services. It takes a series of actions to identify the changes, notify the coupled entities, and adopt appropriate operations to respond to the changes.

Web service optimization helps users identify Web services and/or their combinations to best fulfill their requirements.

Control management and monitoring management might sometimes work cooperatively. For instance, the Web service optimization would need the monitoring process to get the QoWS parameter values of different Web services and/or their combinations. These values will guide the optimization process to return the optimized solutions that best fulfill users' requirements. Similarly, change management also needs the monitoring process to report changes so that it can make the corresponding reactions.

7.1 Monitoring web services

Web services are dynamic and autonomous entities. They rely on communication protocols (e.g., HTTP, SMTP) and a messaging model to offer miscellaneous functionalities on the Web. However, the underlying communication protocols could be unreliable and incapable of meeting the function delivery requirement. For example, HTTP is a stateless protocol that does not guarantee the order of the arriving packets and the successful delivery of these packets to the destination. In addition, the messaging model requires parsing intensive XML data with a lot of type information, which is rather time consuming. These could both introduce failures for Web services in the delivery of their functionalities. Complex WS processes involve multiple Web services and require more complex interactions, where failures could occur frequently. Monitoring offers a "fault detection" mechanism that checks the health of a service in real time and tries to reduce application downtimes by detecting signs of failure. It ensures that the service is available, accessible, and capable of meeting the throughput and latency requirements. In addition, as more Web services are expected to emerge on the Web, several Web services may compete in their offerings for a given functionality. The key difference would be on how these functionalities are to be delivered in terms of QoWS. However, getting the right value for a given QoWS parameter is neither an easy nor a trivial task. In that context, monitoring Web services behavior would be crucial in either calculating QoWS parameters values or assessing a Web service's claim in term of promised QoWS.

Monitoring may take different forms, depending on the QoWS parameter. These include *message interception, probing*, *value collection*, and *user feedback*. To allow probing, Web services need to support some "free" operations that can be invoked without any effect for the invoker. Ideally, Web services should support a "ping" operation. To avoid a high overhead on the system, monitoring could be conducted periodically. Some QoWS parameters are not measurable through the Web service computational behavior and need the feedback from users. For example, *reputation* may be obtained from third party based on the Web service business behavior. Table 1 summarizes how monitoring is conducted for each *QoWS* parameter.

A frequent monitoring strategy helps precisely reflect the execution performance and behavior of a Web service. However, this may put a significant burden on the resources. A fair approach would be to have a configurable monitoring approach. In this case, the different parameters are adjusted depending on the requirements, in terms of rating precision, of the application being used.

7.2 Transactional support

Cross-enterprise business processes involve autonomous Web services from different business entities. Transaction support is required to provide reliable and dependable execution of WS processes. Transaction support is highly coupled with the process level. However, a major difference between transaction support and other process-level technologies (e.g., coordination and composition) is that it is not used to construct a WS process. Instead, it improves the quality of a WS process in terms of reliability and fault-tolerance, etc. In traditional transactions, the transaction monitors are in charge of all the resources that are involved in the transactions. They adopt a 2PC (i.e., two phase commit) transaction model to maintain Atomicity, Consistency, Durability, and Isolation (ACID) of transactions. Since the 2PC requires locking resources for the duration of the entire transaction, typical ACID transactions are tightly coupled and span over short periods of time. In contrast, interactions between Web services take place in a loosely coupled manner. No participating Web service is in full control of others. Business transactions over Web services may be long-lived, which include multiple complex tasks, such as negotiation, commitment, billing, shipping, and tracking. Resource locking strategies (e.g., 2PC) could hinder Web services and users from joining other business processes, hence contradicting the business requirements.

The atomicity of Web services and the long-lived business processes require transactional support for Web services that is less strict than the traditional ACID properties. A current proposal that provides such support is WS-Transaction. WS-Transaction extends WS-Coordination by defining a set of protocols for transaction processing. It relies on the WS-Coordination framework to

Table 1 WS monitoring

Quality type	Parameter	Monitoring type	Monitored information
Runtime	Response time	Messages interception	Average of actual response time
	Reliability	Probing through pinging	Ratio of successful pings over a period of time
	Availability	Probing through pinging	Ratio of successful pings over total number of pings
	Accessibility	Messages interception	Ratio of successful invocations over total number of invocations
Business	Cost	Values collection	Difference between advertised and requested fees
	Reputation	User feedback	Average rating from a group of users
	Regulatory	Not applicable	Not applicable

manage the transactions across multiple Web services based on the transaction protocols. Specifically, a centralized coordinator or a set of distributed coordinators coordinates Web services that participate in a transaction. WS-Transaction defines two coordination types: atomic transaction and business activity. The former supports short-lived transactions involving participants that reside in a trust domain. It adopts the 2PC protocol to coordinate participating Web services, which is similar to traditional transactions. The latter supports long-lived transactions involving loosely coupled participants. It adopts compensations to deal with transaction failures. If one of the transaction steps fails, the completed steps are compensated to ensure consistency. In contrast to the immediate consistency achieved by 2PC, compensations allow intermediate inconsistency of transactions before all compensation steps are finished [81].

The challenge for automating WS transactions is that they have no precisely defined transaction concepts such as commit, abort, resource, and lock [9]. In traditional transactions, these concepts are defined upon a fixed set of operations, which consist of creation, inserting, update, and deletion. However, WS representation (e.g., WSDL) could encode various operations. For different operations, the transaction concepts can be interpreted in different ways. Therefore, it is difficult to define a uniform set of transaction concepts that can be applied to all WS operations. For example, when a WS transaction is aborted, the completed WS operations need to be compensated. However, the compensation strategies would vary based on the functionalities of the operations, the business requirements of the service providers, and other context (e.g., users' profile). This would result in different abort results. In our travel scenario, assume that Joan's travel plan has to be canceled after the air ticket has been purchased. Based on the functionality of ticket purchase, the corresponding compensation is to return the ticket and get the money back to Joan's credit card. However, `AirCompany` requires that customers should notify them 3 days prior to the departure date to get a full refund. Otherwise, a late fee will be charged based on the notification date. In addition, for frequent fliers of `AirCompany`, they only need to send their cancellation notification 2 days prior to the departure date to get the full refund. As we can see, compensation of a simple business operation could be associated with complex conditions. The abort result will be different under different conditions. Therefore, it is important to extend the description of services by *unambiguously* describing transactional semantics of Web service operations. This would be the prime step for automating transaction processing of Web services.

7.3 Change management

The Web service life cycle is characterized by its highly dynamic features: (1) New services may come into play without notification of existing services. For instance, A new hotel service might begin its business and publish the service description on the UDDI repository without releasing any information to `TravelAgency` in advance; (2) operating services might stop functioning without any indication. For instance, `Hotel` might only operate in summers but close in winters because of the little volume of clients; (3) functionalities of services might change overtime. For instance, `AirCompany` could suspend the scheduled flights to a health resort due to the propagation of an infectious disease in its area.

One of the fundamental challenges in enabling WS processes is to manage changes in their lifecycle. We expect that WS processes would only thrive if they are able to quickly and automatically adapt to changes. Changes could happen in any stage of the WS lifecycle, including development, deployment, and maintenance. We discuss change management as part of the maintenance stage of the WS lifecycle. Current methods of change management are usually *ad hoc* and require significant human involvement (e.g., business process re-engineering, continuous process improvement). *Workflows* have traditionally been used to model and enact organizational business logic. In this approach, tasks are mapped to business units for enactment. These units are usually part of one single enterprise. The enterprise's internal structure is fairly static and the rate of organizational change is usually very slow. Therefore, changes in workflows have usually been modeled as *exceptions*. In contrast, the WS processes consider changes as the rule, and any solution to change management would need to treat them as such.

A prerequisite for change management is to derive a uniform specification of changes that may occur in a WS process. There are two approaches to specifying changes: *top-down* and *bottom-up*. A top-down approach focuses on changes that are usually voluntary and business mandated [7]. For example, the WS processes may add a new service to its composition to take advantage of a business opportunity. Hence, top-down changes are motivated by the WS process's business goal. Unlike top-down changes, bottom-up changes are initiated by the member services [8]. For instance, a member service operation may become unavailable during execution and as a result, would trigger the WS process to replace the service. Hence, bottom-up changes are mandated by the member services, and are usually more disruptive than top-down changes.

Changes have been modeled using various tools and technologies [35]. For example, the *Z* language has been used to reflect changes in database schema [36]. However, changes to Web services are fundamentally different from changes in databases. Web services consist of behavioral aspects that are not relevant to data. In [79], the author defines a strategy for extending the service oriented architecture to enable change management. This work proposes the management of functional and non-functional changes to Web services. However, the research is at an early stage, and it does not yet provide a core set of specification for modeling and managing changes. In [76], Web service attributes are defined in a taxonomy of non-functional properties and represented by a series of Object-Role Models (ORM). An example is the attribute of service availability. The authors have defined service availability and its temporal characteristic from two perspectives: service provisioning and service request. Finally, workflows provide the ability to execute business processes that span multiple organizations [99]. Traditional workflows provide little support for dynamic change management. Workflows are geared towards *static* integration of components. Furthermore, workflows do not cater for the behavioral aspects of Web services. For example, they do not distinguish between the internal and external processes of a Web service [21].

The use of visual modeling techniques, such as Petri nets in the design of complex service oriented enterprises, seems justified because of several reasons [49]. Petri nets provide attractive properties for modeling, including formal semantics, graphical representation, powerful expressiveness, and abundant analysis methods [45]. In that respect, the Petri nets are based on well-grounded mathematical foundation. They can represent system models graphically as net diagrams. Therefore, they provide an easy communication between different parties. Petri nets have the ability to also deal with asynchronous concurrent systems. Finally, they provide multiple analysis methods, such as coverability tree, incidence matrix, and state equation. These methods help identify system properties readily. Because of the complexity of changes in WS processes, formal modeling is a useful method to improve the understanding of the problem and the solution. An essential step in change modeling is to validate the model. Petri net theory provides algorithms and methods, which can be applied directly to the model and its analysis to validate the model. Other modeling and specification techniques have been proposed, but lack easy formalization features required by change management in WS processes [35,36].

In [6], a taxonomy of changes in service oriented enterprises has been identified using a bottom-up approach. It first describes triggering changes that may occur in Web services. These changes are then mapped to reactive changes in Web service processes. The specification of changes is a prerequisite to successfully managing changes in a Web service process. It uses Petri nets as a change specification tool because they provide a visual representation of changes. Furthermore, a Petri net model is used readily for verifying changes in Web service processes.

7.4 Optimization

The proliferation of Web services will introduce competition between large numbers of Web services that offer similar functionalities. Selecting the best service from a large space of options would be a big challenge to service consumers. Some services might offer certain benefit in several aspects, less attractive in other aspects. For example, in our travel scenario, the client can choose a hotel that provides the most convenience. The hotel could be close to many local attractions. However, the rate of this hotel could be very expensive, which exceeds the client's budget. In this case, the client may have to look for other hotels, which might be far away from the local attractions but with a lower rate. In this case, the client might need to rent a car or take taxis. In this case, she would have to choose among competing taxi or car rental services. Typically, the available options for each of these services would be many more than what we mentioned earlier.

Web service optimization offers strategies for finding the “*best*” Web services or their *composition* with respect to the expected user-supplied *quality*. Due to the large space of competing Web services, a service request could be potentially resolved by multiple services. Thus, it is necessary for WS optimization to set appropriate criteria to select the “best” among possible choices. Recent research literature shows that QoWS of individual Web services is crucial for their competitiveness [26]. The challenge for WS optimization is to define appropriate metrics to characterize *QoWS* and devise techniques to use it in optimizing service-based queries. However, only selecting individual services is still not enough since some services may be related to each other. Services need to be composed and considered *collectively*. Therefore, how properties of multiple services can be combined will be an important issue. This usually depends on how the composed process uses these constituent services (e.g., sequential, parallel, etc). In [121], a quality model is presented to integrate properties from multiple Web services. The quality model can then be used to evaluate the quality of the composed Web service processes.

Optimization techniques have been widely adopted in different types of DBMS (e.g., central, distributed, and multidatabases) for optimizing data queries. The ultimate goal of any database system is to allow efficient querying. A fundamental difference between database query optimization and WS optimization lies in the manipulated objects. The first class objects in WS optimization are services or their compositions while data is the first class object in databases. In WS optimization, optimization focuses on QoWS parameters related to the behavior of the Web services while in most existing techniques, optimization concerns only the response time of the query execution plan. QoWS optimization can be achieved through a three-phase process: candidate service selection, negotiation, and enactment monitoring in [18].

8 Evaluation of Web service deployment systems

We present a representative set of WS deployment systems in this section. These systems adopt different technologies to provide the functionalities identified by the WSMS framework. These include interoperation, security/privacy, QoWS, and management. The evaluation only covers the most representative systems for the sake of space.

There are some other standardization efforts underway to fully support for semantic Web services. For example, the Semantic Web Services Initiative Architecture (SWSA) committee focuses on providing architecture support for deploying semantic Web services [18]. It has identified a set of requirements for building a semantic Web service architecture. The architecture supports a three-phase interaction with semantic Web services: discovery, engagement, and enactment. Architectural requirements are identified for each interaction phase. Instead of building concrete software components, SWSA aims to generalize the protocols and functional descriptions of capacities across a variety of semantic Web service architectures. This enables specific components that are consistent with the proposed general model to interoperate with one another.

8.1 Research prototypes

In this section, we present a set of WS deployment systems: CMI, Meteor, SELF-SERV, WebDG, AgFlow, WSXM, and IRS.

8.1.1 Collaboration Management Infrastructure

Collaboration Management Infrastructure (CMI) [12, 43] provides architectural support to manage collaboration processes. The kernel component of CMM is a *Core Model* (CORE). The CORE provides a common set of primitives shared by all of its extensions. These primitives fall into two categories – *activity states* and *resources*. Activity states can be either *generic* or *application-specific*. Generic activity states follow the convention of the Workflow Management Coalition [117]. They capture activity behaviors that are independent of applications. Application-specific activity states enable the precise modeling of the peculiar applications. The CORE identifies several primary resource types for the activity execution. For example, data resources refer to the workflow internal data. Helper resources refer to auxiliary programs that help implement the basic activities.

The CORE extensions include the *Coordination Model* (CM), the *Service Model* (SM), and the *Awareness Model* (AM). The CM coordinates participants and automates process enactment. The CM provides two types of advanced primitives: *activity placeholders* and *repeated optional dependencies*. Activity placeholders enable the run-time service selection. They represent services in a process as activity types at specification time. At run time, a concrete activity replaces the activity placeholder to construct an executable process. A resolution policy helps ensure the syntactic and semantic compatibility when replacing the placeholder by an actual activity. Repeated optional dependencies specify the invocation place of an activity in a control flow path of a process. They also specify the number of invocations of the given activity to accomplish the application's objective. The SM provides rich semantics to describe services. It introduces the notion of service ontologies to capture the semantics of services. In addition, the SM uses Quality of Service (QoS) as an important non-functional parameter to characterize services and their providers. The resolution policy can choose the service with the best QoS to optimize the execution of a process. The Awareness Model monitors the process-related events. It allows authorized composition and delivery of such events only to closely related process participants.

8.1.2 METEOR

METEOR [23] is a workflow management platform that supports QoS management and service composition. It sets up a QoS model to describe the non-functional issues of the workflow. The QoS model consists of four parameters, including time, cost, reliability, and fidelity. For each parameter, the description of the operational runtime behavior of a task is composed of two classes of information: basic and distributional. The basic class specifies the minimum value, average value, and maximum value the parameter can take in a given task. The distributional class specifies a constant or of a distribution function (such as Exponential, Normal, and Uniform) which statistically describes a task's behavior at runtime. The values in the basic class are used by mathematical methods to calculate workflow QoS metrics. The distributional class information is used by simulation systems to compute workflow QoS. METEOR forms a mathematical model that collectively uses all QoS parameters. The mathematical model computes the overall QoS of workflow. It can serve as a guide to predict, estimate, and analyze the QoS of production of workflow.

METEOR extends DAML-S to describe Web services. The enriched service description includes three parts of information: syntactic description, semantic description, and operational metrics, such as QoS parameters. All of this information help match the service object with a corresponding service template. METEOR also provides registry services to enable the advertisement and discovery of Web services. The workflow management system executes the composite services and also handles runtime exceptions using a case-based reasoning mechanism.

8.1.3 SELF-SERV

SELF-SERV is a platform for Web services composition [14]. It aims to provide a declarative mechanism to compose Web services. It defines a peer-to-peer mode to support scalable execution of composite Web services. SELF-SERV uses a *state chart* to model the flow of component service operations. It integrates *service containers* into its service composition platform. A service container consists of a set of Web services with common functionalities. The container determines which service is selected to execute. It carries out the selection dynamically and puts off the selection until the invocation time. The service selection is based on a set of QoWS metrics and their relative weights. The service container also handles change management. It provides operations to monitor services, notify the changes, and make reactions to the changes. SELF-SERF relies on a set of state coordinators to enable the scalable execution of composite services. It generates a state coordinator for each state in the state chart. The state coordinator determines when to enter its associated state depending on the notification of other coordinators. Once a state has been entered, the coordinator invokes the services and retrieves the results. The coordinator then notifies other coordinators when it completes the execution of the service. Routing tables maintain the information required by a coordinator. The information may include preconditions for entering a state and postactions that notify successive coordinators.

8.1.4 WebDG

Digital government has turned to be a major application area of Web services. WebDG is proposed based on the available Web service technologies [68]. It aims to provide high quality e-government services by improving the interacting mechanism between government and citizens. Two major contributions make WebDG distinguish itself from other e-government service suppliers. The first one is that it introduces the privacy-preserving scheme into Web services. This is among the leading efforts to combine privacy protection with Web services. While the second contribution is that it realizes the automatic service composition based on the semantic feature of Web services.

WebDG enforces privacy from the technology point of view but not merely depends on the trust of involved entities. It constructs a three-layered privacy model, including *user privacy*, *service privacy*, and *data privacy*. Each layer defines its own privacy policy respectively. Obeying these privacy policies, WebDG implements two privacy preserving schemes, including *DFilter* and *PPM*. These two schemes guarantee that necessary credentials are the keys to access the requested operation. Service composition is a necessity for most Web service systems because a single service could hardly fulfill all requirements from a user. WebDG achieves its automatic service composition resorting to the ontology notion. According to the semantic features of the services, WebDG defines two ontologies for service operations, namely, the *Category* and *Type*. The semantic composability rule is derived based on these ontologies. It states that two operations can be composed with each other semantically if their Categories and Types are compatible, respectively. WebDG also implements a composition template to evaluate the soundness of the service composition.

8.1.5 AgFlow

AgFlow is a QoS-aware middleware for Web service composition [121]. It uses ontologies to model the component services. The data types follow the XML specification and the message exchange relies on the data flow approach. The orchestration model is specified using statecharts and generated by the programming-based composition scheme. AgFlow defines a QoS model to evaluate Web services from five quality aspects: price, duration, reputation, success execution rate, and availability. Users can specify their preferences by assigning weights to each of these quality parameters. AgFlow proposes two planning strategies, local and global, to select the proper component services. The candidate composition plans are evaluated against an objective function, whereby the optimal plan with the highest objective value can be selected. Users' constraints are also considered during the planning. The global strategy can adapt to the dynamic changes in the service environment. When some component service becomes unavailable or significant changes occurs to its QoS, a re-planning process will be triggered. The re-planning is to enable the composite service to remain optimal in a dynamic environment. The performance of AgFlow is efficient when there is a small number of tasks to be accomplished by the composition. However, as the number of tasks increases, the response time of AgFlow exhibits an exponential growth. This situation becomes even more severe when re-planning is required by the composition.

8.1.6 WSMX and IRS-III

Web Service Modeling eXecution environment (WSMX) [105] is the reference implementation of WSMO. It provides an event-based service oriented framework for dynamic service discovery, selection, mediation, and invocation. The core of WSMX architecture is the WSMX manager. It has several major components, including *resource manager*, *discovery*, *selector*, and *mediator*. The resource manager is used to manage the repositories that store WSMO entities and other system-specific information. WSMO entities include Web services, goals, ontologies, and mediators. WSMX discovery provides a three-step solution for service location. First, *goal discovery* is to map a user request to a predefined, formalized goal in the goal repository. Second, *Web service discovery* is to map the formalized goal to the formalized service description in the service repository. During this process, a Web service with the capability that matches the goal would be returned. Finally, *service discovery* is to map the formalized service description to the concrete service. The WSMX selector helps choose the "best" Web service returned from a set of matching Web services. Various selection criteria as well as users' preference can be applied to select the optimal Web service. The WSMX mediator is used to mediate heterogeneous entities. WSMX provides two kinds of mediators: *data mediators* and *Process mediator*. Data mediators are used to address semantic dissimilarity between data from different sources. Process mediators provide a runtime analysis and adjustment of mismatching between communication patterns from service requesters and providers. The WSMX manager controls operational flow to react to incoming requests. WSMX provides an interface to accept user requests. Once a service request is submitted, the WSMX discovery and WSMX selector locate the services that match the request and return the optimal one on demand. Internet Reasoning Service (IRS-III) is another reference implementation of WSMO [32]. It is a framework for description, location, composition of Web services. IRS-III provides two methods for creating semantic Web services, including browser-based and Java API. The IRS-III ontology adopts and extends WSMO. The additional attributes include a new type of mediator, gw-mediator. The gw-mediator is used for service discovery. An applicability function is used for selecting Web services.

8.2 Discussion of Web service deployment platforms

In this section, we evaluate and compare different Web service deployment systems using the proposed WSMS. We first conduct the evaluation by mapping these systems onto the WS interoperation framework. This helps reflect how each system achieves the WS interoperability. We examine the different layers in the interoperation framework, including communication, messaging, representation, discovery, and processes. We then evaluate the deployment systems based on other key components in the WSMS: security/privacy, management, and QoWS. This helps reflect the functionalities of each system for dealing with other issues of WS deployment in addition to WS interoperability.

Table 2 compares the representative Web service deployment systems using layers in the WS interoperation framework. For instance, WebDG uses HTTP at the communication layer. It depends on the three key Web service standards – SOAP, WSDL, and UDDI – for messaging, representation, and discovery. In addition, the WebDG also uses ontologies to describe the semantic features of Web services. At the WS processes layer, the WebDG provides a set of mechanisms to compose Web services automatically. It uses the composition rules to check the semantic and syntactic composability of Web services. The composition plan is optimized based on the QoC (Quality of Composition) model. A composition template is used to evaluate the soundness of the composition.

Table 3 compares the same set of systems using the other key components in the proposed WSMS. For instance, RosettaNet adopts digital certification and digital signatures to ensure security to interact with Web services. The PIP of RosettaNet contains a transaction layer to provide transaction support for the business processes. Privacy, change management, optimization, monitoring, and QoWS are not specified in RosettaNet.

9 Discussion and open issues

Current technologies help establish a solid foundation to enable the functionalities of Web services. Communication protocols, XML-based standards, and management facilities provide a good support for interoperation among Web services. The expected large number of Web service to be deployed on the Web would trigger a significant paradigm shift from the *data-centric* Web to the *service-centric* Web. We identify several research trends for the future evolution of Web service technologies. These include *efficient access of Web service, automatic service composition, ontology management for Web services, failure recovery, Web service mining* and *M-services*.

Efficient access to Web services As the Web is moving from a *data Web* to a *service Web*, it is expected that tomorrow's Web will be the repository of a large number of Web services provided by third-party providers. In that context, the ability to *efficiently* access Web services is poised to become of prime importance [77]. In the simplest scenarios, accessing Web services would consist of invoking their operations by sending and receiving messages. However, for complex applications (e.g., a travel package), there would be a need for an *integrated* and *efficient* approach to *select* and deliver several Web services' functionalities. As Web services with *similar* functionality are expected to be provided by competing providers, a major challenge is devising *optimization* strategies for finding the "*best*" Web services or their *composition* with respect to the expected user-supplied *quality*. A *query infrastructure* that offers complex query optimization facilities over Web services can address the above issues. Querying Web services could leverage the research result from database queries. A first step in enabling such a service query infrastructure is defining the service scheme. The schema can be used to abstract the common features of all the Web services across an application domain and will provide a fixed vocabulary for the service query languages. In addition, the schema should also depict the relationship between different service operations. This relationship will be referred when a service query needs to retrieve multiple service operations. For example, the output of some service operations might serve as the input of another service operation. Therefore, the latter service operation would depend on the execution of the former service operation. Thus, when the latter service operation is retrieved by a service query, the first one should also be retrieved because of this dependency relationship. Service query languages (e.g., calculus and algebra) can be further defined based on the service schema.

Automatic service composition Web services are dynamic and autonomous, which pose great challenges to service composition technologies. Since Web services are unknown *a priori*, interactions between component services cannot be established at service definition time. Current composition technologies requires low-level programming and domain knowledge of the required services. Users needs to submit their requirements to service composers. Service composers are in charge of statically defining the schemas for the composite services, which end users can use to invoke the composite services. This process hinders common users from composing Web services at large. Automatic service composition blurs the distinction between composers and common users. Users can submit a declarative query specifying the business goal together with some the constraints and preferences. The composition process can automatically identify related services and derive a corresponding orchestration model with all constraints and preferences satisfied. Seamless automatic composition is rather challenging. It requires to clearly identify user's requirements (e.g., business goals, constraints, and preferences), unambiguously describe the services, precisely and formally specify the orchestration model, and adopt intelligent composition schemes. In addition, there are still some other important issues, such as security, privacy,

Table 2 WS deployment systems versus layers in the interoperation framework

Systems	Communication	Messaging	Representation	Discovery	Processes
METEOR	Java RMI	Not specified	DAML-S, QoS model	registry service	Workflow engine
CMI	HTTP, CORBA	Not specified	Service model, ontologies	Service broker, advertisement	State machine-based model
SELF-SERV	HTTP	SOAP	WSDL	UDDI	State charts
WebDG	HTTP	SOAP	WSDL, ontologies	UDDI	Composition rules, QoC Model, and composition template
AgFlow	HTTP	SOAP	WSDL, ontologies	UDDI	Statecharts
WSMX	Not specified	SOAP	WSMO, WSML	Ontology repository, WSMX repository	Event manager
IRS-III	HTTP	SOAP	WSMO, IRS-III ontology	PSM	Task specification

Table 3 Deployment systems versus key components in the WSMS

Systems	Security/privacy		Management				QoWS
	Security	Privacy	Transaction	Change management	Optimization	Monitoring	
METEOR	Not specified	Not specified	Not specified	Not specified	Mathematical model to compute the overall QoS of workflow	Not specified	Time, cost, reliability, and fidelity
CMI	Role-based access control	Not specified	Coordination model	Scoped roles	Resolution policy for QoS based service selection	Awareness Model	Services attached with a set of QoS attributes
SELF-SERV	Not specified	Not specified	Not specified	Service containers for monitoring changes	Not specified	Service containers for notifying changes and make reactions.	Weighted QoWS parameters
WebDG	Not specified	Three-layered privacy model	Not specified	Not specified	Quality of Composition model	Not specified	Not specified
AgFlow	Not specified	Not specified	Not specified	Re-planning	Integer programming	Not specified	Price, duration, reputation, success execution rate, and availability
WSMX	Not specified	Not specified	Not specified	Not specified	WSMX selector	Not specified	Accuracy, cost, network-related time, reliability, robustness, scalability, security, trust
IRS-III	Not specified	Not specified	Not specified	Not specified	Applicability function	Not specified	Accuracy, cost, network-related time, reliability, robustness, scalability, security, trust

and legal issues, to name a few. Efforts from the AI planning community could help improve the automaticity of service composition. However, how to map Web services to the classical representation of AI planning involving initial states and goals remains to be a challenging issue. Explicit goals are usually not available from an industry perspective.

Ontology management for Web services Ontologies empower Web services with rich semantics. They help enrich the service description and ease the service advertisement and discovery process. Ontologies offer an effective organization mechanism to deal with the large number, dynamics, and heterogeneity of Web services [66]. Service ontologies are organized in a distributed manner to adapt to the large scale of the Web. Service ontologies are formed based on Web services' domains of interest. Typically, a service ontology contains a set of standard terms to describe service classes. It also contains some inference rules to express complex relations between service classes. One important challenge of using distributed service ontologies is *ontology mappings*. Cross-ontology interactions between Web services may bring terms and rules from one ontology to another. Interpreting and reasoning information from other ontologies precisely and efficiently is crucial for cross-ontology service integrations. Existing efforts in data integration mainly rely on a centralized mediation mechanism [41]. However, the centralized approach can hardly fit into the large scale of service ontologies on the Web. Since ontologies have become a key component for Web service description and organization, an effective mechanism to realize mappings of different ontologies would become increasingly important.

Failure recovery A viable and robust Web service solution needs to have the capacity to deal with failures. Failure recovery is a crucial issue for proper and effective delivery of Web service functionalities. In traditional database and distributed computing systems, failures are treated as *exceptions*. Since failures rarely happen in such fixed and well-controlled environment, some expensive mechanisms are often adopted for recovery from failures. For instance, transactions with ACID properties are the major tools to deal with failures in traditional database systems. Mobile computing is another major application area for failure recovery techniques [61,72,87]. In a mobile environment, failures are more prone to happen due to multiple reasons, such as physical damage, lost of mobile hosts, power limitation, and connectivity problems. The mobile computing community would treat failures as *rules* rather than exceptions due to their high occurrence probability. Checkpoint-based recovery is a representative technique used in mobile environment. Web services are autonomous and loosely coupled. They interact dynamically without *a-priori* knowledge of each other. Therefore, failures in Web service environment are expected to happen frequently. Design of an effective failure recovery mechanism for Web services can be based on the ideas from both database systems and mobile computing. A key step towards such a mechanism is to define what failures are in Web service interaction environment and provide a clear taxonomy for all these failures.

Web service mining The envisioned *service Web* hosts a large population of Web services. Many of these coexisting Web services may be complimentary with each other in terms of functionalities. Discovering the hidden complimentary relationships and combining these Web services may be an effective means to provide value-added services for users. *Web service mining* aims to detect interesting and useful patterns from Web service compositions. It provides a more intelligent and transparent way to deliver high-quality Web services. Web service mining offers two advantages over service composition. First, since it is not restricted by composition conditions, service mining has more flexibility than service composition. Second, the discovery process is capable of finding unexpected service combinations. These combinations are not achievable through traditional composition techniques because the combined Web services may not semantically compatible with each other in terms of functionalities. In addition, the combinations would be very useful because they are summarized from various previous experiences through a training process. Traditional data mining technology is an automated process of extracting structured knowledge from large volume of data [37]. It has achieved great success in various application areas, including medical diagnosis, financial analysis, and stock price prediction. However, extending data mining techniques towards Web service mining needs to deal with several important issues. Web services are more complex objects than data. A service mining technology needs to be equipped with an effective service processing mechanism. In contrast to data, Web services are dynamic and may change over time. Therefore, adaptability to dynamics would greatly affect the service mining performance.

M-services The significant advances in wireless technologies open a promising application area for Web services. As a key extension of Web services, *mobile services* (*m-services*) cater for the increasing population of mobile users. Specifically, *m-services* are a special set of Web services that can be accessible by mobile hosts over wireless networks [86,120,119]. They work together with mobile devices to offer *anytime/anywhere* accessible services. In contrast to Web services with wired infrastructures, m-services are more suitable for time and location critical tasks [103]. For instance, a stock quote service can help users make quick response by providing timely quote prices. However, users may prefer desktops to cell phones or personal digital assistants (PDAs) when carefully preparing a travel package for vacations. The mobile environment poses great challenges for providing and consuming m-services. Mobile devices have low CPU and memory capacities, limited power supply, small screen size, and restricted input mechanisms. Wireless networks are limited by their small bandwidth. They also suffer from link outages, which result in temporary unavailability. These limitations hinder existing Web service technologies from directly working with m-services. For instance, the limited bandwidth may not be enough to convey SOAP messages [86]. In addition, SOAP is expensive for mobile hosts in terms of both power consumption and waiting time. In mobile environment, users' context, such as location and activity, may change rapidly. M-services need to track these changes and provide *context-aware* functionalities. However, service description techniques, such as WSDL, have not provided support to model context. UDDI enables service discovery in the Web service framework. However, the multiple costly round trips required by UDDI lookup are troublesome for m-service discovery [86]. The frequent unavailability of wireless network may cause failures in service discovery processes.

Acknowledgements The authors would like to thank the anonymous reviewers for their valuable comments on earlier drafts of this paper.

References

1. American National Standards Institute: Study Group on Data Base Management Systems. Interim report, FDT, 7:2, ACM (1975)
2. Report of the CODASYL Data Base Task Group: ACM (1971)
3. RosettaNet. http://www.rosettanet.org (2003)
4. Web Services Metadata Exchange (WS-MetadataExchange): http://xml.coverpages.org/WS-MetadataExchange 200409.pdf (2004)
5. Ackerman, M.S.: Privacy in E-commerce: examining user scenarios and privacy preferences. In: Proceedings of the ACM Conference on Electronic Commerce (1999)
6. Akram, M.S.: Managing changes to service oriented enterprises. Master's Thesis, Virginia Tech (2005)
7. Akram, M.S., Bouguettaya, A.: Managing changes to virtual enterprises on the semantic Web. In: 5th International Conference on Web Information Systems Engineering, pp. 472–478. Brisbane, Australia (2004)
8. Akram, M.S., Medjahed, B., Bouguettaya, A.: Supporting dynamic changes in Web service environments. In: First International Conference on Service Oriented Computing, pp. 319–334. Trento, Italy (2003)
9. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services: Concepts, Architecture, and Applications. Springer, Berlin Heidelberg New York (ISBN: 3540440089) (2003)
10. Astrahan, M., Blasgen, M., Chamberlin, D., Eswaran, K., J. Gray, Griffiths, P., King, W., Lorie, R., McJones, P., Mehl, J., Putzolu, G., Traiger, I., Wade, B., Watson, V.: System R: relational approach to database management. ACM Trans. Database Syst. **1**(2), 97–137 (1976)
11. Aurrecoechea, C., Campbell, A., Hauw, L.: A Survey of QoS Architectures. ACM/Springer Verlag Multimedia Syst. J. **6**(3), 138–151 (1998)
12. Baker, D., Georgakopoulos, D., Schuster, H., Cassandra, A.R., Cichocki, A.: Providing customized process and situation awareness in the collaboration management infrastructure. In: Proceedings of the 4th IFCIS International Conference on Cooperative Information Systems, Edinburgh, Scotland, 2-4, September 1999, pp. 79–91. IEEE Computer Society (1999)
13. Benatallah, B., Casati, F., Grigori, D., Nezhad, H.R.M., Toumani, F.: Developing adapters for Web services integration. In: CAiSE Conference, pp. 415–429. Porto, Portugal (2005)
14. Benatallah, B., Sheng, Q.Z., Dumas, M.: The self-serv environment for Web services composition. IEEE Internet Comput. **7**(1), 40–48 (2003)
15. Bertino, E., Sandhu, R.: Database security-concepts, approaches, and challenges. IEEE Trans. Depend. Secure Comput. **2**(1), 2–9 (2005)
16. BPMI: Business Process Modeling Language (BPML): http://www.bpmi.org/bpml.esp (2003)
17. Bultan, T., Su, J., Fu, X.: Analyzing conversations of Web services. IEEE Internet Comput. **10**(1), 18–25 (2006)
18. Burstein, M., Bussler, C., Finin, T., Huhns, M., Paolucci, M., Sheth, A., Williams, S.: A semantic Web services architecture. IEEE Internet Comput. **9**, 52–61 (2005)
19. Bussler, C.: B2B Protocal Standards and their Role in Semantic B2B Integration Engines. IEEE Data Eng. Bull. **24**(1), 3–11 (2001)
20. Bussler, C.: B2B Integration: Concepts and Architecture. Sringer, Berlin Heidelberg New York (2003)
21. Bussler, C.: The role of semantic web technology in enterprise applicatin integration. Data Eng. Bull. **26**(4), 62–68 (2003)
22. Bussler, C., Fensel, D., Maedche, A.: A conceptual architecture for semantic Web enabled Web services. SIGMOD Rec. **31**(4), 24–29 (2002)
23. Cardoso, J.: Quality of service and semantic composition of workflows. Ph.D. Thesis, University of Georgia (2002)
24. Casati, F., Ilnicki, S., Jin, L., Krishnamoorthy, V., Shan., M.C.: Adaptive and dynamic service composition in eFlow. Technical report HPL-2000-39, Hewlett Packard, HP Laboratoris Palo Alto (2000)
25. Codd, E.: A relational model for large shared data banks. Commun. ACM **13**(6), 377–387 (1970)
26. Conti, M., Kumar, M., Das, S.K., Shirazi, B.A.: Quality of Service Issues in Internet Web Services. IEEE Trans. Comput. **51**(6), 593–594 (2002)
27. DAML: DAML-S (and OWL-S) 0.9 Draft Release. http://www.daml.org/services/daml-s/0.9/ (2004)
28. Ding, Y., Fensel, D., M. Klein, a.B.O.: The semantic Web: yet another hip? Data Knowl. Eng. **41**(3), 205–227 (2002)
29. Dogac, A., Kabak, Y., Laleci, G.: Enriching ebXML Registries with OWL ontologies for efficient service discovery. In: RIDE. Boston, USA (2004)
30. Dogac, A., Kabak, Y., Laleci, G.B., Mattocks, C., Najmi, F., Pollock, J.: Enhancing ebxml registries to make them owl aware. Distrib. and Parallel Databases J. **18**(1), (2005)
31. Dogac, A., Laleci, G.B., Kabak, Y., Cingil, I.: Exploiting web service semantics: Taxonomies vs. ontologies. IEEE Data Eng. Bull. **25**(4), 10–16 (2002)
32. Domingue, J., Galizia, S., Cabral, L.: Choreography in IRS-III - Coping with Heterogeneous Interaction Patterns in Web Services. In: ISWC, pp. 415–429. Galway, Ireland (2005)
33. Dustdar, S., Treiber, M.: A View Based Analysis on Web Service Registries. Distributed and Parallel Databases, pp. 147–171 (2005)
34. ebXML: http://www.ebxml.org (2003)
35. van Eck, P., Engelfriet, J., Fensel, D., van Harmelen, F., Venema, Y., Willems, M.: A survey of languages for specifying dynamics: a knowledge engineering perspective. IEEE Trans. Knowl. Data Eng. **13**(3), 462–496 (2001)
36. Edmond, D., Bouguettaya, A., Benatallah, B.: Formal correctness procedures for object-oriented databases. In: Proceedings of the 9th Australasian Database Conference. Perth, Australia (1998)

37. Fayyad, U.: Data mining and knowledge discovery in databases: implications for scientific databases. In: Proceedings of the Ninth International Conference on Scientific and Statistical Database Management, pp. 2–11 (1997)
38. Fensel, D.: Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce. Springer, Berlin Heidelberg New York (2001)
39. Fensel, D., Bussler, C.: The Web Service Modeling Framework WSMF. Electron Commerce Res Appl pp. 113–137 (2002)
40. Fensel, D., Harmelen, F., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F.: OIL: An ontology infrastructure for the semantic Web. IEEE Intell. Syst. **16**(2), 38–45 (2001)
41. Garcia-Molina, H.: The TSIMMIS Project: Integration of Heterogeneous Information Sources. J. Intell. Inf. Syst. **8**(2), 117–132 (1997)
42. Geer, D.: Taking steps to secure web services. IEEE Comput. **36**(10), 14–16 (2003)
43. Georgakopoulos, D., Schuster, H., Chichocki, A., Baker, D.: Managing process and service fusion in virtual enterprises. Inf. Syst. **24**(6), 429–456 (1999). http://dx.doi.org/10.1016/S0306-4379(99)00026-5
44. (GGF), G.G.F.: Web Services Agreement Specification: http://www.omg.org/mda/ (2005)
45. Gou, H., Huang, B., Liu, W., Ren, S., Li, Y.: Petri net based business process modeling for virtual enterprises. In: IEEE International Conference on Systems, Man, and Cybernetics, pp. 3183–3188. Nashville, United States (2000)
46. Gravano, L., Papakonstantinou, Y.: Mediating and metasearching on the Internet. IEEE Data Eng. Bull. **21**(2), 28–36 (1998)
47. Group, W.W.: Web Service Modeling Language (WSML). http://www.wsmo.org/wsml (2004)
48. Group, W.W.: Web Service Modeling Ontology (WSMO). http://www.wsmo.org/ (2004)
49. Hamadi, R., Benatallah, B.: A petri net-based model for web service composition. In: Proceedings of the 14th Australasian database conference on Database technologies, pp. 191–200. Australian Computer Society, Inc. (2003)
50. Heflin, J.: Towards the semantic Web: knowledge representation in a dynamic distributed environment. Ph.D. Thesis, University of Maryland (2001)
51. Hull, R., Su, J.: Tools for composite Web services: a short overview. SIGMOD Rec. **34**, 86–95 (2005)
52. IBM: Web Services Conceptual Architecture: http://www-306.ibm.com/software/solutions/webservices/pdf/WSCA.pdf
53. IBM: Web Services Flow Language (WSFL): http://xml.coverpages.org/ws.html (2003)
54. Kagal, L., Finin, T., Joshi, A.: A policy based approach to security on the semantic Web. In: International Semantic Web Conference. Florida, USA (2003)
55. Kagal, L., Paolucci, M., Srinivasan, N., Denker, G., Finin, T.W., Sycara, K.P.: Authorization and privacy for semantic Web services. IEEE Intell. Syst. **19**(4), 50–56 (2004)
56. Langdon, C.S.: The state of Web services. IEEE Comput. **36**(7), 93–94 (2003)
57. Laymann, F.: Jump onto the bus: a guided tour to the WS-* landscape. In: ICSOC (2003)
58. Maedche, A., Staab, S.: Ontology learning for the semantic Web. IEEE Intell. Syst. **16**(2), 72–79 (2001)
59. Malik, Z., Bouguettaya, A.: Preserving trade secrets between competitors in b2b interactions. Int. J. Cooperative Inf. Syst. **14**(2-3), 265–297 (2005)
60. Marchetti, C., Pernici, B., Plebani, P.: A quality model for multichannel adaptive information. In: WWW04. New York, USA (2004)
61. Martin, C.P., Ramamritham, K.: Recovery guarantees in mobile systems. In: Proceedings of the 1st ACM international Workshop on Data Engineering for Wireless and Mobile Access, pp. 22–28. ACM Press (1999). DOI http://doi.acm.org/10.1145/313300.313325
62. Maximilien, E.M., Singh, M.P.: A framework and ontology for dynamic web services selection. IEEE Internet Comput. **8**(5), 84–93 (2004)
63. Mcllraith, S.A., Martin, D.L.: Bringing semantics to Web services. IEEE Intell. Syst. **18**(1), 90–93 (2003)
64. Mcllraith, S.A., Son, T., Zeng, H.: Semantic Web services. IEEE Intell. Syst. **16**(2), 46–53 (2001)
65. Medjahed, B., Benatallah, B., Bouguettaya, A., Ngu, A.H.H., Elmagarmid, A.K.: Business-to-business interactions: issues and enabling technologies. VLDB J. **12**(1), 59–85 (2003). DOI http://dx.doi.org/10.1007/s00778-003-0087-z
66. Medjahed, B., Bouguettaya, A.: A multilevel composability model for semantic Web services. IEEE Trans. Knowl. Data Eng. (TKDE) **17**(7), 954–968 (2005)
67. Medjahed, B., Bouguettaya, A., Elmagarmid, A.K.: Composing Web services on the semantic Web. VLDB J. **12**(4), 333–351 (2003)
68. Medjahed, B., Rezgui, A., Bouguettaya, A., Ouzzani, M.: Infrastructure for E-government Web services. IEEE Internet Comput. **7**(1), 58–65 (2003)
69. Microsoft: Web Services Routing (WS-Routing): http://msdn.microsoft.com/
70. Microsoft: Web Services for Business Process Design (XLANG). http://xml.coverpages.org/xlang.html (2003)
71. Netscape: Secure Socket Layer (SSL) 3.0 Specification: http://wp.netscape.com/eng/ssl3/
72. Neves, N., Fuchs, W.K.: Adaptive recovery for mobile environments. Commun. ACM **40**(1), 68–74 (1997). DOI http://doi.acm.org/10.1145/242857.242878
73. OASIS: Universal Business Language (UBL): http://www.oasis-open.org/committees/ubl
74. OASIS: SAML: http://www.oasis-open.org/ (2004)
75. OASIS: WSS: http://www.oasis-open.org/ (2004)
76. O'Sullivan, J., Edmond, D., ter Hofstede, A.H.M.: Formal description of non-functional service properties. Technical report, Queensland University of Technology. http://www.servicedescription.com/ (2005)
77. Ouzzani, M., Bouguettaya, A.: Efficient access to web services. IEEE Internet Comput. **8**(2), 34–44 (2004)
78. Paolucci, M., Sycara, K.: Autonomous Semantic Web services. IEEE Internet Comput. **7**(5), 34–41 (2003)
79. Papazoglou, M.: Extending the service oriented architecture. Bus. Integr. J. **65**, 18–21 (2005)
80. Papazoglou, M., van den Heuvel, W.J.: Web services management: a survey. IEEE Internet Comput. **9**(6), 58–64 (2005)
81. Papazoglou, M.P.: Web services and business transactions. World Wide Web **6**(1), 49–91 (2003)
82. Papazoglou, M.P., Dubray, J.: A Survey of Web service technologies. Technical report DIT-04-058, University of Trento (2004)
83. Patel-Schneider, P.F., Siméon, J.: The Yin/Yang Web: A unified model for XML syntax and RDF semantics. IEEE Trans. Knowl. Data Eng. **15**(4), 797–812 (2003)
84. Peltz, C.: Web services orchestration and choreography. IEEE Comput. **36**(10), 46–52 (2003)
85. Petrie, C., Bussler, C.: Service agents and virtual enterprises: a survey. IEEE Internet Comput. **7**(4), 68–78 (2003)

86. Pilioura, T., Tsalgatidou, A., Hadjiefthymiades, S.: Scenarios of using web services in m-commerce. SIGecom Exch. **3**(4), 28–36 (2003). DOI http://doi.acm.org/10.1145/844351.844356
87. Prakash, R., Singhal, M.: Low-cost checkpointing and failure recovery in mobile computing systems. IEEE Trans. Parallel Distrib. Syst. **7**(10), 1035–1048 (1996). DOI http://dx.doi.org/10.1109/71.539735
88. Ran, S.: A model for Web services discovery with QoS. ACM SIGecom Exchanges **4**(1), 1–10 (2003)
89. Rezgui, A., Bouguettaya, A., Eltoweissy, M.Y.: Privacy on the Web: facts, challenges, and solutions. IEEE Sec. Privacy **1**(6), 40–49 (2003)
90. Rezgui, A., Bouguettaya, A., Malik, Z.: A Reputation-based approach to preserving privacy in Web services. In: VLDB Workshop on Technologies for E-Services (TES). Berlin, Germany (2003)
91. Rezgui, A., Ouzzani, M., Bouguettaya, A., Medjahed, B.: Preserving privacy in Web services. In: Proceedings of the 4th ACM Workshop on Information and Data Management (WIDM'02). McLean, VA (2003)
92. Singh, M.P.: Physics of service composition. IEEE Internet Comput. **5**(3), 6 (2001)
93. Singh, M.P., Huhns, M.N.: Service-Oriented Computing Semantics, Processes, Agents. Wiley, New York (2005)
94. Sirin, E., Hendler, J., Parsia, B.: Semi-automatic composition of web services using semantic descriptions. In: Web Services: Modeling, Architecture and Infrastructure Workshop in Conjunction with ICEIS2003 (2003)
95. Sirin, E., Parsia, B., Hendler, J.: Filtering and selecting semantic web services with interactive composition techniques. IEEE Intell. Syst. **19**(4), 42–49 (2004). URL http://www.mindswap.org/papers/IEEE-IS-04.html
96. Skonnard, A.: Understanding WS-Policy. Technical report, Skonnard Consulting: http://www.microsoft.com/webservices/ (2003)
97. Stonebraker, M., Wong, E., Kreps, P., Held, G.: The design and implementation of ingres. TODS **1**(3), 189–222 (1976)
98. SUN: Web Services Reliability (WS-Reliability). http://developers.sun.com/sw/platform/technologies/ws-reliability.html
99. Tagg, R.: Workflow in different styles of virtual enterprise. In: Workshop on Information technology for Virtual Enterprises, pp. 21–28. Queensland, Australia (2001)
100. Tsalgatidou, A., Pilioura, T.: An overview of standards and related technology in Web services. Distrib. Parallel Databases **12**(2), 135–162 (2002)
101. Tumer, A., Dosac, A., Toroslu, H.: A semantic based privacy framework for web services. In: WWW'03 Workshop on E-Services and the Semantic Web (ESSW '03). Budapest, Hungary (2003)
102. Vaughan-Nichols, S.J.: Web services: beyond the hype. IEEE Comput. **35**(2), 18–21 (2002)
103. Venkatesh, V., Ramesh, V., Massey, A.P.: Understanding usability in mobile commerce. Commun. ACM **46**(12), 53–56 (2003). DOI http://doi.acm.org/10.1145/953460.953488
104. Vinoski, S.: Web services interaction models, part 1: current practice. IEEE Internet Comput. **6**(3), 89–91 (2002)
105. W3C: Web Service Execution Environment (WSMX): http://www.w3.org/Submission/WSMX/
106. W3C: Web Services Addressing (WS-Addressing): http://www.w3.org/Submission/ws-addressing/
107. W3C: XML Encryption. http://www.w3.org/Encryption/ (2001)
108. W3C: XML Signature. http://www.w3.org/Signature/ (2001)
109. W3C: Simple Object Access Protocol (SOAP). http://www.w3.org/TR/SOAP/ (2003)
110. W3C: Universal Description, Discovery, and Integration (UDDI). http://www.uddi.org (2003)
111. W3C: Web Service Choreography Interface (WSCI). http://www.w3.org/TR/wsci/ (2003)
112. W3C: Web Services Architecture. http://www.w3.org/TR/ws-arch/ (2003)
113. W3C: Web Services Description Language (WSDL). http://www.w3.org/TR/wsdl (2003)
114. W3C: The Platform for Privacy Preference Specification (P3P). http://www.w3.org/TR/P3P11/ (2004)
115. W3C: Web Services Choreography Description Language (WS-CDL). http://www.w3.org/TR/ws-cdl-10/ (2004)
116. Westin, F.D.: Philosophical Dimensions of Privacy: An Anthology. Cambridge University Press, Cambridge (1984)
117. Workflow Management Coalition: workflow management application programming interface (interface 2&3) specification. Document number WFMC-TC-1009 (1998), Version 2.0
118. WS-I: Web Services Interoperability Organization. http://www.ws-i.org/
119. Yang, X., Bouguettaya, A.: Adaptive data access in broadcast-based wireless environments. IEEE Trans. Knowl. Data Eng. **17**(3), 326–338 (2005)
120. Yang, X., Bouguettaya, A., Medjahed, B., Long, H., He., W.: Organizing and accessing web services on air. IEEE Trans. Syst. Man Cybern. Part A Syst. Hum. **33**(6), 742–757 (2003)
121. Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnanam, J., Chang, H.: Qos-aware middleware for web services composition. IEEE Trans. Softw. Eng. **30**(5), 311–327 (2004)